Eine visuelle Sprache zur Definition räumlicher (ebener) Konstellationen

Michael Wessel

26. Februar 1998

Diplomarbeit
am Fachbereich Informatik
der Universität Hamburg

Erstbetreuer:
Dr. Volker Haarslev
Arbeitsbereich Kognitive Systeme

Zweitbetreuer:
Prof. Dr. Christian Freksa
Arbeitsbereich Wissens- und
Sprachverarbeitung

Inhaltsverzeichnis

Kapitel 1

Einleitung und Übersicht

1.1 Thema der Arbeit

Thema dieser Arbeit ist die Entwicklung und Implementation einer visuellen Sprache zur Definition räumlicher Konstellationen, genannt VISCO. Eine *visuelle Sprache* ist eine Sprache, die in erster Linie nicht textuelle, sondern grafische Sprachelemente auf systematische Weise verwendet, um komplexe Bedeutungen zu kommunizieren.[1] VISCO ist ein Akronym für „Vivid Spatial Constellations“, wobei das Attribut „Vivid“ Vagheit und „Lebendigkeit“ der definierten Konstellationen ausdrücken soll. Eine Konstellation wird hier als Zusammenstellung von individuellen räumlichen Objekten betrachtet – diese Objekte könnten dabei je nach Anwendungskontext unterschiedliche Bedeutungen tragen. Da VISCO ausschließlich zur Definition zweidimensionaler Konstellationen geeignet ist, sollte man vielleicht passender von ebenen Konstellationen sprechen. Räumliche Konstellationen sind in vielen Informatikforschungsbereichen von Interesse, so z.B. in der Geoinformatik, im CAD/CAM-Bereich und in Teilbereichen der Künstlichen Intelligenz (KI).

Die Sprachelemente der visuellen Sprache VISCO können nun zur Bildung ebener Konstellationen verwendet werden, welche dann ihrerseits ebene Konstellationen beschreiben: Interessant scheint die Frage, welche Vor- und Nachteile sich dadurch ergeben, daß hier Raum (in gewisssen Aspekten) direkt durch Raum beschrieben wird. Eine treffende Allegorie für diesen Sachverhalt stammt von Meyer ([Mey92]): „Pictures depicting pictures“. In diesem Zusammenhang werden daher sog. *analogische Repräsentationen* räumlicher Konzepte von Relevanz sein (s. Kap. 2). Es wird deutlich, daß es sich bei VISCO um eine Art Metasprache handeln muß, denn räumliche Konzepte werden hier durch räumliche Konzepte beschrieben. Ebenso wie die natürliche Sprache (z.B. Deutsch) ihre eigene Metasprache ist (da mit ihr über Sprache selbst geredet werden kann), wird hier ein Versuch unternommen, eine Art Metasprache für ebene Konstellationen zu entwickeln.

Die im folgenden eingenommene Sichtweise ist relativ abstrakt, als daß sie sich zunächst nicht an den pragmatischen Erfordernissen spezieller Anwendungskontexte orientiert. Die entwickelte Sprache ist daher in erster Linie *Selbstzweck* – das verwendete Vorgehen kann als *synthetisch* bezeichnet werden. Dies mag auf den ersten Blick dilettantisch erscheinen: Ich rechtfertige das Vorgehen damit, daß die im Rahmen dieser Arbeit verwendeten räumlichen Konzepte in gewisser Weise generisch sind – sie finden sich in vielen raumbezogenen Anwendungskontexten wieder. Dies gilt sowohl für die verwendeten geometrischen Objekte, wie Punkte, Strecken, Polygone usw., als auch für die betrachteten räumlichen Eigenschaften und Beziehungen zwischen diesen. Bei Objekten wie Punkten, Strecken und Polygonen handelt es sich um fundamentale Abstraktionen räumlicher Gegebenheiten. Interessante Eigenschaften dieser sind z.B. „Form“, „Orientierung“,

[1] Obwohl es auch Definitionen dieses Begriffes gibt, nach denen auch textuelle Sprachen als visuell charakterisiert werden (s. Kap. 2), werden im Rahmen dieser Arbeit ausschließlich visuelle Sprachen betrachtet, deren Sprachelemente nicht-textuell sind!

„Länge“ und „Position“, während interessante Relationen u.a. „enthält“, „enthalten in“, „schneidet“, „disjunkt“, „im Abstand von n Metern“ etc. sind. Offensichtlich ist die Ebene der Träger dieser Informationen (z.B. könnte ein Blatt Papier das Repräsentationsmedium darstellen), denn bei geeigneter Interpretation könnten anhand der Geometrie einer Konstellation dieser Objekte all diese intrinsisch repräsentierten Eigenschaften und Relationen explizit gemacht werden. Analogische Repräsentation bieten (gegenüber propositionalen Repräsentationen) den Vorteil, daß evtl. aufwendige Deduktions- bzw. Explizierungsprozesse durch „einfaches“ Nachsehen (Inspektion) – sog. *perzeptuelle Inferenzen* – ersetzt werden können (s. Kap. 2).

Die semantischen und pragmatischen Dimensionen der betrachteten räumlichen Objekte werden in dieser Arbeit (größtenteils) ignoriert – lediglich die räumlichen Aspekte werden als relevant betrachtet. In der Literatur zum Thema „Geographische Informationssysteme“ (s. Kap. 3) wird oftmals betont, daß die Ebene bzw. der Raum selbst den Integrationsrahmen für unterschiedlichste Datensätze darstellt (s. z.B. [Fra93]) – der Raumbezug dieser Daten ist der gemeinsame verbindende Faktor unterschiedlichster Datensätze. So erscheint es naheliegend, in einem weiteren Schritt alle pragmatischen Aspekte zu ignorieren und sich ausschließlich auf die Repräsentation und Verarbeitung des Raumes (der Ebene) bzw. räumlicher Konzepte selbst zu beschränken. Diese Betrachtungsweise wird u.a. im Rahmen der sog. *räumlichen Informationstheorie (Spatial Information Theory)* eingenommen, wobei es sich um eine Art Grundlagenwissenschaft für den Bau Geographischer Informationssysteme handelt (vergleichbar der Geoinformatik, s. Kap. 3). In der KI kommt insbesondere noch die Untersuchung von Inferenzmethoden hinzu, wie sie z.B. im Rahmen des *qualitativen räumlichen Schließens (Qualitative Spatial Reasoning)* entwickelt werden – auch hierbei handelt es sich um theoretische Grundlagenforschung.

Die Anwendungsunabhängigkeit wird auch durch den Begriff des „Geometrischen Informationssystems“ ([KVV97, S. 76]) zum Ausdruck gebracht:

> Unter dem Begriff „Geometrische Informationssysteme“ (GI) verstehen wir in Anlehnung an Geographische Informationssysteme (GIS) Systeme, die Repräsentation, Verarbeitung und Darstellung von strukturierten räumlichen Informationen ermöglichen, ohne jedoch die geospezifischen Eigenschaften von GIS-en aufzuweisen. Zum Funktionsumfang von GI-en gehören damit allgemein Strukturen und Prozesse zur Darstellung und Verarbeitung räumlichen Wissens, wie sie beispielsweise im Bereich des geometrischen Schließens oder des qualitativen räumlichen Schließens untersucht werden.

In der Literatur findet man zudem den verwandten Begriff „Räumliches Informationssystem“ (Spatial Information System).[2] Diese Begriffe stellen (ebenso wie der Begriff des GI) *Generalisierungen des GIS-Konzeptes* dar, so daß der Fokus auf allgemeineren Fragen der Repräsentation und Verarbeitung räumlicher Daten liegt. So schreibt z.B. Egenhofer in [Ege90]:

> Such a computer system, dealing with information about the geometry and location of spatial data and its relation to other objects, is called a spatial information system, and the data collected in this systems is often referred to as the spatial database. ... The desire to analyze various combinations of spatial data and to reveal otherwise hidden information is the justification for spatial information systems.

VISCO wird in erster Linie als Sprache zur Definition räumlicher Konstellationen angesehen. Die Nützlichkeit und Verwendbarkeit der Sprache wurde im Rahmen eines speziellen Anwendungskontextes überprüft, indem VISCO als visuelle räumliche Anfragesprache auf den Daten der „Digitalen Stadtkarte (DISK)“ des Hamburger Vermessungsamtes eingesetzt wurde. Ein wesentlicher Anteil dieser Diplomarbeit besteht somit in der prototypischen Implementierung des VISCO-Systemes (s. Kap. 6). VISCO kann auch als visuelle räumliche Anfragesprache für Geometrische Informationssysteme bezeichnet werden.

[2] Die Begriffe „Räumliches Informationssystem“ und „Rauminformationssystem“ sind nicht synonym – Rauminformationssysteme sind spezielle GIS.

1.2 Struktur der Arbeit

Zunächst soll in Kap. 2 ein Rundumblick auf einige Informatikbereiche getätigt werden, in denen räumliche Konzepte von Interesse sind. Hierzu gehören u.a. *spezielle KI-Forschungsgebiete* und der für diese Arbeit sehr wichtige Bereich der *visuellen Sprachen.* Einige wichtige im weiteren Verlauf der Arbeit *verwendete Begriffe sollen präzisiert werden,* worin der Hauptzweck des Kapitels besteht. Natürlich kann hier nur ein sehr grober Überblick vermittelt werden, der keinen Anspruch auf Vollständigkeit und Tiefe erhebt.

In Kapitel 3 sollen einige wesentliche Aspekte der *Geoinformatik* dargestellt werden – wie auch in Kap. 2 soll hier lediglich ein grober Überblick vermittelt werden. Die im Rahmen der Geoinformatik interessierenden Konstellationen werden von *Geo-Objekten* gebildet, wobei es sich um Abstraktionen geographischer Objekte (z.B. Flüsse, Seen, Wälder etc.) handelt. Aber auch „virtuelle“ Objekte sind – sofern sie einen Bezug zur Erdoberfläche haben – von Interesse (z.B. Länder, Ländergrenzen, Grundstücksgrenzen, Buslinien, etc.). Im Rahmen dieser Arbeit interessieren vor allem Geographische Informationssysteme (GIS), denn VISCO könnte (in einigen Aspekten) als visuelle räumliche Anfragesprache für GIS verwendet werden. Dies wird in Kap. 6 und Kap. 7 mit Hilfe der Daten der Digitalen Stadtkarte (DISK) des Hamburger Vermessungsamtes demonstriert.

In Kap. 4 werden einige in der Literatur zu findende visuelle räumliche Anfragesprachen für GIS diskutiert. Auffallend ist die geringe Anzahl bisher implementierter Systeme bzw. Prototypen. Durch die Diskussion können einige wesentliche Ideen sowie Vor- und Nachteile derartiger Anfragesprachen extrahiert werden.

Der informellen Vorstellung der Sprache VISCO ist Kap. 5 gewidmet. Hier werden auch einige Beispiele vorgestellt.

In Kap. 6 wird die Implementation des VISCO-Prototypen diskutiert: hierbei handelt es sich um ein System, welches die Nutzung der Sprache VISCO als visuelle räumliche Anfragesprache auf den Daten der DISK erlaubt. Der Prototyp besteht aus einer Umgebung, die eine Inspektionskomponente, einen syntaxgesteuerten Grafikeditor zum Konstruieren visueller Anfragen, einen optimierenden Compiler zum Übersetzen der Anfragen und eine Ergebnisinspektionskomponente integriert. Tatsächlich ist es nicht sehr sinnvoll, VISCO losgelöst von einer mächtigen Sprachumgebung bzw. Benutzungsoberfläche zu betrachten (dies gilt übrigens für die meisten visuellen Sprachen – die Benutzungsoberfläche hat hier einen *sehr* entscheidenen Einfluß, s. Kap. 2). Die Diskussion von Details geschieht auf einer Ebene, die keine LISP-, CLOS- oder CLIM-Kentnisse erfordert (hierbei handelt es sich um die verwendeten Sprachen bzw. Frameworks).

In Kap. 7 wird die Benutzung des implementierten Prototypen anhand eines einfachen Beispieles dargestellt.

Schließlich soll Kap. 8 für eine kurze kritische Bewertung der erzielten Ergebnisse genutzt werden. Detailfragen und -probleme werden jedoch im Verlauf der Arbeit stets an Ort und Stelle diskutiert, so daß in diesem Kapitel eine übergeordnete Perspektive eingenommen wird.

Im Anhang wird der Versuch unternommen, eine formale Sprachdefinition für VISCO anzugeben.

Die Reihenfolge der Kapitel stellt in gewisser Weise auch den zeitlichen Werdegang dieser Arbeit dar. Während die Kap. 2, 3 und 4 mit den Attributen „breit und oberflächlich“ umschrieben werden können – schließlich geht es hier primär um die Orientierung in einer komplexen Forschungslandschaft –, können die Kap. 5 bis 8 und der Anhang durch „speziell und tiefgehend“ charakterisiert werden, da sie den eigentlichen Beitrag dieser Arbeit darstellen.

Ich halte es für wichtig, daß der Leser die Arbeit unter dem richtigen Blickwinkel betrachtet: für die Implementierung des Prototypen wurden ca. 70 % der Gesamtbearbeitungszeit verwendet. Es handelt sich also um eine *praktische* Diplomarbeit, in der Fragen bzgl. Design und Implementierung eines komplexen Softwaresystemes vordergründig sind.

Ich möchte darauf hinweisen, daß in dieser Arbeit die alte Rechtschreibung verwendet wird ([Dud86]).

1.3 Danksagungen

In erster Linie danke ich Dr. Volker Haarslev für seine ständige Gesprächsbereitschaft – er hatte stets ein offenes Ohr, auch für unfertige oder gewagte Ideen. Zudem hat er einen großen Anteil an der Entwicklung von VISCO: diverse Sprachelemente und -mittel nahmen erst im Laufe der Diskussion mit ihm konkretere Form an. Seinem Engagement sind auch die Publikationen [HW96] und [HW97] zu verdanken.

Dr. Ralf Möller danke ich für seine stetige, kompetente und freundliche Beratung in Sachen LISP, CLOS und vor allem CLIM sowie den Kaffee an den Wochenenden im KOGS-Labor. Auch er hat viele wichtige Ideen – insbesondere zur Gestaltung der Benutzungsoberfläche für VISCO – beigesteuert. Auch für seine moralische Unterstützung möchte ich mich bedanken.

Prof. Dr. Christian Freksa möchte ich für die Übernahme der Rolle des Zweitbetreuers, sein Interesse an der Arbeit und die Zeit in seinen Sprechstunden danken.

Prof. Dr. Bernd Neumann danke ich für sein Interesse an der Arbeit, das er insbesondere auch in einer Systemvorführung des Prototypen bekundete.

Nicht zuletzt danke ich meinen Eltern, daß sie mir das Informatikstudium ermöglichten.

Schließlich sei noch dem Hamburger Vermessungsamt für einen Auszug der DISK (Digitalen Stadtkarte) gedankt (insbesondere Fr. Wagener und Hr. Matthias).

Kapitel 2

Relevante Forschungsgebiete

Thema dieser Diplomarbeit ist die „Entwicklung und Implementation einer visuellen Sprache zur Definition ebener Konstellationen“: die Bedeutung dieses – zunächst sehr vagen – Satzes soll zuerst etwas genauer analysiert werden. Die so identifizierten relevanten Aspekte möchte ich im weiteren Verlauf des Kapitels etwas ausführlicher darstellen. Andere im Verlauf der Arbeit verwendete Termini sollen ebenfalls präzisiert werden, u.a. die Begriffe „Repräsentation“ und „analogische Repräsentation“; statt von „Repräsentation“ rede ich auch manchmal von „Modellierung“. Ich halte eine begriffliche Klärung für wichtig, damit nicht der Eindruck entsteht, ich würde Begriffe unreflektiert verwenden.

Räumliche Konstellationen und Konzepte werden u.a. von der Mathematik, der KI und auch der Kognitionswissenschaft untersucht – aus diesen Bereichen werden einige theoretische Grundlagen dargestellt, die sich im Werdegang der Arbeit als relevant erwiesen haben. Besondere Aufmerksamkeit wird den visuellen Sprachen gewidmet.

2.1 Terminologie

Thema dieser Diplomarbeit ist die „Entwicklung und Implementation einer visuellen Sprache zur Definition ebener Konstellationen“: für den Begriff „Definition“ (lat. Abgrenzung) findet man in der Literatur u.a. Definitionen wie „Festlegung und Beschreibung eines Begriffes“ ([Dud94]) und „Begriffsbestimmung, eindeutige begriffl. Fixierung eines Sachverhaltes.“ ([Fis79]).

Unter einem „Begriff“ versteht man dabei „eine Vorstellung, die im Gegensatz zur Anschauung mehrere Merkmale oder Gegenstände zu einer Einheit zusammenfaßt“ ([Fis79]). Offensichtlich ist eine *Vorstellung* zunächst etwas subjektives, geistig-mentales und somit schlecht objektivierbar – soll über sie kommuniziert werden, so werden u.a eben sprachliche Definitionen benötigt. Begriffe werden nun durch Abstraktion von realen Gegebenheiten gewonnen: man unterscheidet u.a. zwischen Individual- und Klassenbegriffen. Neue Begriffe können jedoch auch mental gebildet werden (z.B. durch Kombination bereits bekannter anderer Begriffe) und dann mittels Sprache kommuniziert werden. Komplexe sprachliche Ausdrücke referieren bzw. bezeichnen dann wieder entsp. Objekte, Vorgänge etc. in der Welt.

Im Rahmen dieser Arbeit sind vor allem räumliche Klassenbegriffe relevant: eine interessante Vorstellung könnte sprachlich z.B. durch „gleichseitiges Rechteck“ umschrieben bzw. definiert werden (wobei es sich um einen Klassenbegriff handelt): diese Abstraktion nennen wir „Quadrat“. Eine angemessene Beschreibung eines interessanten Individualbegriffes ist in Abb. 2.1 dargestellt: diese Beschreibung steht in gewisser Weise für sich selbst – man spricht auch von Eigennamen. Hier wird deutlich, daß eine Umschreibung oder Definition eines Begriffes nicht unbedingt in natürlicher Sprache erfolgen muß. So bieten *visuelle Sprachen* die Möglichkeit, Begriffe in gewisser Weise

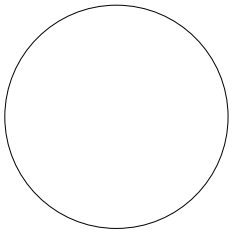


Abbildung 2.1: Ein Kreis mit $r = 1,5$ cm

„nichtsprachlich“ zu beschreiben. Im weiteren Verlauf der Arbeit wird insbesondere die Frage, wie man mit grafisch-visuellen Mitteln (wie in Abb. 2.1) Klassenbegriffe (und nicht nur Individualbegriffe) beschreiben kann, noch von Bedeutung sein.

Im Englischen wird für das Wort „Begriff“ das Wort „Concept“ verwendet – im Deutschen hat das aus dem lat. stammende Wort „Konzept“ lt. [Fis79, Dud86] jedoch die Bedeutung „Entwurf; erste Fassung; Rohschrift“. Speziell in der Künstlichen Intelligenz (KI) werden jedoch auch im Deutschen die Begriffe „Begriff“ und „Konzept“ synonym verwendet: offensichtlich dienen dann Definitionen zur „Festlegung und Beschreibung von Konzepten“. Eine „Beschreibung“ setzt offensichtlich eine Sprache (z.B. ein visuelle Sprache) voraus, wie dies oben schon deutlich wurde.

Für den Begriff des „räumlichen Konzeptes“ (Spatial Concept) findet man bei Freksa und Barkowsky ([FB95]) folgende Umschreibung:

> When we consider real world object, we usually are interested in certain properties of these objects, i.e., we regard the objects under certain aspects. For example, when we take a look at a geographic entity, say a lake, we regard it with respect to horizontal extension, depth, shape, or the like. All these notions that describe spatial aspects of a subset of the world, we call spatial concepts. ... Concepts can be anything we have a notion of; thus, „size“ can be a concept and „big“ can be a concept as well.

Im folgenden werden insbesondere *geometrische* und *topologische* räumliche Konzepte eine Rolle spielen: solche Raumkonzepte sind z.B. „Form“, „Orientierung“, „Position“, „Größe“, „Verbundenheit“, „Enthaltensein“, „Rechts von“ (Präpositionen), etc. Aber auch *strukturelle* Konzepte wie „Konstellation“, „Aggregat“, „Endpunkt von“ etc. werden eine Rolle spielen. Somit können sowohl Objekte als auch Eigenschaften dieser Objekte und Relationen bzw. Beziehungen zwischen diesen als räumliche Begriffe (und somit Raumkonzepte) bezeichnet werden.

An dieser Stelle bietet nun die *Semiotik („Zeichenlehre“)* die Möglichkeit, einige weitere wichtige Begriffe zu klären. Zur Klärung dieser Begriffe soll das Semiotische Dreieck verwendet werden (Abb. 2.2, aus [Sch91], [Sch87]): es wurde bereits deutlich, daß Begriffe durch Abstraktion (gewisser Aspekte) von Objekten einer Welt gewonnen werden – der entsp. Prozeß heißt *Begriffsbildung* bzw. *Begreifen.* Diese Begriffe können nun durch *symbolische Strukturen (Zeichen, Wörter und Sätze einer Sprache)* bezeichnet, beschrieben, definiert oder auch *repräsentiert* werden (es gibt Unterschiede zwischen diesen Begriffen, s. [Sch87]). Komplexe symbolische Strukturen einer Sprache werden Sätze genannt – eine *Repräsentation* ist also eine Beziehung zwischen mentalen Begriffen bzw. Konzepten und Sätzen einer Sprache. Wenn keine Verwechslungsgefahr besteht, werde ich auch manchmal den Begriff der Repräsentation für die symbolischen Strukturen *selbst* verwenden. Die symbolischen Strukturen sind selbst wieder (komplexe) Objekte einer (symbolischen) Welt und werden auch *Repräsentanten* genannt. Oftmals spricht man von einem *Repräsentationsformat,* wenn man strukturelle bzw. syntaktische Eigenschaften dieser Objekte betonen will. Durch *Interpretation* der symbolischen Strukturen kann die *Bedeutung (Semantik)* in Form von Begriffen mental beim Interpreten *rekonstruiert* werden. In der Regel sagt man jedoch, daß Objekte der wirklichen Welt Begriffe interpretieren – wird also von „Interpretation“ in der Rolle der „Bedeutung“ gesprochen, so wird damit zum Ausdruck gebracht, daß die symbolischen Ausdrücke der Sprache als bedeutungstragende Objekte selbst betrachtet werden.

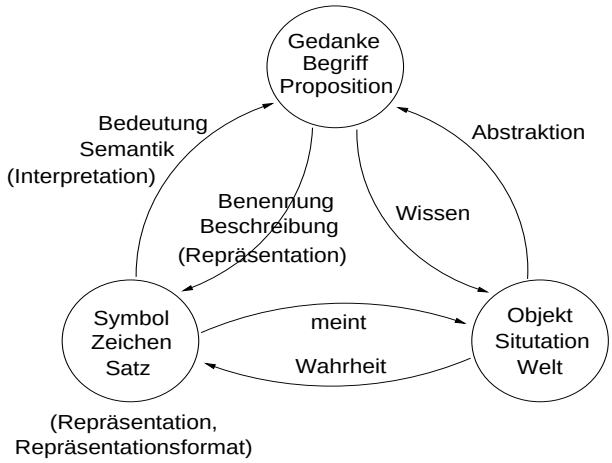


Abbildung 2.2: Das Semiotische Dreieck (reproduziert nach [Sch91])

Schließlich soll nach geklärt werden, daß im Kontext diese Arbeit unter einer *räumlichen (ebenen) Konstellation* eine Zusammenstellung bzw. Anordnung von individuellen räumlichen bzw. ebenen Objekten im Raum bzw. der Ebene verstanden wird.

2.2 Die Bedeutung von Raumkonzepten und räumlichen Metaphern in der Informatik

Die Bedeutung räumlichen Wissens (und somit die Relevanz räumlicher Konzepte) wird von vielen Forschern in der Allgegenwart räumlicher Konzepte gesehen: wir alle *leben im Raum* und müssen uns somit zwangsläufig mit räumlichen Gegebenheiten auseinandersetzen. So gibt Habel in [Hab87] auf die Frage, wo denn räumliches Wissen bzw. Wissen über Raum überhaupt eine Rolle spiele, folgende Antwort:

> (Fast) überall, denn menschliches Handeln (und somit u.a. Kommunizieren, Planen, etc.) ist in Raum und Zeit verankert.

Daß räumliche Konzepte in der Informatik bisher wenig Beachtung gefunden haben, liegt lt. Habel in erster Linie an den bisher realisierten bzw. realisierbaren Anwendungen: so ist es z.B. in nahezu allen geschäftsorientierten Informatikanwendungen (Logistik, Lagerverwaltung, Personalinformationssysteme etc.) möglich, von den räumlichen Eigenschaften der repräsentierten Objekte vollständig zu abstrahieren. In folgenden Anwendungsbereichen ist diese Abstraktion jedoch nicht möglich:

- **Robotik:** zweifelsfrei benötigen *autonome Roboter* – aus den gleichen Gründen wie auch der Mensch – räumliche Inferenzkompetenz.
- **Expertensysteme:** *Konfigurierungsprobleme* bestehen u.a. darin, komplexe räumliche Konstellationen zusammenzustellen, z.B. die Inneneinrichtung eines Flugzeuges.
- **Bildverarbeitung:** hier sollen u.a. räumliche Objekte *klassifiziert* bzw. anhand ihres Erscheinungsbildes erkannt werden – dies setzt natürlich entsp. repräsentiertes räumliches Wissen voraus.
- **Computergrafik:** in *CAD/CAM-Anwendungen* müssen komplexe geometrische Objekte repräsentiert, visualisiert und nicht zuletzt manipuliert werden.

Raum ist jedoch nicht gleich Raum: so identifiziert das Lexikon [Fis79] geometrischen, physikalischen, philosophischen und psychologischen Raum.

Die Bedeutung von Raumkonzepten in der menschlichen Kognition (psychologischer Raum) wird u.a. innerhalb der Kognitionsforschung und kognitiven Psychologie untersucht. In der sog. *Imagery-Debatte* wurde kontrovers diskutiert, wie und ob menschliches Wissen über räumliche Gegebenheiten in gewisser Weise bildhaft repräsentiert wird oder nicht. Ausgangspunkt hierfür war die Beobachtung des Auftauchens „mentaler Bilder“ beim Lösen von Problemen. Es könnte sich jedoch auch um Epiphänomene handeln.

Auch im Bereich des kognitiv orientierten qualitativen räumlichen Schließens wird darauf hingewiesen, daß räumliche Konzepte im menschlichen Denken eine herausragende Rolle spielen. So schreiben Freksa und Zimmermann in [FZ92]

> Our research on spatial representations and reasoning is motivated by the intuition that ‘dealing with space’ should be viewed as cognitively more fundamental than abstract reasoning. Afterall, one of the very first tasks we learn to accomplish is to orient ourselves in the environment.

Schließlich wird von vielen Informatikanwendungsdomänen versucht, die von der Evolution in langer Zeit entwickelten menschlichen Fähigkeiten im Umgang mit Raum und räumlichen Konzepten nutzbar zu machen, vor allem für Zwecke der Mensch-Maschine-Kommunikation (HCI, Human Computer Interaction). Offensichtlich profitieren hiervon Anwendungen wie virtuelle Realität (Virtual Reality), visuelle Sprachen und nicht zuletzt Geographische Informationssysteme (GIS). So schreibt z.B. Kuhn in einer Abhandlung über die Nutzbarmachung räumlicher Konzepte durch sog. *räumliche Metaphern* für Benutzungsoberflächen für GIS ([Kuh96]):

> Space is fundamental to perception and cognition because it provides a common ground for our senses as well as for our actions. The constant mutal reinforcement of visual, auditory and tactile cues has developed our spatial cognition to an extent unmatched by any other domain. Perception, manipulation, and motion in space are largely subconscious activities, imposing little cognitive load, while offering intuitive inference patterns. Space has a strong inner coherence that proves useful for designing and combining metaphors. For example, we know that opening a door gets us to a room or building, not to a desktop or a road, and that we are likely to encounter other doors and windows behind. *Thus, space is not just any domain of experience, but the most important one and as such uniquely qualified as a metaphor source. (Im Original nicht kursiv, d. V.)*

Kuhn spricht in diesem Zusammenhang von der „Verräumlichung von Benutzungsoberflächen“ (Spatialization of User Interfaces).

Daß solche *räumlichen Metaphern* bereits Einzug in die Informatik gehalten und beträchtliche Erfolge vorzuweisen haben, sieht man u.a. an den allgegenwärtigen *grafischen Benutzungsoberflächen.* Die Relevanz von Metaphern wird insbesondere im HCI-Bereich unterstrichen (auch in dieser Arbeit werden sie noch eine wichtige Rolle spielen): während das Lexikon ([Fis79]) eine Metapher als „rhetorisches Stilmittel, das einen (meist abstrakten) Sachverhalt bildlich-anschaulich wiedergibt“ definiert, wird im HCI-Bereich in Metaphern ein mächtiges Mittel zur Verkürzung der *Distanz* zwischen der mentalen Vorstellungswelt des Benutzers und dem internen formalen Weltmodell des Rechners gesehen. Eine tragfähige Definition für den Begriff „Metapher“ stammt von Lakoff und Johnson ([LJ80], aus [Com94]):

> *[A metaphor is]* a rhetoric figure whose essence is understanding and experiencing one kind of thing in terms of another *(Hinzufügungen in Kursivschrift, d.V.)*

Die Bedeutung von Metaphern wird in [Com94] folgendermaßen charakterisiert:

> Methapors play the key role in mapping knowledge from a domain in which the user is conversant into a new one, the one of the application so as to help him/her in understanding the system, its features and commands.

Durch die *Simulation* einer räumlichen Umgebung oder die Verwendung räumlicher Metaphern wird es dem Benutzer also ermöglicht, sich in einer abstrakten, ansonsten unbekannten Welt zu orientieren und somit in gewisser Weise heimisch zu fühlen. Metaphern ermöglichen es ihm, seine im Umgang mit der realen Welt erworbenen Fähigkeiten nun zur effizienten Bedienung des Systemes zu nutzen. Eine ausführlichere Diskussion findet man in [Lev90], Überlegungen im GIS-Kontext in [Mar93, Kuh96].

2.3 Theoretische Grundlagen

2.3.1 Palmers Repräsentationstheorie

Die allgemeine Repräsentationstheorie von Palmer ([Pal78]) emöglicht die Klärung des Begriffes der „Repräsentation“ – während im Semiotischen Dreieck eine Repräsentation als Korrespondenz zwischen mentalen Konzepten (Begriffen) und symbolischen Strukturen (Zeichen, Sätzen) einer Sprache definiert wurde, ist eine Repräsentation lt. Palmer eine Korrespondenz zwischen zwei Welten: der *repräsentierten Welt* und der *repräsentierenden Welt.* Die repräsentierte Welt beinhaltet also die „Konzepte“ oder „Begriffe“, die dann in Objektstrukturen der repräsentierenden Welt abgebildet werden. Obwohl Palmer seine Repräsentationstheorie ursprünglich zur Klärung des Repräsentationsbegriffes in der kognitiven Psychologie entwickelt hat, läßt sie sich aufgrund ihrer Allgemeinheit auch in der Informatik verwenden. Interessant ist, daß der Palmersche Begriff der Repräsentation nicht unbedingt etwas mit „mentalen Konzepten“ zu tun hat: es handelt sich nämlich einfach um eine Abbildung zwischen „verschiedenen Welten“.

Laut Palmer müssen bzgl. einer Repräsentation folgende Fragen geklärt werden (die Darstellung ist [Reh90] entnommen):

1. Was ist die repräsentierte Welt $\mathcal{W}_1$? Welches sind ihre Objekte $\mathcal{O}_1$?
2. Was ist die repräsentierende Welt $\mathcal{W}_2$? Was sind ihre Objekte $\mathcal{O}_2$?
3. Welche Aspekte (Eigenschaften $\mathcal{E}_1$ und Relationen $\mathcal{R}_1$) in $\mathcal{W}_1$ sollen modelliert werden?
4. Welche Aspekte (Eigenschaften $\mathcal{E}_2$ und Relationen $\mathcal{R}_2$) in $\mathcal{W}_2$ modellieren?
5. Wie sieht die Korrespondenz $\rho : \mathcal{W}_1 \rightarrow \mathcal{W}_2$ bzw. $\rho : (\mathcal{O}_1, \mathcal{E}_1, \mathcal{R}_1) \rightarrow (\mathcal{O}_2, \mathcal{E}_2, \mathcal{R}_2)$ aus? Die Korrespondenz ρ wird *Repräsentation* genannt.

Abb. 2.3 verdeutlicht nun, wie verschiedene Relationen zwischen Objekten (Rechtecken) einer abstrakten Welt $\mathcal{W}_1 = \mathcal{A}$ in verschiedenen Welten $\mathcal{W}_2 \in \{\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{H}\}$ repräsentiert werden können. Dabei wird von allen anderen nicht in der Korrespondenz berücksichtigten Aspekten der beiden Welten abstrahiert: sie haben keine Bedeutung.

Eine Repräsentationskorrespondenz bzw. Abbildung zwischen Welten ist aus zwei Gründen sinnvoll:

1. ρ bietet die Möglichkeit, von allen belastenden und für die Lösung des Problemes irrelevanten Eigenschaften zu abstrahieren, um den Kern des Problemes herauszuschälen.
2. Im Modell $\mathcal{W}_2$ können Inferenzen (bzw. Operationen) durchgeführt werden, die in der Welt $\mathcal{W}_1$ entweder nicht möglich, zu gefährlich, zu unpraktisch, zu langsam oder zu teuer wären (s. auch [Pag91, Kap. 1]). Die so erhaltenen Einsichten können u.U. – bei geeigneter Interpretation – als Erkenntnisse über die Welt $\mathcal{W}_1$ angesehen werden.

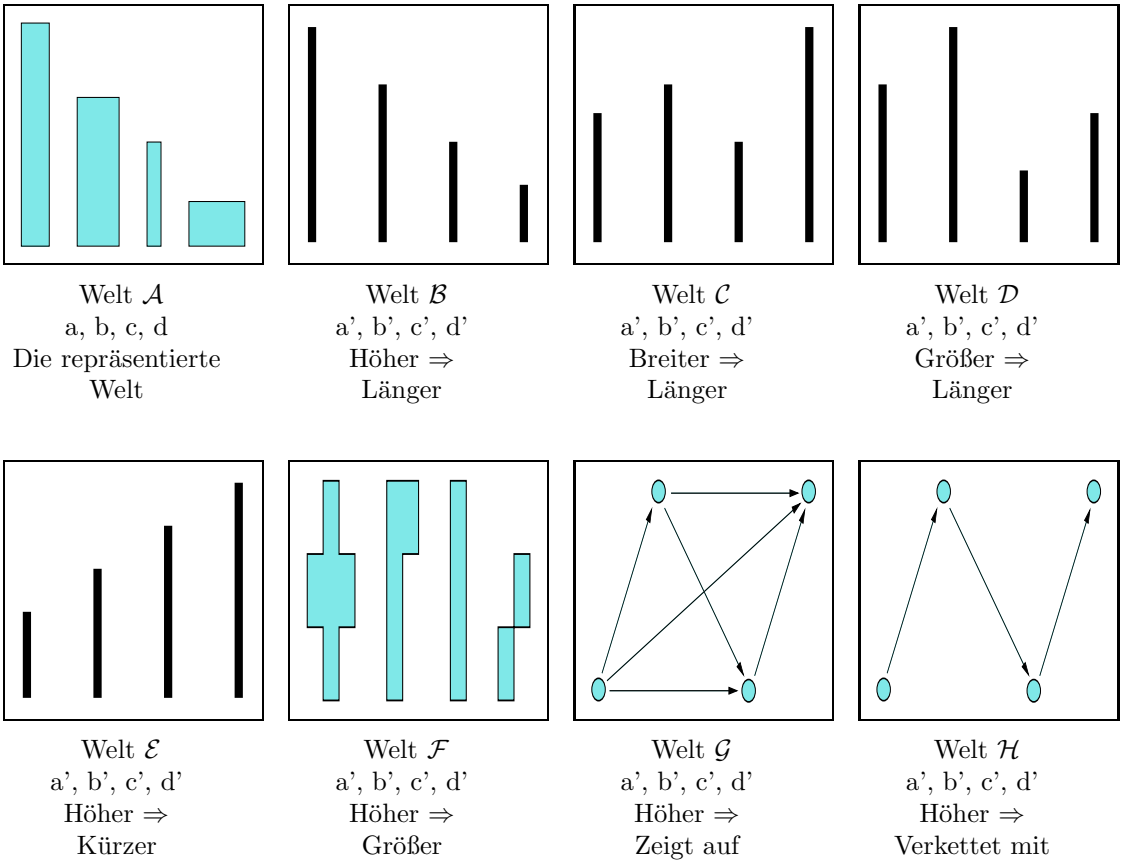


Abbildung 2.3: Palmers Welten (reproduziert nach [Pal78])

Man kann also Aktionen bzw. Operationen entweder in der Welt $\mathcal{W}_1$ direkt durchführen (wenn dies möglich ist, s.o.), oder aber entsprechende Aktionen in der Welt $\mathcal{W}_2$ durchführen und die so erhaltenen Ergebnisse auf $\mathcal{W}_1$ übertragen: damit dies möglich wird, müssen an ρ spezielle Bedingungen gestellt werden. Prinzipiell sollte es sich bei ρ um einen *Isomorphismus* handeln, also einen *bijektiven Homomorphismus*. Ein Homomorphismus ist in der Mathematik eine operationentreue Abbildung zwischen algebraischen Strukturen (z.B. Halbgruppen): für einen solchen Homomorphismus $\rho : (\mathcal{A}, \circ) \rightarrow (\mathcal{B}, \bullet)$ gilt: $\rho(x \circ y) = \rho(x) \bullet \rho(y)$, wobei $x, y \in \mathcal{A}$ und $x', y' \in \mathcal{B}$. Die Operationstreue und die Bijektivität von ρ sichert also, daß Inferenzen – bzw. Operationen wie $\bullet$ – die mit den Stellvertretern x', y' in $\mathcal{B}$ durchführt werden, auch in Bezug auf die Objekte x, y in $\mathcal{A}$ bzgl. $\circ$ Gültigkeit haben und sich somit auf die Welt A übertragen lassen. Schließlich kann ρ als Repräsentationsfunktion und ρ^{-1} (diese existiert ja für einen Isomorphismus) als Interpretationsfunktion verstanden werden. Interpretiert man $\circ$ und $\bullet$ als Prädikate, so müßte man zusätzlich $\rho(\top) = \top$ und $\rho(\bot) = \bot$ definieren ($\top$ steht für „wahr“, $\bot$ für „falsch“). Für die Welten $\mathcal{A}$ und $\mathcal{C}$ (Abb. 2.3) erhielte man dann für $\circ = \mathit{breiter}$ und $\bullet = \mathit{länger}$ $\rho(a \circ b) = \rho(\mathit{breiter}(a, b)) = \rho(\bot) = \bot$ auf der einen Seite, und $\rho(a) \bullet \rho(b) = \mathit{länger}(\rho(a), \rho(b)) = \mathit{länger}(a', b') = \bot$ auf der anderen Seite, also Gleichheit (die Formel $a \circ b \Rightarrow \rho(a) \bullet \rho(b)$ ist deutlicher). Anderseit könnte man auch in $\mathcal{C}$ beginnen und die dort gemachte Beobachtung bzw. „perzeptuelle Inferenz“ auf $\mathcal{A}$ übertragen: $\rho^{-1}(a' \circ b') = \rho^{-1}(\mathit{länger}(a', b')) = \rho^{-1}(\bot) = \bot$ auf der einen Seite, und $\rho^{-1}(a') \bullet \rho^{-1}(b') = \mathit{breiter}(\rho^{-1}(a'), \rho^{-1}(b')) = \mathit{breiter}(a, b) = \bot$ auf der anderen Seite, also Gleichheit (die Formel $a' \bullet b' \Rightarrow \rho^{-1}(a') \circ \rho^{-1}(b)$ ist deutlicher). Dabei wurde angenommen, daß Objekte auf Objekte abgebildet werden und daß Objekte für Objekte stehen – dies muß jedoch nicht immer so sein. U.a. sind in Welt $\mathcal{G}$ Pfeilobjekte eingetragen, die nicht für Objekte in $\mathcal{A}$ stehen, sondern für Relationen (hier müßte man dann auch anders als in obiger „Rechnung“ vorgehen). Derartige

Isomorphismen findet man seit langem in der mathematischen Systemtheorie, der Kybernetik, der Nachrichtentechnik und auch der Simulation ([Pag91]) – als Repräsentationen werden sie dort jedoch nicht bezeichnet.

Im Kontext dieser Arbeit ist zudem interessant, daß es sich bei den diskutierten Beispielwelten um *visuelle bzw. grafische Welten* handelt. Die Welten $\mathcal{B}$ bis $\mathcal{H}$ beschreiben jeweils andere Aspekte der Welt $\mathcal{A}$, und zwar wiederum auf grafische Weise. Werden die visuellen bzw. grafischen Elemente in einer konsistenten Weise verwendet, um komplexe Bedeutungen zu kommunizieren – was in den einzelnen Beispielwelten sicherlich der Fall ist –, so spricht man auch von einer *visuellen Sprache* oder einem *visuellen Formalismus*. Somit werden in den Palmer-Welten räumliche Konzepte durch räumliche Konzepte beschrieben. Die Rekonstruktion der relevanten Eigenschaften der Welt $\mathcal{A}$ ist jeweils nur bei Kenntnis der Interpretations- bzw. Repräsentationsfunktion möglich: da es sich um visuelle Repräsentationen handelt, besteht anderseits die Gefahr der Überinterpretation. Sicherlich lassen sich auch geometrische und metrische Eigenschaften in diesen Welten ablesen – diese tragen jedoch keine Bedeutung. Hier werden lediglich *qualitative* räumliche Konzepte wie „höher“ und „breiter“ betrachtet. Alle Welten $\mathcal{B}$ bis $\mathcal{H}$ sind unvollständig, denn sie erlauben es auch bei korrekter Interpretation nicht, die Welt $\mathcal{A}$ in all ihren Aspekten vollständig zu rekonstruieren. Lediglich die als relevant betrachteten Aspekte wurden ja repräsentiert. Eine Welt, die alle Aspekte von A repräsentiert, ist in Abb. 2.4 dargestellt – hier werden alle Eigenschaften der Welt $\mathcal{A}$ *direkt* dargestellt. Ähnlich einem Photo einer zweidimensionalen Welt handelt es sich hier um den trivialen Isomorphismus in Form der Identitätsfunktion.[1]

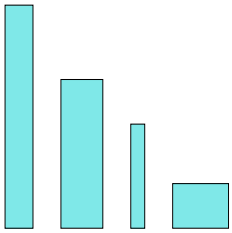


Abbildung 2.4: Welt $\mathcal{I}$

Unter anderem diskutiert Palmer nun einen Isomorphismus, den er *natürlichen Isomorphismus* nennt – eine solche Repräsentation hat folgende Eigenschaften (sinngemäß nach [Pal78, Reh90]):

1. In der repräsentierenden Welt $\mathcal{W}_2$ dürfen keine relationalen Elemente explizit erwähnt werden – dies bedeutet, daß die Relationen $\mathcal{R}_1$ ausschließlich auf Relationen in $\mathcal{R}_2$ abgebildet werden, und nicht etwa auf Objekte. Diese Eigenschaft ist z.B. für die Welten $\mathcal{G}$ und $\mathcal{H}$ verletzt.

2. Eine physikalische Identität der Repräsentationsmedien muß nicht vorliegen.

3. In den verschiedenen Repräsentationsmedien (der Welten $\mathcal{W}_1$ und $\mathcal{W}_2$) müssen dieselben *inhärenten Beschränkungen* existieren – dies bedeut zum einen, daß eine *funktionale Äquivalenz* zwischen repräsentierten und repräsentierenden Relationen vorliegen muß. So kann z.B. die transitive und asymmetrische „breiter als“-Relation (Palmer spricht auch von *Dimensionen*) nur durch eine ebenfalls transitive und asymmetrische Relation modelliert werden, wie z.B. „länger als“ – in diesem Fall spricht Palmer von *funktionalem Isomorpismus.* Zusätzlich soll nun aber gelten, daß die *Eigenschaften dieser Relationen (wie Transitivität etc.) automatisch durch die inhärenten Beschränkungen des Mediums erhalten bleiben sollen und daher nicht explizit modelliert werden müssen.* Diese Eigenschaft schafft eine gewisse *intrinisische Selbstkonsistenz:* bestimmte Bedingungen ergeben sich automatisch aufgrund der Struktur der Repräsentationsmedien und müssen somit nicht extra modelliert bzw. deren Konsistenz aufwendig verwaltet werden.

[1] Auf eine philosophische Diskussion der Unterschiede zwischen Identität und Gleichheit sei hier verzichtet.

Alsdann kann laut Palmer von einer *analogen Repräsentation* gesprochen werden. Er schreibt ([Pal78, S. 296]):

> Thus, whatever structure is present in an analog representation exists by virtue of the inherent constraints within the representing world itself, without reference to the represented world.

Diese Art der *intrinsischen Repräsentation* unterscheidet er von *extrinsischen oder propositionalen Repräsentationen* ([Pal78, S. 297]):

> Thus, whatever structure there is in a propositional representation exists solely by virtue of the extrinsic constraints placed on it by the truth-preserving informational correspondence with the represented world.

Während die repräsentierte Information also im Falle einer analogen Repräsentation automatisch aufgrund der inhärenten Beschränkungen der repräsentierenden Welt erhalten bleibt (Selbstkonsistenz), so muß im Falle einer propositionalen Repräsentation zusätzlicher Aufwand getrieben werden, damit die Repräsentation (in Bezug auf die relevanten Aspekte) *wahrheitserhaltend* ist – u.a. muß sichergestellt werden, daß die Eigenschaften der repräsentierten Relationen (wie Transitivität etc.) in der expliziten Modellierung erhalten bleiben.

Nach dieser Klassifikation können nun die Welten $\mathcal{B}$ bis $\mathcal{F}$ als *analoge Repräsentationen* bezeichnet werden, während die Welten bzw. Repräsentationen $\mathcal{G}$ und $\mathcal{H}$ als propositional einzustufen sind. So wird z.B. die Transitivität der durch die „Pfeil zeigt auf“-Relation repräsentierte „größer als“-Relation in Welt $\mathcal{G}$ nicht automatisch sichergestellt: u.a. kann man z.B. den Pfeil von a' nach d' einfach entfernen und erhält so eine inkonsistente bzw. nicht wahrheitserhaltende Repräsentation, da es sich nicht mehr um einen Isomorphismus handelt.

Will man Bilder (in gewissen Aspekten) durch Bilder beschreiben, so scheint es sinnvoll, gewisse Aspekte *direkt* bzw. *identisch* zu repräsentieren: hierbei handelt es sich um eine wesentlich stärkere Forderung als die des Palmerschen natürlichen Isomorphismuses. Gleiche Dimensionen sollen also durch gleiche Dimensionenen repräsentiert werden: wird also Farbe durch Farbe, Größe durch Größe, Enthaltensein durch Enthaltensein etc. repräsentiert, so spricht Palmer von einem *physikalischen Isomorphismus.*

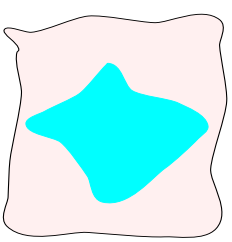
2.3.2 Analogische Repräsentationen

Die Palmersche Definition der analogen Repräsentation bzw. des natürlichen Isomorphismuses wird in der Literatur auch als *analogische Repräsentation* ([Fre91]) bezeichnet. Analogisch bringt dabei den Aspekt der Ähnlichkeit (Analogizität) zum Ausdruck – die relevanten Aspekte (Relationen und Eigenschaften) sollen also möglichst ähnlich repräsentiert werden. Diese Ähnlichkeit bezieht sich auf die strukturelle Übereinstimmung gewisser Aspekte der Welten $\mathcal{W}_1$ und $\mathcal{W}_2$. Will man zum Ausdruck bringen, daß best. Aspekte identisch durch dieselben Aspekte repräsentiert werden, so spricht man auch von *direkten Repräsentationen* – so kann z.B. die Eigenschaft, quadratische Form zu haben, direkt durch quadratische Form repräsentiert werden.

Bibel, Hölldobler und Schaub ([Bib93]) beschreiben analoge Repräsentationen folgendermaßen:

> Eine Darstellung eines Objektes (oder einer Szene) nennt man *analog*, wenn signifikante Eigenschaften, die auf das Objekt oder seine Teile zutreffen, in gleicher Weise auch auf das repräsentierte Objekt zutreffen. ... Insbesondere erkennt man, daß analog immer „analog in bezug auf eine gegebene Menge von Eigenschaften“ (nämlich die als signifikant eingestuften) bedeutet.

Als Beispiel diskutieren sie eine Bitmatrix, die eine weiße Flagge aus Leinen mit einer blauen Raute repräsentiert (s. Abb. 2.5). Solche Darstellungen wurden in der Literatur vielfach als *quasi-*



Die repräsentierte Welt

```
0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0
0 0 0 1 1 1 0 0 0
0 0 1 1 1 1 1 0 0
0 1 1 1 1 1 1 1 0
0 0 1 1 1 1 1 0 0
0 0 0 1 1 1 0 0 0
0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0
```

Die repräsentierende Welt

Abbildung 2.5: Eine analogische Repräsentation (reproduziert nach [Bib93])

analog bezeichnet. Die Darstellung repräsentiert u.a. (in gewissem Umfang) die Geometrie sowohl der Flagge als auch der blauen Raute. Eigenschaften wie Farbe und Material etc. werden nicht repräsentiert. Als weitere Beispiele analoger Repräsentationen führen sie Landkarten an. Die Bedeutung derartiger Darstellungen wird von ihnen in erster Linie darin gesehen, daß viele Verarbeitungs- und Deduktionsprozesse eingespart würden. So kann von einem Menschen die Frage, ob die Flaggenmitte in Abb. 2.5 in der Raute liegt oder nicht, direkt anhand einer perzeptuellen Inferenz ohne aufwendige Deduktionsprozesse (durch einfaches „Nachsehen“) beantwortet werden, wozu in einer propositionalen Repräsentation aufwendige Deduktionsprozesse und auch zusätzliches Wissen notwendig wären.

Oben wurde anhand der Welten $\mathcal{G}$ und $\mathcal{H}$ bereits deutlich, daß nicht automatisch jede bildhafte Darstellung (als solche wäre ja zunächst z.B. auch geschriebener Text zu werten) als analogische Repräsentation betrachtet werden kann – Bildhaftigkeit ist also keine hinreichende Bedingung für eine analoge Repräsentation. Bildhafigkeit ist jedoch auch keine notwendige Bedingung: so diskutiert Sloman ([Slo75]) das Beispiel einer sortierten Liste, in der die Relation „größer als“ zwischen den durch die Listenelementen repräsentierten Objekten durch die Ordnung auf den Listenelemente dargestellt wird.

Aufgrund der strukturellen Eigenschaften des Datentyps „Liste“ werden somit die strukturellen Eigenschaften der Relation „größer als“ intrinsisch durch die Relation „Nachfolgeelement von“ repräsentiert. Bei beiden handelt es sich um totale Strikt-Ordnungen. Hier scheint es wesentlich zu sein, auf welcher Ebene man argumentiert: letztlich müssen die strukturellen Beschränkungen des abstrakten (mathematischen) Datentyps „Liste“ in einer Computerimplementierung intern wieder durch flankierende Maßnahmen sichergestellt bzw. implementiert werden (Datenstruktur) – so wird die Relation bzw. Operation „Nachfolgeelement“ in der Regel durch eine Referenz (Pointer) auf das nächste Element der Liste implementiert. Auf dieser internen Ebene ist also die Relation „Nachfolgeelement“ in Form von belegten Speicherplätzen bzw. Objekten physisch repräsentiert. Auf dieser Ebene müßte man lt. Palmer somit von einer propositionalen Repräsentation sprechen - letzlich handelt es sich um eine „quasi-analogische“ Repräsentation. Es scheint sich also primär um eine Frage der Betrachtungsebene zu handeln, ob eine Repräsentation als analogisch oder propositional angesehen werden kann. Palmer bezeichnet Digitalrechner als „unstrukturierte Repräsentationsmedien“: dies gilt jedoch auch für die phsikalische Welt, wenn man sie auf einer (sub)atomaren Ebene betrachtet. Auf dieser Ebene gibt es für jede Eigenschaft und Relation eine „physikalische Implementation“ in Form von „Objekten“ (Elementarteilchen), so daß hier ähnliche Effekte auftreten, wie sie oben bei der Implementation der Liste diskutiert wurden. Geschichtete Softwarearchitekturen bzw. Virtuelle Maschinen sind ein fundamentales Konzept in der Informatik - selten wird man Repräsentationen auf der „unstrukturierten“ Ebene von Bits und Bytes vornehmen (hierin wird ja auch die Bedeutung von sehr hohen Programmiersprachen wie LISP in der KI gesehen). Während Palmer analogische und propositionale Repräsentationen als unvereinbar sieht („Moreover, it does put analog and propositional representations in opposition to one another.“, S. 296 [Pal78]), verschmelzen die Unterschiede auf einer genügend tiefen Betrachtungsebene – zahlreiche vergleichbare Debatten wurden in der Informatik bisher geführt (u.a. die „Deklarativ vs.

Prozedural“- oder „Symbolisch vs. Subsymbolisch (Konnektionistisch)“-Debatte).

Dennoch halte ich die Unterscheidung zwischen analogischen und propositionalen Repräsentationen für sinnvoll, da es wichtig ist, auf *semantisch hohen Ebenen* zu argumentieren – es ist wenig sinnvoll, alles auf der Ebene von Bits und Bytes oder Elementarteilchen vergleichen zu wollen, da hier schwerlich Unterschiede sichtbar werden. Auf dieser Ebene sind analogische und propositionale Repräsentationen nicht wesentlich verschieden voneinander: die Frage ist also, ob man möglichst viele Aspekte intrinsisch repräsentieren bzw. strukturelle Äquivalenzen der beiden Welten ausnutzen kann, indem man die Betrachtungsebene möglichst hoch bzw. ähnlich und somit problemadäquat wählt.

In der Literatur findet man insbesondere auch Versuche, die Vor- und Nachteile analogischer und propositionaler Repräsentation in *hybriden Systemen* zu kombinieren – so wurde z.B. im (Hamburger) „LILOG“-Projekt (hier ging es um das Verstehen natürlichsprachlicher Reisetexte) eine „mental imagery“-Komponente zur „mentalen Inspektion“ seitens des Systemes auf der Grundlage von Zellmatrizen verwendet, die als *Depiktionen* bezeichnet wurden ([HHP93, S. 189]; ein anderes hybrides System wird in [MK95] vorgestellt). Als Problem wurde insbesondere die Darstellung von Unterbestimmtheit in Bildern erkannt:

> Da für den Aufbau eines Bildes die Verwendung von Defaultinformationen über typische Form- und Lageeigenschaften unumgänglich ist *[unbekannte räumliche Aspekte – wie z.B. Form – müssen zum Aufbau des Bildes evtl. ergänzt werden, da sie in der propositionalen Repräsentation nicht festgelegt werden – daher sind Standardannahmen bzw. Defaultinformationen notwendig]*, sind auf einem Einzelbild auch keine sicheren Inferenzen möglich. Der beschriebene LILOG-Ansatz umgeht einen Teil dieses Problemes, indem zumindest für die Lageinformation Gebiete verwendet werden, die die Vereinigung einer Menge von möglichen Positionen repräsentieren. Ein weiteres Problem ist die Interaktion mit propositionalen Systemen, für die noch keine hinreichende Formalisierung existiert. *(Hinzufügungen in Kursivschrift, d.V.)*

Auch in VISCO werden derartige Gebiete verwendet - sowohl vage Position (Lageinformation) als auch vage Form können mit ihrer Hilfe beschrieben werden (s. Kap. 5). Im folgenden sind vor allem bildhafte Repräsentationen von Interesse: in dieser Arbeit sollen ebene Konstellationen durch ebene Konstellationen, also Bilder durch Bilder beschrieben werden. Offensichtlich kann eine analogische, direkt Repräsentation viele Vorteile bieten: Quadrate könnten durch Quadrate repräsentiert werden, Kreise durch Kreise, etc. Die Diskussion wird in Kap. 4 weitergeführt.

2.3.3 Mathematische Konzepte

Grundlegende mathematische räumliche Konzepte werden u.a. von der *(analytischen) Geometrie* und der *Topologie* untersucht – im folgenden sollen lediglich einige zentrale Begriffe identifiziert werden, die im weiteren Verlauf noch benötigt werden.[2] Im Sinne eines anwendungsorientierten Vorgehens geht es hier um die Nutzbarmachung mathematischer Theorien für die Zwecke bzw. Anwendungen der Informatik.

Das Wort *Geometrie* stammt ursprünglich aus dem Griechischen und bedeutet Erdmessung – geometrische Probleme traten also im Zusammenhang mit der Landvermessung auf, aber auch in der Astronomie und in der Architektur ([Dud94]). Als wichtigstes Werk der Antike ist das Buch „Die Elemente der Geometrie“ von Euklid (300 v. Chr.) zu erwähnen: hier wurde bereits versucht, eine *Axiomatisierung* der Geometrie vorzunehmen; das heute verwendete Axiomensystem geht auf David Hilbert zurück (es gibt mehrere äquivalente Axiomensysteme für die euklidische Geometrie). Grundbegriffe bzw. -terme der euklidischen Geometrie sind dabei *Punkt, Gerade* und *Ebene,* deren wechselseitigen Beziehungen und Eigenschaften nun durch die einzelnen Axiome festgelegt werden

[2] Eine sehr konzise und verständliche Darstellung der wesentlichsten mathematischen Konzepte findet man im Vorlesungsskript „Wissen über Raum, Zeit und Ereignisse“ von Habel, [Hab96].

(„Ein Punkt ist, was keine Teile hat.“). Lt. [Dud94] ist die euklidische Geometrie des Raumes eine Theorie einer Menge $\mathcal{R}$, *euklidischer Raum* genannt. Die Elemente von $\mathcal{R}$ werden *Punkte* genannt, spezielle Teilmengen von $\mathcal{R}$ bilden die *Geraden* und *Ebenen.* Die einzelnen Axiome werden in fünf Gruppen aufgeteilt: man unterscheidet zwischen den Axiomen der Verknüpfung, der Anordnung, der Kongruenz, der Stetigkeit und schließlich dem berühmten *Parallelenaxiom:* In jeder Ebene $\Phi \subset \mathcal{R}$ gibt es für jede Gerade $g \subset \Phi$ und jeden Punkt $P \notin g$ genau eine Gerade h mit $P \in h$ und $g \cap h = \emptyset$.

Aus diesen Axiomen können alle Lehrsätze der räumlichen Geometrie abgeleitet bzw. deduziert werden (im Sinne einer mathematischen Theorie) – weniger Axiome braucht man, um die euklidische Geometrie der Ebene zu definieren. Erst im 19. Jahrhundert wurde nachgewiesen, daß sich das Parallelenaxiom nicht aus den anderen Axiomen herleiten läßt – schließlich entdeckte man sog. *nichteuklidische Geometrien,* in denen das Parallelenaxiom gegen andere Axiome ersetzt wird (z.B. das *hyperbolische Parallelenaxiom* von F. Klein).[3]

Ein *euklidischer Raum* ist hier also eine abstrakte mathematische Struktur, die von Punkten, Strecken und Ebenen handelt. Die Trägermenge $\mathcal{R}$ muß nicht unbedingt $\mathbb{R}^n$ sein: u.a. kann es sich auch um abzählbare oder sogar endliche Mengen handeln. In dieser Arbeit sind nur einfache geometrische Konzepte wie *Länge, Orientierung, Winkel, Position* etc. von Interesse: während in der „Sprache“ der euklidischen Geometrie von diesen zunächst noch nicht gesprochen werden kann, werden nun in der ebenen *analytischen Geometrie* den Punkten in $\mathcal{R}$ Koordinaten über ein *Koordinatensystem* zugeordnet – so wird z.B. $\mathcal{R} = \mathbb{R}^2$. In dieser Arbeit werden lediglich *kartesische Koordinatensysteme* verwendet: bei diesem sind die Achsen rechtwinklig zueinander, die Achseneinheiten sind gleich groß.

Der Begriff des *metrischen Raumes* bietet nun eine Möglichkeit, von der Geometrie zur *Topologie* ([Jäh96, Dud94]) zu kommen, die eine Vielzahl weiterer interessanter Konzepte zu bieten hat (die folgenden Definitionen stammen größtenteils aus [Jäh96] und [Dud94]):

Definition (Metrischer Raum): Ein *metrischer Raum* ist ein Tupel $(\mathcal{M}, d)$, wenn für jedes Paar $(x, y) \in \mathcal{M}^2$ eine nichtnegative reelle Zahl $d(x, y)$ mit folgenden Eigenschaften definiert ist (die sog. Metrik):

1. $\forall x, y\ (d(x, y) = 0 \Leftrightarrow x = y)$ (Identitätsaxiom)
2. $\forall x, y\ (d(x, y) = d(y, x))$ (Symmetrieaxiom)
3. $\forall x, y, z\ (d(x, y) \leq d(x, z) + d(z, y))$ (Dreiecksungleichung)

Die bekannteste Metrik ist die euklidische Metrik: $d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Es gibt verschiedene Metriken auf $\mathbb{R}^2$ – im Sinne obiger Definition ist $\mathbb{R}^2$ dann sowohl ein analytischer Raum als auch ein metrischer Raum. In Verbindung mit einer Metrik kann er nun auch als topologischer Raum betrachtet werden:

Definition (Topologie eines metrischen Raumes): Sei $(\mathcal{M}, d)$ ein metrischer Raum. Eine Teilmenge $\mathcal{V} \subset \mathcal{M}$ heiße offen, wenn es zu jedem $x \in \mathcal{V}$ ein $\epsilon > 0$ gibt, so daß die „ϵ-Kugel“ $K_\epsilon(x) := \{y \in \mathcal{M} \mid d(x, y) \leq \epsilon\}$ um x noch ganz in $\mathcal{V}$ liegt (also $K_\epsilon(x) \subseteq \mathcal{V}$). Die Menge $\mathcal{O}(d)$ aller offenen Teilmengen von $\mathcal{M}$ heißt die *Topologie* des metrischen Raumes $(\mathcal{M}, d)$. Die offenen Teilmengen heißen auch *Umgebungen.*

In diesem Zusammenhang spricht man auch von der durch die Metrik *induzierten Topologie:* sehr verschiedene Metriken können die gleiche Topologie hervorbringen. Die bereits verwendete Sprechweise, daß die Geometrie „Träger“ der Topologie ist, wird somit gerechtfertigt. Es gibt diverse äquivalente Definitionen des topologischen Raumes. Während in obiger Definition die sog. offenen Mengen (oder Umgebungen) durch die Metrik konstruiert werden, gibt es auch Definitionen, in denen die offenen Mengen in gewisser Weise „explizit“ vorliegen:

[3] Nichteuklidische Geometrien spielen z.B. in Kosmologischen Weltmodellen (gekrümmten Räumen) oder der Allgemeinen Relativitätstheorie eine Rolle.

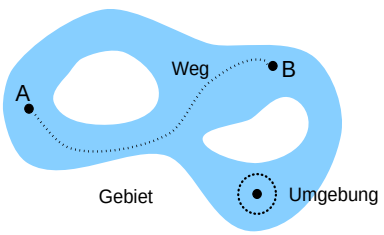


Illustration Gebiet

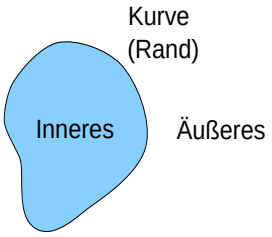


Illustration Jordanscher Kurvensatz

Abbildung 2.6: Gebiete

Definition (Topologischer Raum): Ein *topologischer Raum* ist ein Paar $(\mathcal{M}, \mathcal{O})$, bestehend aus einer Menge $\mathcal{M}$ und einer Menge $\mathcal{O}$ von Teilmengen (genannt offene Mengen) von $\mathcal{M}$, derart daß gilt:
Axiom 1: Beliebige Vereinigungen von offenen Mengen sind offen.
Axiom 2: Der Durchschnitt von je zwei offenen Mengen ist offen.
Axiom 3: $\emptyset$ und $\mathcal{M}$ sind offen.

Da Geometrie und Topologie also zunächst eigenständige mathematische Theorien sind, zwischen denen jedoch über den Begriff der Metrik eine Verbindung geschaffen werden kann, findet man in der Literatur oft die Aussage, daß topologische Konzepte *abstrakter* seien als geometrische Konzepte – eine Geometrie kann topologisch „interpretiert“ werden. Wird z.B. ein Diagramm bzw. dessen konkrete Geometrie topologisch interpretiert, so werden alle dargestellten geometrischen bzw. metrischen Aspekte wie Orientierung, Position, Form etc. als irrelevant betrachtet und ignoriert. Lediglich topologische Konzepte wie Verbundenheit, Inneres und Äußeres zählen dann (s.u.). Eine solche Struktur ist z.B. ein *Graph:* in einer Visualisierung eines Graphen haben die konkreten geometrischen Formen, Abmessungen und Position sowie Kantenverläufe keine Bedeutung. Lediglich die Verbundenheit diskreter Objekte durch Kanten zählt. Die Visualisierung eines Graphen muß somit topologisch interpretiert werden. Statt von *Graphentheorie* spricht man auch von *kombinatorischer Topologie.*

Definition (ϵ-Umgebung): Die ϵ-Umgebung eines Punktes $p = (x_0, y_0) \in \mathbb{R}^2$ ist $U_\epsilon(p) := \{(x, y) \in \mathbb{R}^2 \mid (x - x_0)^2 + (y - y_0)^2 < \epsilon^2\}$. Eine Menge $U \subset \mathbb{R}^2$ heißt *Umgebung* von p, wenn $\exists\, \epsilon\ (U_\epsilon(p) \subseteq U(p))$. $U_\epsilon(p)$ ist also eine offene Menge (s.o.).

Definition (Gebiet): Eine Teilmenge G von $\mathbb{R}^2$ heißt ein *Gebiet,* wenn sie offen und zusammenhängend ist (s. Abb. 2.6). Eine Menge $\mathcal{G}$ ist *(weg)zusammenhängend,* wenn je zwei Punkte aus $\mathcal{G}$ durch eine ganz in $\mathcal{G}$ verlaufende Kurve $\mathcal{K}$ verbunden werden können (so daß $\mathcal{K} \subseteq \mathcal{G}$, s. Punkte A und B in Abb. 2.6(a)). Die (Weg)Zusammenhangsrelation ist eine *Äquivalenzrelation* (reflexiv, symmetrisch, transitiv): die entsp. *Äquivalenzklassen* heißen *Wegzusammenhangskomponenten.* In der Standardtopologie fallen Zusammenhang und Wegzusammenhang zusammen, sodaß man auch von *Zusammenhangskomponenten* redet.

Definition (Innerer Punkt, Inneres, offener Kern): Ein Punkt $p \in \mathcal{M} \subseteq \mathbb{R}^2$, für den es eine Umgebung $U(p)$ gibt, so daß $U(p) \subseteq \mathcal{M}$ gilt, heißt *innerer Punkt.* Die Menge der inneren Punkte von $\mathcal{M} \subseteq \mathbb{R}^2$ wird *offener Kern* oder *Inneres* genannt und mit $\mathcal{M}^\circ$ bezeichent.

Definition (Äußerer Punkt, Äußeres): Ein Punkt $p \in \mathcal{M} \subseteq \mathbb{R}^2$, für den es eine Umgebung $U(p)$ gibt, so daß $U(p) \cap \mathcal{M} = \emptyset$ gilt, heißt *äußerer Punkt.* Die Menge der äußeren Punkte von $\mathcal{M} \subseteq \mathbb{R}^2$ wird *Äußeres* genannt. Sie ist gleich der Menge $\mathbb{R}^2 \setminus \mathcal{M}$ und wird mit $\mathcal{M}^{-1}$ bezeichnet.

Definition (Randpunkt, Rand): Ein Punkt $p \in \mathcal{M} \subseteq \mathbb{R}^2$ heißt *Randpunkt,* wenn für jede Umgebung $U(p)$ von p gilt: $U(p) \cap \mathcal{M} \neq \emptyset$ und $U(p) \cap (\mathbb{R}^2 \setminus \mathcal{M}) \neq \emptyset$. Die Menge der Randpunkte von $\mathcal{M}$ wird mit $\partial\mathcal{M}$ bezeichnet und *Rand* genannt.

Schließlich sei noch der Begriff des *Homöomorphismus* erwähnt:

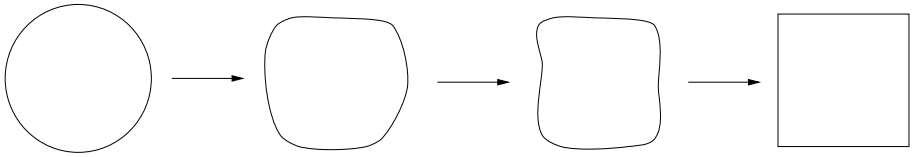


Abbildung 2.7: Eine Reihe von Homöomorphismen, die einen Kreis in ein Quadrat überführen

Definition (Homöomorphismus): Ein *Homöomorphismus* ist eine bijektive Abbildung zwischen topologischen Räumen $\mathcal{T}$ und $\mathcal{T}'$, so daß die Abbildung die $f : \mathcal{T} \to \mathcal{T}'$ die topologische Struktur von $\mathcal{T}$ auf die topologische Struktur von $\mathcal{T}'$ überträgt, und analog für f^{-1}. In diesem Fall nennt man f und f^{-1} *stetig.* Im Kontext dieser Arbeit ist nur relevant, daß sich topologische Abbildungen auf $\mathbb{R}^2$ als *Verformungen* interpretieren lassen – man bezeichnet die Topologie auch als „Gummigeometrie“ (Rubbersheet Geometry). Eigenschaften (wie z.B. Zusammenhang), die sich durch die Abbildung f nicht ändern, werden *topologische Invarianten* genannt. Man spricht auch von *topologischer Äquivalenz:* so läßt sich z.B. ein Kreis über einen geeignete Homöomorphimus zu einem Quadrat deformieren (s. Abb. 2.7). Kreis und Quadrat sind daher topologisch äquivalent. Nicht äquivalent sind jedoch Kreisring und Kreis, Punkt und Kurve, etc.

An dieser Stelle wird deutlich, *daß topologische Begriffe allein nicht ausreichend sind um ein breite Flächendeckung räumlicher Konzepte zu ermöglichen:* zumindest in meinem Denken sind die Begriffe Kreis und Quadrat unvereinbar. Das Begriffspaar „rund – eckig“ bildet in meinem Denken einen ebenso starken Konstrast wie z.B. „heiß – kalt“ oder „schwarz – weiß“.

Der **Jordansche Kurvensatz** (s. Abb. 2.6) besagt, daß eine *einfache geschlossene Kurve* die Ebene $\mathbb{R}^2$ in zwei Gebiete (lt. obiger Definition) aufteilt, genannt das *Innere* und das *Äußere* der Kurve. Eine *geschlossene Kurve* heißt *einfach,* wenn sie sich nicht selbst berührt oder überschneidet. In diesem Sinne wird z.B. auch von *einfachen Polygonen* gesprochen. Eine *geschlossene Kurve* hat keine Endpunkte. Statt von Kurven wird oft auch von *Linien* gesprochen – *Strecken* sind (geradlinige) spezielle Linien bzw. Kurven. Eine *ebene Kurve* kann kann z.B. über eine analytischen Ausdruck der Art $f : \mathbb{R} \to \mathbb{R}^2$ beschrieben werden, z.B. $f(t) = (x, y) = (a \cdot \cos t, b \cdot \sin t)$ *(Parameterform).*

In der *digitalen Topologie* wird nun versucht, topologische Konzepte wie Zusammenhang etc. in adäquater Weise auf die diskreten Strukturen digitaler Rechner zu übertragen, um sie z.B. für die maschinelle Bilderverarbeitung (Computer Vision) nutzbar zu machen: die digitale Topologie handelt statt von der überabzählbaren Menge $\mathbb{R}^2$ von der abzählbaren Menge $\mathbb{Z}^2$ (oder $\mathbb{N}^2$), wie sie z.B. ausschnittsweise direkt in Feldern bzw. Speicherrastern (Arrays) im Rechner implementiert werden könnten. Hier gibt es jedoch Problem mit dem Begriff des *Zusammenhangs.*

2.3.4 Qualitatives räumliches Schließen

Schlägt man im Lexikon ([Fis79]) unter *Qualität* nach, so findet man Definitionen wie „Die Beschaffenheit eines Dinges, die sich durch seine Eigenschaften zeigt; philosophisch gilt die Qualität als Kategorie“. Im Gegensatz hierzu steht die *Quantität,* die durch „Menge, Anzahl, Masse; philosophisch: die Größe (in Raum und Zeit), als Grundeigenschaft jedes Dinges.“ bezeichnet wird. Eine Kategorie wird allgemein einfach als „Klasse“, im philosophischen Sinne jedoch als „jeder für einen bestimmten Seinsbereich grundlegende Begriff“ umschrieben: „... Aristoteles nannte ... zehn Kategorien als Grundbestimmungen jedes Dinges: Substanz, Größe (Quantität), Beschaffenheit (Qualität), Verhältnis zu anderen Dingen (Relation), Ort, Zeit, Tätigkeit, Leiden, Sichverhalten und Lage.“

Betrachtet man räumliche Objekte, so wird deutlich, daß die Qualitäten bzw. Beschaffenheiten wie „Form“, „Größe“, „Orientierung“, „Position“ etc. *quantitativ* durch Zahlen repräsentiert werden können, indem man z.B. *Messungen* dieser Eigenschaften vornimmt (sofern es sich um physische

Dinge handelt). Die Form eines Rechteckes kann quantitativ durch die Angabe der Positionen der vier Eckpunkte repräsentiert werden. Offensichtlich sind geometrische Konzepte zur Repräsentation dieser Qualitäten geeignet – die Repräsentation geschieht auf quantitative (numerische) Art.

Hierzu im Gegensatz stehen *qualitative Repräsentationen* dieser Qualitäten: dabei werden die als relevant betrachteten Eigenschaften und Relationen räumlicher Objekte nicht quantitativ durch Zahlen, sondern qualitativ wiederum durch Klassen bzw. Kategorien repräsentiert. Solche Kategorien (Kategorien sind Begriffe, hier: Klassenbegriffe) sind z.B. in der natürlichen Sprache vorhanden: „groß“, „klein“, „rund“, „eckig“, „breit“, „hoch“, etc. Die Kategorisierung bzw. Klassifizierung eines Objektes als „hoch“ hat natürlich einen anderen Status als „1,85656 Meter hoch“. Qualitative Repräsentationen sind somit abstrakter als quantitative Repräsentationen und repräsentieren daher in der Regel größere Klassen. Die Aufteilung des Merkmalsraumes in die entsp. Kategorien und die *Granularität* der einzelnen Kategorien wird in der Regel vom Kontext bestimmt: ein Elefant ist „groß“, neben einer Ameise aber „sehr groß“. Als Vorteil hervorzuheben ist nun, daß die Granularität der Kategorien so gewählt werden kann, daß sie gerade ausreichend für die entsp. Problemstellung bzw. problemadäquat ist – somit müssen nur so viele Unterscheidungen wie für den aktuellen Problemkontext notwendig gemacht werden. Vage Information läßt sich gut durch entsp. vage bzw. grobe Kategorien darstellen. So kann es für eine qualitative Simulation des Schmelzvorganges eines Eiswürfels ausreichen, statt des Temperaturkontinuums die drei Intervalle „unter 0 Grad“, „unter 100 Grad“, „über 100 Grad“ zu betrachten. Derartige qualitative Simulationen werden in der *qualitativen Physik* untersucht, und für die entsp. Kategorien wird der Begriff des „Quantity Space“ verwendet.

Über verschiedene Ebenen können qualitative Repräsentationen in der Granularität nun schließlich beliebig nahe an die Auflösung quantitativer Repräsentationen herankommen: der Unterschied zwischen quantitativen und qualitativen Repräsentationen ist somit im wesentlichen ein „Bottom-Up“ vs. „Top-Down“-Unterschied. Im Sinne eines „Teile und Herrsche“-Vorgehens können Kategorien durch rekursive Dekomposition auf eine problemadäquate Granularität heruntergebrochen werden („Teile“) und dann entsp. Inferenzen auf diesen Ebenen durchgeführt werden („Herrsche“). Die Möglichkeit, nur *so viele Unterscheidungen wie nötig* vorzunehmen, wird als großer Vorteil gesehen. Das Inferieren auf verschiedenen Detaillierungsgraden einer Repräsentation wird als *hierarchisches Schließen (Hierachical Reasoning)* bezeichnet – die Inferenzprozesse haben in der Regel eine Komplexität, die vom Detaillierungsgrad bzw. der Granularität der Ebene abhängt; teilweise lassen sich große Effizienzgewinne in den entsp. Inferenzalgorithmen erzielen (manchmal kann eine *kombinatorische Explosion* vermieden werden, s. [Fre92b]).

Mit wachsender Abstraktheit der Repräsentation wird die (evtl.) strukturelle Übereinstimmung gewisser Aspekte zwischen repräsentierter und der repräsentierenden Welt jedoch immer schwächer, und so nähert man sich schließlich wieder überwiegend propositionalen Beschreibungen und verliert u.U. die Vorteile, die sich – wie bei den analogischen Repräsentationen – aufgrund der Ausnutzung struktureller Äquivalenzen gewisser Aspekte beider Welten ergeben. Dies mag sich auch wieder negativ auf die Effizienz der entsp. Inferenzalgorithmen auswirken. Schlimmstenfalls können in bezug auf die repräsentierte Welt inkorrekte Deduktionen gemacht werden. Die „Selbstkonsistenz“ wurde ja z.B. bei analogischen Repräsentationen als großer Vorteil gesehen.

Solche Probleme, die sich durch Unterspezifiziertheit und Abstraktheit ergeben, werden auch in der qualitativen Physik beobachtet ([GM95a]):

> There seems little doubt that the notion of quantity space is a fundamental one, and in researches such as those cited above, it has been used to good effect to reduce the descriptions of physical variables to cognitively more sensible proportions. The typical use is to replace quantitative differential equations by their qualitative analogues, and use a kind of qualitative mathematics to deal with them.
>
> It must be admitted however that the results of „qualitative physics“ so far have not been too impressive. For instance, the formalization of the behavior of rather simple

> physical systems by de Kleer and Brown and Kuipers, lead to an alarmingly large multiplicity of solutions, and the fragmentary attempts to axiomatize the behavior of liquids of Hayes and Forbus ... suggest that very large axiom systems will be needed, coming up against all the usual control problems of systems of deductive inference as well as the frame problem.

In der Literatur lassen sich verschiedenste Gründe für die Beschäftigung mit qualitativen Repräsentation (und Inferenzmethoden auf diesen) finden:

- **Kognitive Adäquatheit:** Vielfach wird – insbesondere aus Reihen der kognitionswissenschaftlich geprägten KI – darauf hingewiesen, daß Menschen keine Kompetenz im Umgang mit numerischen Quantitäten besitzen: der Mensch verfügt weder über „direkt Zahlen liefernde" Sinnesorgane noch ein Gehirn, welche auf die Verarbeitung numerischer Werte ausgelegt ist. Größen von Gegenständen können z.B. visuell gut miteinader verglichen und qualitativ kategorisiert werden (Kategorien wie „groß" und „klein" spielen hier eine Rolle), doch quantitativ kann die Größe meist nur sehr ungenau angegeben werden.[4] Will man also KI im Sinne einer Simulation des menschlichen Geistes betreiben, so sollte für die entsp. Systeme (und somit die verwendeten Repräsentationsformate und Inferenzmethoden) *kognitive Adäquatheit* angestrebt werden – hier stellt sich die Frage, ob solche Systeme dann nicht auch menschliche Schwächen und Defizite haben sollten und somit *fehlerhaft* wären.
- **Simulation physikalischer Systeme:** der Mensch schafft es, z.B. das Verhalten von Flüssigkeiten vorherzusehen, ohne eine große Anzahl von Differentialgleichungen zu lösen. Letzteres Vorgehen ist – trotz enormer Rechenleistungen heutiger Rechner – nicht praktikabel. Komplexe physikalische Systeme lassen sich wesentlich effizienter durch qualitative Physik (in ihren relevanten Aspekten) simulieren (s. [Gör93, Kap. 1.3]). Das Lösen physikalischer Differentialgleichungen kann nicht als kognitiv adäquates Schlußfolgern z.B. des Menschen über seine physikalische Umwelt angesehen werden – das notwendige Hintergrundwissen über die Welt ist für den richtigen Umgang bzw. das Schlußfolgern mit physikalischen Formeln essentiell und nicht in den entsp. Gleichungen formuliert. Während qualitative Physik im Sinne einer naiven Physik des Alltäglichen vielleicht für die Lösung vieler Alltagsprobleme ausreichen mag, so sollte doch eine Wetterprognose möglichst quantitativ ausfallen.
- **Unzureichende Abstraktheit quantitativer Repräsentationen:** oftmals werden größere Klassen benötigt, als sie durch quantitative Konzepte definiert werden können. Eine geometrische Repräsentation ist z.B. nicht in der Lage, die Form von räumlichen Objekten unspezifiziert zu lassen – sollen jedoch Schlüsse mit vager oder gänzlich unbekannter Information durchgeführt werden (z.B. unbekannter Form), so müssen Repräsentationsformate und Inferenzmechanismen entwickelt werden, die derartige Abstraktionen ermöglichen.

Ebenso wie in der „extrinsisch – intrinsisch"- bzw. „propositional – analogisch"- oder hier nun „quantitativ – qualitativ"-Debatte scheint es auf eine ausgewogene Mischung entsp. Anteile aus beiden Welten anzukommen. Speziell an *hybriden Modellen* wird intensiv gearbeitet, da sie eine möglichst effiziente und adäquate Ausnutzung der besten Aspekte aus beiden Welten ermöglichen sollen.

Im Bereich des qualitativen räumlichen Schließens findet man vor allem viele *einschränkungs- bzw. constraintbasierte Ansätze:* die repräsentierende Welt nimmt hier die Form eines Graphen an, dessen Knoten die repräsentierten räumlichen Objekte und dessen Kanten qualitative räumliche Relationen zwischen diesen Objekten repräsentieren (evtl. auch ganze Disjunktionen von Relationen). Einzelne Ansätze unterscheiden sich in erster Linie im Inventar an unterschiedlichen Kanten und in den repräsentierbaren räumlichen Objekten: diese reichen von einfachen Punkten über Rechtecke, teilweise konvexe, teilweise einfache topologische Gebiete (s.o.) bishin zu Gebieten

[4] Hierzu sei bemerkt, daß es dennoch Menschen gibt, die mit außergewöhnlichen *quantitativen* Fähigkeiten glänzen – u.a. Rechenkünstler oder sog. „Idiot Savants".

mit unscharfen Grenzen oder gar Aggregaten. Die meisten Ansätze verwenden qualitative räumliche topologische Relationen – solchen Relationen sind z.B. „überlappt“, „enthält“, „dijunkt“; sie können anhand topologischer Beschreibungen definiert werden (s.u.). Bei Befarf können derartige Relationen dann mit qualitativen Orientierungen und Distanzen verfeinert werden.

Durch einen *Einschränkungsfortpflanzungsprozeß (Constraintpropagierung)* in einem *Einschränkungsgraphen (Constraintnetzwerk)* können nun sowohl inkonsistente Beschreibungen als auch unvollständige Informationen ergänzt werden – so kann z.B. eine fehlende (also unbekannte räumliche Relation zwischen zwei Objekten) durch den Propagierungsprozeß ergänzt werden. Hier wird deutlich, daß es sich um eine überwiegend propositionale Repräsentation handelt: erst durch einen aufwendigen (in der Regel sogar NP-vollständigen, s. [GPP95]) Inferenzprozeß können inkonsistente Beschreibungen entdeckt werden. Offensichtlich ist dies der Preis, den man für die Abstraktion zu zahlen hat. Die einzelnen Inferenzregeln werden in Form einer sog. *Kompositions-* oder *Transitivitätstabelle* angegeben: dabei wird anhand der (bereits bekannten) Relationen zwischen a und b und b und c (also $R_1(a,b)$ und $R_2(b,c)$) auf die möglichen Relationen zwischen a und c geschlossen. Die Transitivitätstabelle listet nun eine Reihe von möglichen $R_3(a,c)$-Einträgen auf. Durch weitere Propagierungen werden schließlich die möglichen Relationen zwischen a und c weiter eingeschränkt – der Propagierungsprozeß stoppt, wenn keine Veränderung mehr erzielt wird oder eine Inkonsistenz entdeckt wurde. Aufgrund der Transitivitätstabelle können sowohl inkonsistente Beschreibungen erkannt als auch unvollständige Beschreibungen ergänzt werden. Konsistente Netzwerke werden als *relational konsistent* bezeichnet.

Von Grigni et al. wurde in [GPP95] gezeigt, daß die Berechnung der *deduktiven Hülle* durch einen Einschränkungspropagierungssprozeß für die Egenhofer-Relationen (s.u.) nicht ausreichend ist, um inkonsistente Formulierungen zu erkennen: relationale Konsistenz ist zwar korrekt, aber unvollständig bzgl. einer geometrischen Interpretation, denn evtl. gibt es dennoch keine *Realisierung* einer als nicht inkonsistent erkannten Beschreibung – dies hängt mit einer zusätzlichen *Planaritätsforderung* zusammen, die bisher übersehen wurde. Unter einer Realisierung wird hier eine Konstellation von topologischen Gebieten in der Ebene verstanden, die genau in den angegebenen Relationen stehen. Dies steht im Gegensatz zu in diesem Zusammenhang gemachten Aussagen wie in [HHP93, S. 179]:

> Stattdessen findet eine intrinsische Modellierung statt, bei der die Eigenschaften des Raumes vollständig oder zumindest teilweise im Repräsentationssystem selbst verankert sind. Das System „simuliert“ die betrachtete Domäne statt sie explizit zu formalisieren; inkonsistente Formalisierungen sind dann nicht mehr möglich.

Es ist klar, daß mit zunehmender Abstraktheit einer Repräsentation auch wachsende Realitätsferne (von der repräsentierten Domäne) erkauft wird – insofern besteht zwischen „vollständig“ und „zumindest teilweise“ ein großer Unterschied.

Als Beispiel eines sehr populären qualitativen räumlichen Kalküls soll der bereits erwähnte Egenhofer-Kalkül ([Ege91]) vorgestellt werden (Abb. 2.8): in ihm können qualitative topologische Relationen zwischen einfachen topologischen Gebieten behandelt werden. Da es sich um topologische Relationen handelt, sind die Beschreibungen invariant gegenüber homöomorphen Abbildungen wie Deformationen, Rotationen, Skalierungen und Translationen (der gesamten Szene, nicht einzelner Objekte). Oftmals sind die Egenhofer-Relationen jedoch zu generell bzw. zu abstrakt; dann bietet es sich an, die Relationen mit metrischen Eigenschaften zu verfeinern ([Ege96b]). Die entsp. Relationen werden nun durch die sog. 9er-Schnittmatrix (9-Intersection) charakterisiert, die folgende Form hat:

$$\begin{pmatrix} \partial A \cap \partial B & \partial A \cap B^\circ & \partial A \cap B^{-1} \\ A^\circ \cap \partial B & A^\circ \cap B^\circ & A^\circ \cap B^{-1} \\ A^{-1} \cap \partial B & A^{-1} \cap B^\circ & A^{-1} \cap B^{-1} \end{pmatrix}$$

Die in der Matrix auftauchenden Schnittmengen können entweder leer oder nichtleer sein; es wurden acht interessante topologische Relationen identifiziert, die in Abb. 2.8 veranschaulicht werden.

$$disjoint(A,B) = \begin{pmatrix} \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix} \quad meet(A,B) = \begin{pmatrix} \neg\emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$$

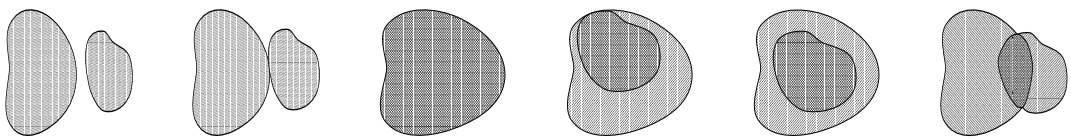
$$equal(A,B) = \begin{pmatrix} \neg\emptyset & \emptyset & \emptyset \\ \emptyset & \neg\emptyset & \emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix} \quad inside(A,B) = \begin{pmatrix} \neg\emptyset & \neg\emptyset & \emptyset \\ \emptyset & \neg\emptyset & \emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$$

$$covered\_by(A,B) = \begin{pmatrix} \neg\emptyset & \neg\emptyset & \emptyset \\ \emptyset & \neg\emptyset & \emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix} \quad contains(A,B) = \begin{pmatrix} \emptyset & \emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix}$$

$$covers(A,B) = \begin{pmatrix} \neg\emptyset & \emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix} \quad overlaps(A,B) = \begin{pmatrix} \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$$

Die Kompositionstabelle enthält nun 64 Eintragungen und hat folgende Form (d = *disjoint*, m = *meet*, e = *equal*, i = *inside*, cb = *covered_by*, ct = *contains*, cv = *covers*, o = *overlaps*):

	d	m	e	i	cb	ct	cv	o
d	d m e i cb ct cv o	d m i cb o	d	d m i cb o	d m i cb o	d	d	d m i cb o
m	d m ct cv o	d m e cb cv o	m	i cb o	m i cb o	d	d m	d m i cb o
e	d	m	e	i	cb	ct	cv	o
i	d	d	i	i	i	d m e i cb ct cv o	d m i cb o	d m i cb o
cb	d	d m	cb	i	i cb	d m ct cv o	d m e cb cv o	d m i cb o
ct	d m ct cv o	ct cv o	ct	e i cb ct cv o	ct cv o	ct	ct	ct cv o
cv	d m ct cv o	m ct cv o	cv	i cb o	e cb cv o	ct	ct cb	ct cv o
o	d m ct cv o	d m ct cv o	o	i cb o	i cb o	d m ct cv o	d m ct cv o	d m e i cb ct cv o



disjoint disjoint | meet meet | equal equal | covers covered-by | contains inside | overlaps overlaps

Abbildung 2.8: Topologische Relationen nach Egenhofer (und ihre Inversen)

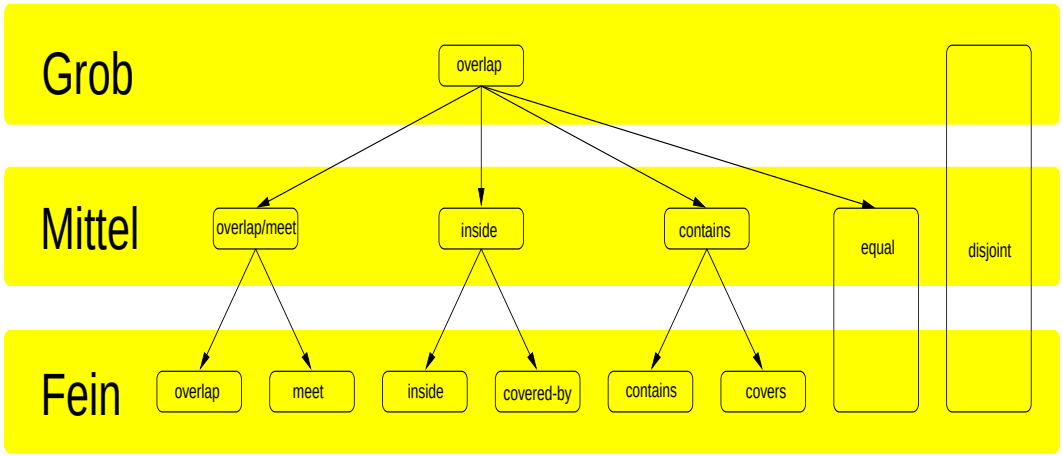
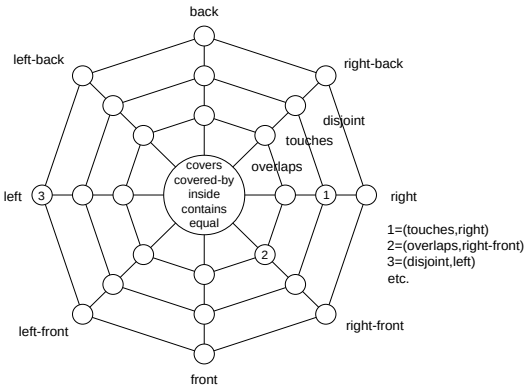
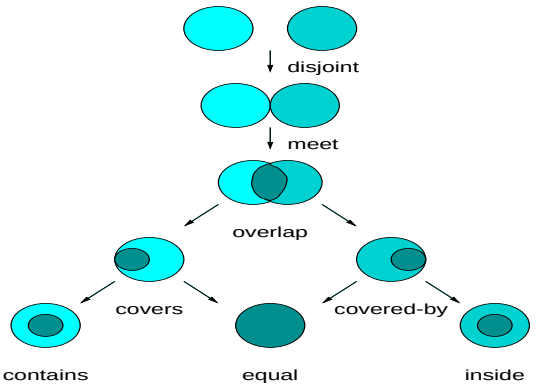


Abbildung 2.9: Hierarchische Egenhofer-Relationen (reproduziert nach [GPP95])

Liegt nun z.B. die Information $meet(a, b)$ und $covers(b, c)$ vor, so kann anhand der Tabelle geschlossen werden, daß $disjoint(a, c) \vee meet(a, c)$ gelten muß. Anhand dieser abgeleiteten Information können dann weitere Inferenzen gezogen werden. Von Grigni et al. wurde in [GPP95] gezeigt, daß die Entscheidung auf relationale Konsistenz im Egenhofer-Kalkül NP-hart ist. Schränkt man jedoch das Constraintsystem so ein, daß zwischen je zwei Knoten im Constraintgraphen entweder genau eine Relation bekannt ist oder vollständige Unkenntnis bzgl. der Relationen herrscht, so wird das Problem polynomial (P). Da aber relationale Konsistenz nicht ausreichend ist und auch noch ein Planaritätstest durchgeführt werden muß, ist das vollständige *Realisierbarkeitsproblem („Gibt es eine Konstellation von Gebieten in der Ebene, die genau in den angegebenen Relationen stehen?“)* dennoch NP-hart. Egenhofers Relationen lassen sich *hierarchisch zusammenfassen:* Grigni et al. untersuchten die Komplexität des Realisierbarkeitsproblemes auf den drei Granularitätsebenen „Fein“, „Mittel“ und „Grob“ (s. Abb. 2.9). Das allgemeine Realisierbarkeitsproblem bleibt jedoch NP-hart: für die Entscheidung auf relationale Konsistenz kann jedoch im allgemeinen Fall gezeigt werden, daß das Problem auf grober Granularitätsebene in P, für die mittlere und feine Granularität jedoch NP-hart ist.



(a) Orientierungen und Relationen nach Hernández (reproduziert nach [HHP93])



(b) Nachbarschaften nach Egenhofer (reproduziert nach [Ege96b])

Abbildung 2.10: Konzeptuelle Nachbarschaften

Hernández zeigt, wie z.B. die topologischen Egenhofer-Relationen nun mit *qualitativen Orientierungen* kombiniert werden können: dabei entsteht eine sog. *konzeptuelle Nachbarschaft.* Die Kno-

ten in Abb. 2.10(a) repräsentieren jeweils Paare (*Egenhofer_Relation*, *Orientierung*). Zwei Knoten A und B sind genau dann benachbart, wenn eine kleine Positionsveränderung eines der beiden in der Relation A stehenden Objekte die Relation von A nach B ändert. Die qualitativen Orientierungen bilden somit eine zyklische Struktur, und die topologischen Relationen bilden eine lineare Nachbarschaft (radial). Der Begriff der konzeptuellen Nachbarschaften wurde ursprünglich von Freksa geprägt ([Fre92b]) und wird seitdem vielfach genutzt. Unter anderem bietet er die Möglichkeit, eine Menge so benachbarter Relationen wieder zusammenzufassen und dann auf einer abstrakteren Ebene zu behandeln, wodurch Inferenzen u.U. wieder effizienter (weil weniger detailliert) durchgeführt werden können. Die topologische Distanz bzw. konzeptuelle Nachbarschaft für Egenhofer-Relationen kann ermitteln werden, indem zwischen je zwei Relationsmatrizen die Anzahl der unterschiedlichen Einträge gezählt und stets die Relationen als benachbart erklärt werden, für deren Matrizen die Differenz der unterschiedlichen Einträge minimal wird (Abb. 2.10(b)). Die topologische Distanz (bzw. die Anzahl der unterschiedlichen Matrizeneinträge) kann auch als Ähnlichkeitsmaß verwendet werden ([BE96, Ege96b]): so sind „meet“ und „overlap“ ähnlicher, als „overlap“ und „disjoint“.

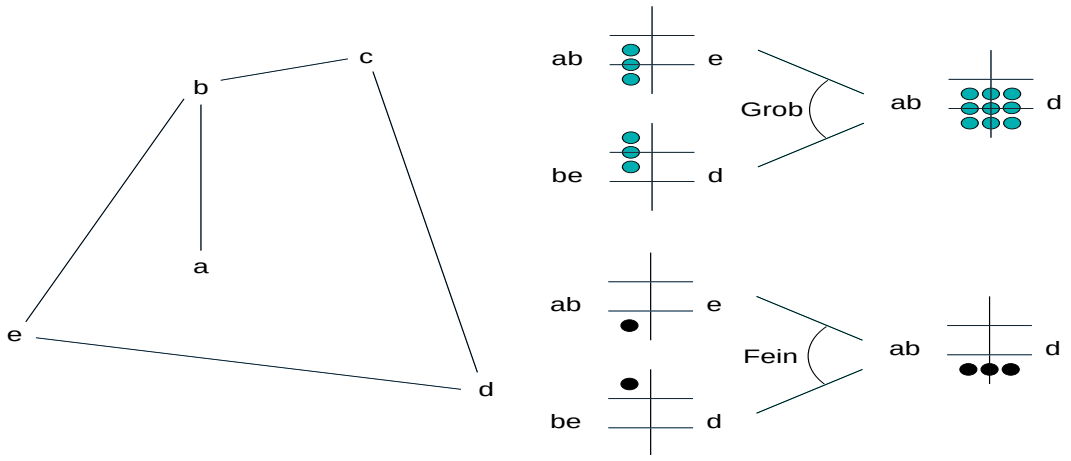


Abbildung 2.11: Komposition im Kalkül von Freksa und Zimmermann (reproduziert nach [ZF95])

Freksa und Zimmermann entwickelten einen Kalkül zum qualitativen räumlichen Schließen über *relative Orientierungen.* Sie unterscheiden 15 qualitative Orientierungen eines Punktes c, die in bezug auf einen Vektor ab ermittelt werden. Dieser Vektor kann z.B. als Weg eines (kognitiven) Agenten vom Punkt a zum Punkt b aufgefaßt werden, wobei er den Punkt c beobachtet. Alsdann werden 15 qualitative Orientierungen von Punkt c in bezug auf den Vektor ab unterschieden, die in einer ikonischen Darstellung eingezeichnet werden und textuell mit $ab : c$ bezeichnet wird (Freksa spricht in [Fre92c] von Kategorien wie „rechts vorne“, „gerade vorne“, „links vorne“ etc.). Ein eingezeichneter Punkt steht für genau eine der 15 wechselseitig-exklusiven qualitativen Relationen, die in einer konzeptuellen Nachbarschaft angeordnet werden (so sind z.B. „rechts“ und „rechts vorne“ konzeptuelle Nachbarn). Noch vagere Information läßt sich nun durch entsp. Disjunktionen dieser Relationen (also mehrere eingezeichnete Punkte) darstellen. Die Kompositionstabelle gibt nun an, welche Relation(en) $ab : d$ anhand der Kenntnis von $ab : c$ und $bc : d$ ermittelt werden können. Ein Eintrag der Kompositionstabelle ist in Abb. 2.11 dargestellt (aus [ZF95]), also die Inferenz $ab : e \wedge be : d \Rightarrow ab : d$. Freksa und Zimmermann unterscheiden zwischen *grober* und *feiner Komposition:* grobe Komposition ist ein Generalisierung feiner Komposition, indem benachbarte Relationen (im Sinne der konzeptuellen Nachbarschaften) als äquivalent betrachtet werden. Eine Erweiterung des Kalküls wurde in [ZF95] vorgenommen, indem durch Integration von Ideen des sog. Δ-Kalküls ([Zim94]) auch *qualitative Distanzen* behandelbar werden. Dabei werden die Beträge der Vektoren $|ab| = A$, $|bc| = B$ und $|ac| = C$ (bezogen auf $ab : c$) miteinander verglichen (A mit B, B mit C und A mit C) und jeweils bzgl. der Relation kleiner, größer oder gleich unterschieden.

In der Literatur zum Thema qualitatives räumliches Schließen (und Repräsentation) findet man nicht nur topologische Relationen für bzw. zwischen Gebieten, sondern es werden auch Linien (Kurven) und/oder Punkte integriert ([CDFvO93]). Neuerdings werden auch qualitative Positionen, komplexe Gebiete und beliebige Aggregate ([CDFC95]) sowie vage Gebiete (Gebiete mit unscharfem bzw. breitem Rand, ähnlich einem Spiegelei in einer Bratpfanne – Cohn und Gotts reden von „Egg-and-Yolk“) modelliert ([CDF97, ES97]) – hierbei ist festzustellen, daß die entsp. Kalküle immer komplexer werden und somit ihre Eignung für z.B. Anfragesprachen für GIS zumindest fraglich erscheint. Eine ausführliche Übersicht über dieses umfangreiche Gebiet findet man in [HM95].

2.4 Einige relevante Anwendungskontexte

Aus der großen Breite möglicher Anwendungskontexte in denen räumliche Konstellationen und Konzepte von Interesse sind, sollen hier nur drei vorgestellt werden, die im historischen Werdegang der Arbeit etwas näher betrachtet wurden. Hierzu gehören

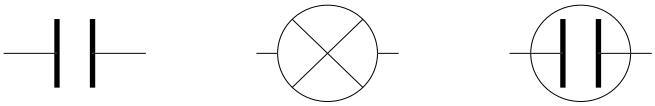
- Zeichnungsinterpretation
- Visuelle Systeme, Sprachen und Formalismen
- Geographische Informationssysteme (GIS)

Da Geographische Informationssysteme ausführlicher in Kap. 3 diskutiert werden, soll in diesem Kapitel in erster Linie das Forschungsgebiet „visuelle Sprachen“ vorgestellt und charakterisiert werden. Die Relevanz ergibt sich zum einen aus dem Titel der Arbeit, zum anderen sind die Konstrukte visueller Sprachen grafische Objekte, die somit komplexe Konstellationen bilden (Diagramme, Formulare, etc.). Zur Beschreibung der abstrakten Syntax dieser Sprachen werden oftmals qualitative räumliche Relationen verwendet, anhand derer dann z.B. automatisch ein Layout generiert werden kann. Der Bereich „Zeichungsinterpreation“ ist deswegen interessant, weil auch hier vergleichbare Probleme auftreten und gelöst werden müssen.

2.4.1 Zeichnungsinterpretation

Exemplarisch für das Forschungsgebiet „Zeichnungsinterpretation“ soll hier lediglich das von Pasternak entwickelte System „ADIK“ betrachtet werden, wobei es sich um ein System zur Interpretation technischer Zeichnungen handelt ([PN93, Pas95]). Die Bedeutung maschineller Zeichnungsinterpretation liegt insbesondere darin, daß in den Archiven vieler Firmen große Mengen technischer Papierzeichnungen lagern, die sowohl Kapital als auch über Jahrzehnte angesammeltes Wissen der Firmen repräsentieren. Eine automatische Zeichnungsinterpretation – durch Abtasten (Scannen) und anschließende automatische Nachbearbeitung – könnte dieses Datenbestände einer CAD-Verarbeitung zugänglich machen. Auschließliches Abtasten bzw. Vektorisieren dieser Datenbestände ist nicht ausreichend, da die so entstehenden unstrukturierten (Spaghetti-)Vektordaten für die meisten CAD-Systeme auf einer semantisch zu niedrigen Ebene liegen. So erfordert z.B. eine Schaltungssimulation die Kenntnis von Objektzugehörigkeiten zu Klassen (wie x ist ein Transistor, bzw. $transistor(x)$). Diese Kenntnis kann jedoch durch eine anschließende maschinelle Interpretation der durch einen Vektorisierer gelieferten Daten erhalten werden – hier soll also eine Klassifikation vorgenommen werden.

„ADIK“ (Adpatable Drawing Interpration Kernel) zeichnet sich gegenüber bisher implementierten spezialisierten Systemen zur automatischen Konvertierung spezieller technischer Zeichnungsklassen (z.B. von Kabelnetzwerkplänen) durch eine *generische Architektur* aus: der *Kern (Kernel)* kann durch verschiedene *deklarative Wissensbasen* zur Interpretation verschiedenster Zeichnungsklassen verwendet werden. Die Wissensbasis enthält dabei die für die spezielle Zeichnungsklasse



(a) Kondensator (b) Lampe (c) Glimmlampe

Abbildung 2.12: Einige Schaltzeichen

relevanten räumlichen Konzepte – für Elektronikschaltpläne z.B. Diode, Zener-Diode, Transistor, Spule, Transformator, etc.

Die entsp. Konzepte werden in einem eigens entwickelten Repräsentationsformalismus definiert. Die Sprache basiert in erster Linie auf geometrischen Beschränkungen (Constraints) und bietet sowohl taxonomische als auch kompositionale Hierarchien – ein rechtwinkliges Dreieck ist ein spezielles Dreieck und hat drei Strecken als Bestandteile. Mehrfachvererbung ist nicht vorgesehen: stattdessen werden Einstreuungen (Mixins) verwendet. Die Vererbungstaxonomie ist dual – während das Schaltsymbol einer Glimmlampe durch Spezialisierung eines Kondensatorschaltzeichens erhalten werden kann (indem ein Kreis herumgesetzt wird), ist eine Glimmlampe doch kein Kondenstator, sondern eine spezielle Lampe (s. Abb. 2.12). Pasternak unterscheidet zwischen *grafischen („graphical-is-a“)* und *funktionellen (functional-is-a)* Hierarchien. Eine Glimmlampe ist funktional eine spezielle Lampe (sie leuchtet), grafisch jedoch ein spezieller Kondensator. Auf diese Weise ist es möglich, maximal redundanzfreie Wissensbasen zu erhalten, da somit die (evtl. aufwendige) Spezifikation eines Kondensators durch Vererbung auch für Glimmlampen genutzt werden kann. Strenggenommen handelt es sich jedoch um eine Partonomie und keine Taxonomie, denn das Schaltzeichen eines Kondensators *ist Teil* des Schaltzeichens einer Glimmlampe. Hier wird Vererbung in einer Weise verwendet, von der in der Regel abzuraten ist (s. auch [GHJV96, Kap. 1]). Offensichtlich geht es in diesem Anwendungskontext jedoch primär um die Struktur bzw. den Aufbau der Objekte und nicht deren Verhalten, so daß der in „ADIK“ zu findende Umgang mit Vererbung dennoch gerechtfertigt erscheint. Klassenbildung im Rahmen der Erstellung einer Wissensbasis ist – wie immer – ein nicht-trivialer Prozeß ([Boo94, Kap. 4]), da eine Lampe funktional sowohl eine Spule als auch ein Widerstand sein könnte (Mehrfachvererbung wird jedoch nicht unterstützt).

Geometrische Basisobjekte in „ADIK“ sind Textobjekte, Punkte, Linien (bzw. beliebige Bögen, die durch Steigung der beiden Tangenten in den Endpunkten sowie die Endpunktpositionen definiert werden), Polygone sowie Kreise. Diese Objekte werden von kommerziellen Vektorisierern extrahiert (mit Ausnahme von Polygonen). Als geometrische Basiseigenschaften können in der Sprache nun Attribute wie Position, Orientierung, Ausdehnung, Radius, Länge, Textinhalt etc. von Objekten definiert werden. Die Basisrelation ist die sog. „LOC“-Relation: sie beschreibt die Orientierung und Distanz zwischen Punktobjekten (z.B. Endpunkten von Linien). Da es sich um eine quantitative (metrische) Relation handelt, kann man sie formal als attributierte Relation betrachten (auf den Relationstupeln sind die Funktionen Orientierung und Distanz definiert). Komplexe geometrische Relationen und Eigenschaften können nun durch Kombination dieser Basiseigenschaften und -relationen definiert werden.

In „ADIK“ können Eigenschaften bzw. Relationen sowohl *konstant, variabel* als auch *funktional* eingeschränkt werden. Eine *konstante Einschränkung* erfordert einen speziellen Wert einer Eigenschaft (z.B. soll eine spezielle Linie immer 10 cm lang sein). Durch *variable Einschränkungen* können verschiedene Eigenschaften miteinander verknüpft werden (z.B. sollen zwei Linien die gleiche – aber variable – Länge haben). Hierzu werden entsp. Variablen in den beschreibenden Sprachausdrücken gebunden. Schließlich können *funktionale Einschränkungen* verwendet werden, um beliebige funktionale Abhängigkeiten zwischen gebundenen Variablen definieren zu können

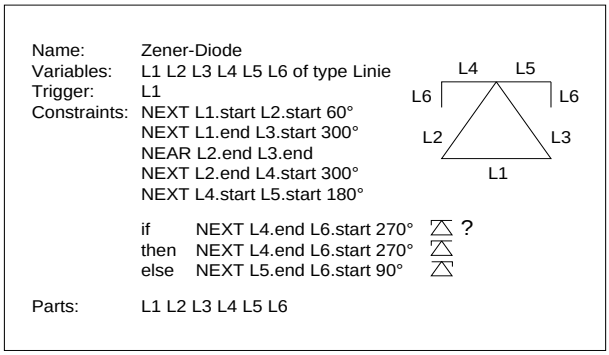


Abbildung 2.13: Spezifikation einer Zener-Diode (reproduziert nach [Pas95])

(z.B. soll die variable Länge von Linie a immer doppelt so lang sein wie die variable Länge von Linie b). In der Sprache können insbesondere auch rekursive Muster beschrieben werden. Das Schaltzeichen einer Spule z.B. kann eine beliebige Anzahl von Spulenwindungen haben. Rekursive Spezifikationen machen es möglich, die Klasse aller Spulenschaltzeichen durch eine Spezifikation zu beschreiben: während die Sprache also in erster Linie geometrische und partonomische Definitionen erlaubt, könnten durch rekursive Spezifikationen auch gewisse topologische Einschränkungen (wie z.B. „verbunden mit“) überprüft werden.

Eine so erstellte deklarative Wissensbasis wird schließlich compiliert („ADIK“ ist in LISP realisiert) und die Interpretation eines vektorisierten Datenbestandes gestartet. Das Verarbeitungsmodell ist eine „Blackboard“-Architektur: die Spezifikationsüberprüfung (Geometrieüberprüfung) von Kandidaten wird durch sog. „Trigger“-Ereignisse angestoßen. So kann z.B. die Überprüfung einer Dreiecks-Hypothese beginnen, wenn eine der Linien vorliegt. Die anderen beiden Linien würden dann anhand der Dreiecksspezifikation gezielt unter Verwendung eines *räumlichen Indizierungsverfahrens* gesucht. Hierfür sieht Pasternak *Buckets* und *R-Bäume* vor. Besteht ein Kandidat nun die Spezifikations- bzw. Geometrieüberprüfung, so ist er Instanz des entsp. Konzeptes. Die Trigger müssen vom Ersteller der Wissensbasis angegeben werden, wodurch die Sprache nicht mehr als rein deklarativ einzustufen ist, denn hier wird – wenn auch deklaratives – Kontrollwissen kodiert: u.a. muß der Ersteller der Wissensbasis dafür Sorge tragen, daß das Verfahren terminiert (man denke an zyklische Ketten von Triggerereignissen).

Abschließend ist zu bemerken, daß durch den generischen Kernel und die Möglichkeit, hochgradig anwendungsspezifische Wissensbasen mit entsp. anwendungsadäquatem Kontrollwissen zu definieren, eine maximale Effizienz erzielt werden kann. Hierzu tragen auch die sehr quantitativen bzw. wenig abstrakten verwendeten Basisrelationen bei. Ihre geometrisch-metrischen Eigenschaften können sehr effizient sowohl überprüft als auch zur Kandidatengenerierung verwendet werden. Die Sprache ist jedoch zu kompliziert und abstrakt, als daß ein Endbenutzer seine eigenene Wissensbasis erstellen könnte. Dies ist im vorliegenden Anwendungskontext zwar nicht unbedingt notwendig, wäre aber natürlich von Vorteil. Insbesondere die rekursiven Definitionen sind von großem Vorteil (man denke z.B. an Polygone oder Ketten) und steigern die Ausdrucksmächtigkeit sehr. „ADIK“ bietet viele weitere Möglichkeiten zur Definition von Strukturvariabilität, die hier nicht dargestellt werden konnten (u.a. können geometrische Definition von Oberklassen vereinigt, geschnitten und/oder subtrahiert werden, versch. Sichten definiert werden, etc.). Abb. 2.13 zeigt die (für sich selbst sprechende) Spezifikation einer Zener-Diode. Man beachte, daß diese geometrische Spezifikation sowohl translations-, skalierungs- als auch rotationsinvariant ist. Zudem ist eine Strukturvariabilität in Form des „If-Then-Else“-Konstruktes vorgesehen.

2.4.2 Visuelle Systeme, Sprachen und Formalismen

Da es in dieser Arbeit um die Entwicklung einer *visuellen Sprache* geht, soll dieser Abschnitt etwas ausführlicher ausfallen. Wiederum liegt der Fokus der folgenden Darstellung nicht auf einzelnen visuellen Sprachen oder Systemen, sondern auf verwendbaren Definitionen und Begriffen.

S.-K. Chang definiert eine visuelle Sprache in [Cha90, S. vi] folgendermaßen:

> By visual language we mean the systematic use of visual expressions to convey meaning. Therefore, a visual programming language is a visual language. On the other hand, a language supporting visualization is also a visual language, although such a language may not be a programming language.

Eine vergleichbare Definition verwenden auch Catarci et al. in ([CMS91]). Zusätzlich wird hier der Begriffes des visuellen Systemes folgendermaßen festgelegt:

> We call **visual system** any system providing a visual man-machine interaction. Such integration is typically based on a **visual language**. In order to give a concise definition of visual language, we may say, according to the point of view of many authors . . . , that it implies the systematic use of visual expressions (such as icons, drawing or gestures) to convey a meaning in a formal way.

Visuelle Sprachen sind also Sprachen, die grafische Sprachelemente auf systematische und somit formalisierbare Weise verwenden, um komplexe Bedeutungen zu kommunizieren. Die Sprachen dienen typischerweise zur systematischen Kommunikation mit einem *visuellen System,* z.B. zur Programmierung mit visuellen Ausdrücken oder Abfrage einer Datenbank, aber auch zur Visualisierung der Dynamik komplexer Algorithmen oder der Darstellung des Datenbestandes einer Datenbank – beide Kommunikationsrichtungen werden also berücksichtigt. Es handelt sich somit um einen sehr weitgefassten Begriff, der nur durch entsp. Teilklassenbildung gewinnbringend anwendbar ist.

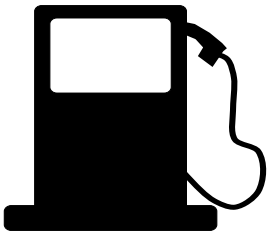
Prinzipiell sollte eine visuelle Sprache - sofern sie zur Kommunikation mit Rechnern bzw. rechnergestützten visuellen Systemen vorgesehen ist – *visuelle Formalismen* als bedeutungstragende Entitäten verwenden. Der Begriff „visueller Formalismus“ wurde von Harel geprägt und bringt zum Ausdruck, daß visuelle Entitäten in einem Rechnerkontext nur dann sinnvoll verwendet werden können, wenn sie sowohl ausdrucksstark und problemadäquat für den menschlichen Benutzer als auch formal definierbar sind, da die visuellen Darstellungen von einem Rechner verarbeitet werden müssen ([Har88]):

> The intricate nature of a variety of computer-related systems and situations can, and in our opinion should, be represented by *visual formalisms:* visual, because they are to be generated, comprehended, and communicated by humans; and formal, because they are to be manipulated, maintained, and analyzed by computers.

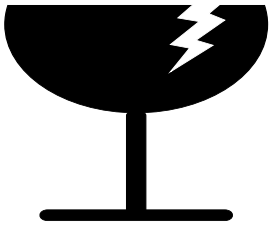
Der verwendete visuelle Formalismus ist ein wesentliches Charakteristikum visueller Sprachen. Prinzipiell werden

- Ikonen (Icons),
- Diagramme, und
- Formulare (Tabellen etc.)

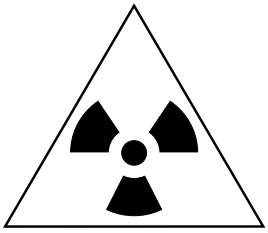
für visuelle Sprachen verwendet. Oft findet man auch hybride Systeme, die z.B. Ikonen und Diagramme verwenden ([CMS91, CCLB97]).



(a) Repräsentationales Ikon



(b) Abstraktes Ikon (Symbol)



(c) Arbiträres Ikon (Zeichen)

Abbildung 2.14: Einige Ikonen (reproduziert nach [Lod83])

Zwei Definitionen für den Begriff *Ikon (Icon)* sind „a visually segmented object wich tells the viewer about an inside message or information (concept, function, state, mode, etc.) assigned by the designer ([FK91])“ und „a perceptible (physical) pattern to which a community of users agrees to asign a meaning so that it can be exploited for human communication and reasoning (P. Mussio)“. Ikonen werden z.B. in Form von Piktogrammen häufig in unserer Gesellschaft verwendet – insbesondere in der zweiten Definition kommen semiotische Aspekte (wie Syntax, Semantik und Kontext), wie sie zur Interpretation von Ikonen oftmals benötigt werden, gut zum Ausdruck. Eine gängige Klassifikation unterteilt die Klasse der Ikonen in ([Lod83])

- repräsentationale bzw. bildhafte Ikonen (auch Piktogramme genannt),
- abstrakte bzw. symbolische Ikonen, sowie
- arbiträre bzw. zeichenhafte Ikonen.

Ein *repräsentationales Ikon (Piktogramm)* repräsentiert in der Regel durch die Darstellung eines prototypischen (stilisierten vereinfachten) Exemplares eine ganze Klasse von Objekten. Anhand von Abb. 2.14(a) wird deutlich, daß lediglich die wesentlichsten funktionalen und visuellen Aspekte bzw. Charakteristika stark stilisiert und vereinfacht dargestellt werden. Sowohl das Erlernen als auch der Entwurf dieser Ikonen wird als einfach angesehen. Ein *abstraktes Ikon* kommuniziert ein Konzept, für welches es kein offensichtliches visuelles Erscheinungsbild gibt – so bringt z.B. das Ikon in Abb. 2.14(b) das abstrakte Konzept „Zerbrechlichkeit“ zum Ausdruck. Der Umgang mit und Entwurf von abstrakten Ikonen wird schon als schwieriger angesehen, da es für die zu vermittelnde Botschaft kein offensichtliches visuelles Erscheinungsbild gibt, das Ikon aber als *passend* für die Botschaft empfunden werden muß. Ein *arbiträres Ikon* steht schließlich in einer Beziehung zur vermittelten Botschaft, die lediglich durch Konvention vorhanden ist. So wurde z.B. das Radioaktiv-Ikon (Abb. 2.14(c)) *erfunden* und per Konvention die Bedeutung „Achtung! Radioaktiv!“ zugewiesen. Der Lernaufwand für die einzelnen Ikonen steigt in der Reihenfolge der obigen Aufzählung, da die Ikonen zunehmend auf Ähnlichkeit mit entsp. (im Kontext auftauchenden) „Objekten“ verzichten, ähnlich der historischen Entwicklung der Schriftzeichen mancher Sprachen – es gibt somit verschiedene Abstraktheitsgrade, und Konventionen und Kontext spielen für die richtige Interpretation von Ikonen in der Reihe obiger Aufzählung eine zunehmend wesentlichere Rolle. In der Literatur findet man die Aussage, das Ikonen starke *metaphorische Kraft* besitzen: tatsächlich ist diese sogar notwendig, da Ikonen anderseits aufgrund ihrer Abstraktheit meist nicht interpretiert werden könnten. Hingegen haben Diagramme (oder im Extremfall Photos) nahezu gar keine metaphorische Kraft.

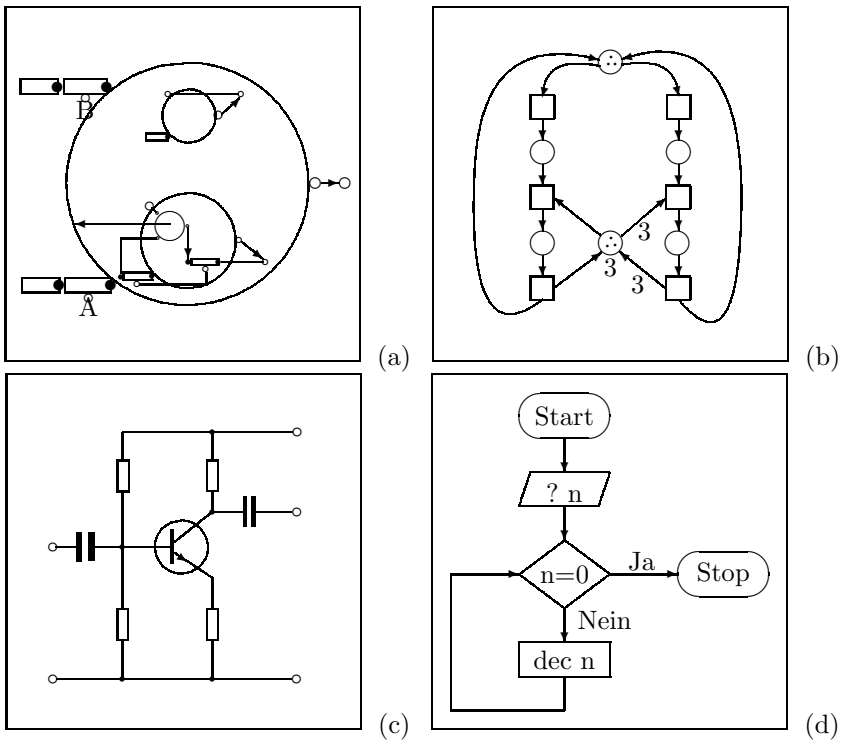
Im Sinne einer *ikonischen Sprache* ist man daran interessiert, durch *Kombination* von Ikonen *ikonische Sätze* zur erzeugen, deren Bedeutung sich dann *kompositional* durch die verwendeten Ikonen bzw. Bestandteile ergibt. Ansätze hierzu finden sich in den Ikonen des Apple Macintosh, aber auch in Verkehrszeichen: Warnschilder sind stets dreieckig und rot umrandet. Große Systematik ist auch in *Elektronikschaltzeichen* zu beobachten: ein Potentiometer (Drehwiderstand, Abb.

(a) Kondensator (b) Widerstand (c) Drehkondensator (d) Drehwiderstand

Abbildung 2.15: Einige Schaltzeichen

2.15(d)) enthält als Teilbild das Schaltzeichen eines Widerstandes (Abb. 2.15(b)). Ein Drehkondensator (Abb. 2.15(c)) hat mit einem Drehwiderstand den Pfeil (er symbolisiert Veränderbarkeit) gemeinsam, und enthält ebenfalls das Kondensatorsymbol (Abb. 2.15(a)).

Chang entwickelte eine *Theorie generalisierter Ikonen (Generalized Icon Theory)*, s. [Cha87, Cha90]: ein generalisiertes Ikon besteht aus einem *physikalischen Teil (dem Bild)* und einem *logischen Teil (der Bedeutung)* und wird als Tupel (Xm, Xi) notiert. *Ikonische Operatoren* werden nun verwendet, um komplexe Konstellationen zu erzeugen, deren Bedeutung sich je nach Operator kompositional ergibt. Ein generischer visueller Compiler-Compiler („SIL-ICON“, s. [CTBY89]) für ikonische Sprachen konnte anhand dieser Theorie entwickelt werden. Ähnlich wie der in der „UNIX“-Welt bekannte Compiler-Compiler „YACC“ kann hier anhand einer formalen Ikon-Grammatik ein Compiler für eine spezielle ikonische Sprache generiert werden. Eine ausführliche Diskussion von Ikonen kann hier nicht erfolgen. Schließlich sei noch erwähnt, daß es auch viele negative Erfahrungen im Umgang mit Ikonen (speziell im Rechnerkontext) zu berichten gibt ([Pot88]).



(a) (b) (c) (d)

Abbildung 2.16: (a) „Pictorial Janus“-Programm, (b) Petrinetz, (c) Schaltplan und (d) Flußdiagramm

Oben wurden noch *Diagramme* und *Formulare (z.B. Tabellen)* als relevante visuelle Formalismen erwähnt. Formulare und Tabellen sind im Rahmen dieser Arbeit nicht relevant. Unter einem

Diagramm versteht J. Bertin folgendes (aus [CCLB97]):

> The graphic is a diagram when the correspondences on the plane can be established among all the elements of one component and all the elements of another component.

Wesentlich ist also, daß räumliche Relationen im Diagramm zwischen allen Objektpaaren etabliert werden können, und diese Relationen können Bedeutungen tragen. In der Informatik werden in erster Linie graphenartigen Diagramme (z.B. Semantische Netze, ER-Diagramme) verwendet. Die relevanten Relationen werden in diesen Diagrammen durch linienhafte Verbindungen bzw. Kanten zwischen diskreten Objekten dargestellt. Aber auch Graphen mathematischer Funktionen oder Geschäftsgrafiken wie Balken- und Tortendiagramme werden als Diagramme bezeichnet, wodurch die Breite des Begriffes deutlich wird. Der Wert von Diagrammen wird in [LS95] gewürdigt, formale Aspekte und das Schließen mit ihnen (Diagrammatic Reasoning) in [WLZ95]. Diagrammbasierte (diagrammatische) visuelle Sprachen zeichnen sich gegenüber ikonischen Sprachen durch eine noch größere Vielfalt sowohl in den verwendeten grafischen Entitäten als auch in den relevanten Relationen zwischen diesen aus (s. Abb. 2.16). Während Ikonen zur Kommunikation von Konzepten oder Botschaften verwendet werden, dienen Diagramme in erster Linie zur Darstellung komplexer Verhältnisse bzw. *relationaler Sachverhalte* zwischen Objekten. Diagramme stellen die weitaus größte Klasse echter visueller Formalismen dar: hierzu gehören ER-Diagramme, Darstellungen endlicher Automaten und Petrinetze, aber auch die aus der Mengenlehre bekannten Venn-Diagramme und andere visuelle Logik-Formalismen wie Freges „Begriffsschrift“ (1879) oder die „Existential Graphs“ von C. S. Peirce (1896). Diagrammatische visuelle Programmiersprachen wie „Pictorial Janus“ von K. Kahn ([KS90, Kah90], s. Abb. 2.16(a)) oder die „Higraphs“ von D. Harel ([Har88]) sind ebenfalls vollwertige visuelle Formalismen.

In der Definition von Chang kamen bereits die Begriffe *visuelle Programmiersprache (Visual Programming Language, VPL)* und *Sprache zur Erzeugung von Visualisierungen* oder auch *Visualisierungssprache (Language Supporting Visualizations)* vor. Offensichtlich handelt es sich hier um Teilklassen visueller Sprachen. Es gibt jedoch eine Vielzahl weiterer Kategorien: während visuelle Sprachen oftmals auch anhand des zugrundeliegenden visuellen Formalismuses charakterisiert werden (s.o.), ist der *Zweck* der visuellen Sprache natürlich eine weiteres Unterscheidungsmerkmal. Chang unterscheidet zwischen vier großen Kategorien visueller Sprachen (s. [Cha87]):

- **Sprachen zur Unterstützung visueller Interaktion (Languages that Support Visual Interaction)** sind Sprachen, bei denen logisch-abstrakten Objekten visuelle Repräsentationen zugeordnet werden. Durch die visuellen Erscheinungsformen und evtl. direkt-manipulativen Umgang mit diesen soll die Benutzungsfreundlichkeit gesteigert werden. Die Konstrukte der Sprache selbst sind jedoch meist textuell: der Benutzer muß sowohl Erscheinungsform als auch die Interaktion mit den visuellen Objekten textuell spezifizieren bzw. programmieren. UIMS (User Interface Management Systemes) gehören in diese Kategorie, wie z.B. das CLIM-Framework (Common LISP Interface Manager). Zusehends verdrängen spezielle visuelle Werkzeuge (Interface Builder) die textuellen Beschreibungssprachen zur Festlegung von Erscheinung und Interaktion. Aber auch spezielle Sprache z.B. zur Algorithmenanimation gehören in diese Kategorie.

- **Visuelle Programmiersprachen (Visual Programming Languages, VPLs)** streben die Mächtigkeit konventioneller textueller Programmiersprachen an (Turing-Vollständigkeit). Auch diese Sprachen handeln von logisch-abstrakten Objekten, wie z.B. Variablen, Feldern, Listen etc., für die adäquate Visualisierungen gefunden werden müssen. Hierfür bieten sich visuelle Formalismen an – hier werden also komplexe visuelle Ausdrücke bzw. visuelle Sätze konstruiert, um Algorithmen zu beschreiben (man spricht auch von *ausführbaren Grafiken*). Die meisten visuellen Programmiersprachen sind für Ausbildungszwecke entwickelt worden (oftmals z.B. mit dem Anspruch, Kindern das Programmieren beizubringen). Prinzipiell scheint es jedoch Probleme in der Visualisierung und Behandlung abstrakterer Sachverhalte zu geben (z.B. benutzerdefinierte Datentypen). Hier ist auch das *Progammieren durch*

Beispiele oder Demonstration (Programming By Examples / Demonstration) zu erwähnen – da hier jedoch ein Programm durch Interaktionen bzw. Gesten des Benutzers erzeugt wird, handelt es sich strenggenommen nicht um eine visuelle Programmierung. Dennoch spielt hier eine *visuelle Sprachumgebung* eine wesentliche Rolle.

- **Sprachen zur Verarbeitung visueller Informationen (Visual Information Processing Languages)** sind Sprachen, deren zugrundeliegende Datenobjekte *inhärent visuelle Erscheinungsformen* haben, z.B. Bilder einer Bilddatenbank oder Geo-Objekte in einem GIS. So werden z.B. in der Bildverarbeitung spezielle Datenflußsprachen zur pipeline-artigen Verarbeitung von Bildobjekten verwendet (z.B. „Khoros“). Aber auch räumliche Anfragesprache (wie z.B. für GIS-Zwecke erweitertes SQL) gehören in diese Kategorie. Laut Chang handelt es sich hierbei in erster Linie um textuelle Sprachen, die spezielle Konstrukte zur Behandlung der inhärent räumlichen Datenobjekte aufweisen. Schließlich lassen sich die Konstrukte dieser Sprachen aber auch wieder entsp. visualisieren, so daß man zur Kategorie der

- **Ikonischen Sprachen zur Verarbeitung visueller Informationen (Iconic Visual Information Processing Language)** kommt. Hierzu gehören insbesondere die *visuellen räumlichen (ikonischen) Anfragesprachen ([Iconic] Visual Spatial Query Languages),* die in Kap. 4 untersucht werden. Da Chang einen generalisierten Ikon-Begriff verwendet (s.o.), fallen jedoch nicht nur ikonische, sondern auch diagrammbasierte visuelle Sprachen in diese Kategorie. Auch für Zwecke der Bildverarbeitung existieren spezielle visuelle Systeme, die nun wiederum die Visualisierung und Interaktion mit den inhärent räumlichen Objekten (den Bildern) visuell und direkt-manipulativ unterstützen (für die textuelle Sprache bzw. Bibliothek „Khoros“ – s.o. – existiert eine ikonische visuelle Oberfläche namens „Cantata“).

Chang beschreibt alle vier Kategorien visueller Sprachen durch seine Theorie generalisierter Ikonen. Bei der Diskussion wurde deutlich, daß die einzelnen Kategorien visueller Sprachen nicht disjunkt sind und auch Sprachen, deren Konstrukte textuell sind, als visuell klassifiziert werden. Im Kontext dieser Arbeit sollen vor allem Sprachen der letztgenannten Kategorie (Iconic Visual Information Processing Languages) betrachtet werden (s. Kap. 4). Zudem verwende ich den Begriff *visuelle Sprache* ausschließlich für Sprachen, deren Konstrukte überwiegend grafisch-visuell sind (dies wurde bereits in Kap. 1 deutlich), so daß einige der obigen Kategorien aus der Betrachtung ausgeschlossen werden.

Eine vergleichbare Klassifikation findet man auch bei Shu ([Shu86]): sie unterscheidet – ähnlich wie Chang – zwischen Sprachen, die

- visuelle Interaktion unterstützen,
- das Programmieren mit visuellen Ausdrücken ermöglichen, sowie Sprachen zur
- Verarbeitung visueller Informationen.

Generell halte ich es für angebrachter, statt von visueller Information von räumlichen Daten zu reden (s. Kap. 3). Auch die in [Shu86] vorgenommene Klassifizierung von „Query-by-Example (QBE)“ als visuelle *Programmiersprache* scheint problematisch, da es pragmatisch hier nicht um Algorithmenformulierung, sondern Datenbankabfrage geht.

Erwartungen und Warnungen

Motiviert wird die Beschäftigung mit visuellen Sprachen oftmals durch kognitive Betrachtungen: der inzwischen zur Parodie verkommene Satz „Ein Bild sagt mehr als 1000 Worte“ wird immer wieder gerne als primäre Begründung verwendet. Letzlich verspricht man sich also, die *hohe Effizienz und Bandbreite des menschlichen visuellen Systemes zur effektiven Kommunikation mit Rechnern auszunutzen.* Abstrakte und komplizierte Zusammenhänge können von Menschen – wenn sie in

konkrete, anschauliche Formen gebracht, also adäquat *visualisiert* werden – wesentlich leichter aufgenommen und verarbeitet werden als ausschließlich textuelle lineare Darstellungen. Die Kompetenz und Effektivität des Menschen im Umgang mit Raum und räumlichen Konzepten wurde ja bereits oben zur Sprache gebracht.

Shu führt in [Shu86] folgende Motive an:

1. People, in general, prefer pictures over words.
2. Pictures are more powerful than words as a means of communication. They can convey more meaning in a more concise unit of expression.
3. Pictures do not have the language barriers that natural languages have. They are understood by people regardless of what language they speak.

Visuelle Sprachen werden manchmal als Schlüsselfaktor gesehen, der End- und/oder Gelegenheitsbenutzern (die keine EDV-Spezialisten sind) u.a. die Adaptierbarkeit komplexer Softwaresysteme an individuelle Anforderungen ermöglichen soll. Neben der Adaptierbarkeit kommerzieller Softwaresysteme soll dieser Benutzergruppe – und teilweise sogar Kindern – auch das Programmieren eigener Anwendungen ermöglicht werden.[5] Der Prozentsatz an EDV-Spezialisten unter den Benutzern der Produkte unserer „Informationsgesellschaft“ wird immer weiter sinken: die Benutzbarkeit dieser rapide fortschreitenden Technologien muß somit durch entsprechende Maßnahmen gesichert werden (s. [Pos95]). Shu bringt es folgendermaßen auf den Punkt:

> Thus, the real bottleneck in access to computing is in the ease of programming. The challenge is to bring computer capabilities simply and usefully to people whose work can be benefited by programming.

Die These, daß visuelle Systeme (bzw. Sprachen) hier evtl. einen großen Beitrag leisten könnten, wird vor allem von der Beobachtung gestützt, daß die grafischen Benutzungsoberflächen (in Zusammenhang mit immer leistungsfähigerer Hardware und sinkenden Hardwarekosten) einen großen Einfluß auf die Verbreitung der Personalcomputer hatte (und hat). Zentraler Punkt bei den grafischen Benutzungsoberflächen ist hierbei die direkt-manipulative Interaktion ([Shn83]) mit grafischen Objekten in Zusammenhang mit der starken Metapher des virtuellen Schreibtisches.

Graf gibt eine Reihe von Empfehlungen für das Design visueller Sprachen und warnt eindrücklich davor, visuelle Sprachen als Antwort auf alle Fragen und Probleme anzusehen ([Gra90]). Wesentlich seien ein zugrundeliegendes einheitliches Prinzip für die Sprache, konzeptuelle Einfachheit (wenige Konstrukte, Orthogonalität, wenige Sonderregeln), sowie eine sehr gute, HCI (Human Computer Interaction)-Anforderungen genügende Benutzungsoberfläche, die nach Möglichkeit auch Animationen verwendet. Er warnt vor mit Ikonen überladenen Oberflächen und dogmatischen Entscheidungen: oftmals sind Texte adäquater als Grafiken und sollten dann auch verwendet werden. Es darf nicht vergessen werden, daß Menschen in der Regel jahrelange Erfahrung im Umgang mit Texten haben, während dies für neu erschaffene visuelle (z.B. ikonische) Sprachen zunächst nicht der Fall ist. Das es sich prinzipiell aber nur um eine Frage des Trainings handelt, offenbart ein Blick auf chinesische oder japanische Schriftzeichen. Formale Aspekte seien beim Entwurf einer neuen visuellen Sprache zunächst nicht vordergründig, sondern könnten inkrementell entstehen: oberstes Ziel sollte die Nützlichkeit der visuellen Sprache und der ihr zugrundeliegenden Konzepte sein, nicht die Theorie. Das zugrundeliegende Prinzip muß jedoch einer - evtl. anschließenden – Formalisierung standhalten. Eine gute visuelle Spezialsprache sollte nicht zu einer mit heterogenen Konzepten überladenen Universalsprache gemacht werden. Stattdessen sollte man das Design an der Stelle abbrechen, wo die Sprache unhandlich, unübersichtlich oder einfach zu kompliziert wird und somit ihre Schönheit, Eleganz und Orthogonalität verliert („Quit While Winning“). Von

[5] S. Papert führte diesbezüglich Experimente mit Kindern im Kindergartenalter unter Verwendung der Programmiersprache „LOGO“ durch.

vielen Forschern wird die Auffassung vertreten, das die erfolgsversprechensten Kandidaten für visuelle Sprachen in der Klasse der Spezialsprachen für spezielle Anwendungen (z.B. Information Retrieval) und nicht in der Klasse der universellen Programmiersprachen gefunden werden.

Formalisierungsbemühungen

Die Suche nach einer breit anwendbaren formalen Basis für visuelle Sprachen wird in erster Linie durch die Breite der möglichen Ausprägungen visueller Sprachen erschwert: während bei textuellen Sprachen die bedeutungstragenden Entitäten stets die selben sind (nämlich Wörter bzw. Sätze einer formalen – meist kontextfreien – Grammatik), liegt bei visuellen Sprachen eine potentiell größere Varianz möglicher visueller Formalismen vor (Ikonen, Diagramme, Formulare, Tabellen, Texte, ...). In der Regel werden formale Modelle für einen paradigmatischen visuellen Formalismus entwickelt. Fortschritte wurden in der Vergangenheit in erster Linie für ikonische Sprachen erzielt. Zudem ist festzustellen, daß in erster Linie Probleme des visuellen Parsings formalisiert wurden. Entsprechende Semantikzuweisungen ergeben sich dann meist durch Abbildung auf eine textuelle Sprache bzw. einen speziell geschaffenen logischen Formalismus.

Im Gegensatz hierzu werden Syntaxbeschreibungen für textuelle Sprachen heutzutage nahezu immer in einer erweiterten Backus-Naur-Form (EBNF) angegeben. Eine rechnerinterne Repräsentation (z.B. Ableitungsbaum) eines Programmes (einer beliebigen Programmiersprache) kann in der Regel durch das Standardvorgehen „lexikalische Analyse – syntaktische Analyse“ mit Hilfe von Standardalgorithmen erzeugt werden ([ASU88]). Das entsp. Vorgehen ist heutzutage (für vegleichbare Klassen textueller Programmiersprachen) so einheitlich, daß in der Regel sogar generische Compiler-Compiler verwendet werden (z.B. „YACC“). Aber auch die (dynamische) Semantik textueller Programmiersprachen läßt sich in der Regel unter Verwendung translationaler, operationaler, axiomatischer oder denotationaler Semantik spezifizieren ([Mey90]). So läßt sich z.B. der Hoare-Kalkül prinzipiell sowohl zur Semantikbeschreibung von Algol als auch Pascal oder C nutzen (alle drei sind imperative Programmiersprachen). Während LISP in der Vergangeheit operational definiert wurde, wird heutzutage in der Regel die „elegantere“ denotationale Semantik verwendet (z.B. für Scheme). Dies zeigt, daß sowohl die Formalismen zur Syntax- als auch zur Semantikbeschreibung im Bereich textueller Sprachen breit anwendbar sind (es läßt sich jedoch eine gewisse Präferenz bestimmter Paradigmen – imperativ, funktional, logisch, objektorientiert – für bestimmte Semantikbeschreibungen feststellen).

Offensichtlich besteht eine *starke Korrespondenz zwischen textuellen und visuellen Sprachen:*

- **Atome:** eindimensionale Wörter ⇔ mehrdimensionale Ikonen, Zellen in Formularen, Diagrammelemente (z.B. Kreise eines ER-Diagrammes).
- **Symbolische Komplexe:** Sätze durch Aneinanderreihung von Wörtern ⇔ komplexe Konstellationen durch intrinsische oder extrinsische Relationen zwischen entsp. Atomen.

Im letzten Punkt liegt der wesentliche Unterschied zwischen textuellen und visuellen Sprachen: die *eindimensionale Aneinanderreihung (Konkatenation) von Wörtern* ist in der Regel die einzige bedeutungstragende intrinsische Relation in textuellen Sprachen (textuelle Sprachen sind *eindimensional*; eine Ausnahmen bildet z.B. die Sprache „OCCAM“, bei der auch Einrückungen bzw. eine visuelle Blockstruktur bedeutungstragend ist - solche Sprachen können als 1,5-dimensional bezeichnet werden), während in visuellen Sprachen mehrere bedeutungstragende Relationen sowohl extrinrinsisch (z.B. durch Kanten-Verbindungen in Diagrammen) als auch intrinsisch repräsentiert werden können. Aufgrund der Zweidimensionalität kann ein Objekt viele Relationen gleichzeitig eingehen und somit vielerorts referenziert werden; oftmals entfällt für diesen Zweck die Notwendigkeit von *Bezeichnern.*

Aufgrund der starken Korrespondenz syntaktischer Aspekte wurden (und werden) diverse Grammatikformalismen entwickelt: sie beschreiben ein Bild textuell, indem sie die möglichen räumlichen

Relationen zwischen Objekten entweder *symbolisch explizit machen* oder die Terminale der Grammatik entsp. *attributieren,* so daß die entsp. Relationen *berechnet* werden können *(implizite räumliche Relationen).* In diesem Fall redet man auch von *attributierten Multimengen-Grammatiken (Attributed Multiset Grammars, AMGs).* Die möglichen visuellen Symbole werden durch Terminale der Grammatik beschrieben (wie z.B. „Kreis", „Pfeil" etc.), und ihre relevanten Attribute als entsp. Funktionen auf diesen Objekten repräsentiert (wie z.B. „Position", „Radius" etc.). Ein *Bild* ist dann einfach eine *attributierte Multimenge* dieser Terminale. Die *Produktionen* einer derartigen Grammatik sind in der Regel *kontextsensitiv* und nur dann anwendbar, wenn zusätzliche Einschränkungen (Constraints) über den Attributen der Kontextobjekte erfüllt sind (z.B. kann eine „Verbinde"-Produktion zwei Kreise – dies ist der Kontext – nur dann über einen Pfeil verbinden, wenn dieser die Ränder der beiden Kreise berührt). Spezielle AMGs sind *„Picture Layout Grammars" ([Gol91, GR89])* und *„Constraint Multiset Grammars" ([Mar94, CM95, HM90]).*

Prinzipiell gibt es bei eindimensionalen Grammatiken das Problem, daß lediglich *Ableitungsbäume* dargestellt werden können. Da aber Objekte in Diagrammen i. d. R. mehrere Relationen eingehen können, führt dies zu einer Vervielfachung des referenzierten Objektes im Ableitungsbaum – es wird im Ableitungsbaum in mehreren Blättern bzw. Terminalen repräsentiert. Diese müssen dann evtl. wieder zusammengefaßt werden.

Ein inhärent zweidimensionaler Formalismus, in dem obiges Problem daher nicht auftritt, sind *Graphgrammatiken:* statt einzelne Nichterminale durch die rechten Seiten von Grammatikproduktionen in einer eindimensionalen Zeichenkette zu ersetzen, bestehen die linken und rechten Seiten von Produktionen in Graphgrammatiken aus *Graphen.* Komplexe Graphen werden dann – ausgehend von einem initialen Graphen – durch schrittweise Anwendung von Produktionen aufgebaut, indem linke Teilgraphen gegen rechte Teilgraphen von entsp. Produktionen ersetzt werden ([RS96, RS95, MV95, Min97]).

Als Beispiel eines relativ allgemein einsetzbaren formalen Rahmenwerkes im Bereich visueller Sprachen soll der in [BCLM95, BCLM97] von Bottoni et al. vorgeschlagene Formalismus vorgestellt werden: er erlaubt die Präzisierung der Begriffe „visuelle Sprache", „visueller Satz", „visuelles Parsing", „Interpreation" und „Visualisierung". Basisobjekte dieses Formalismus sind sog. *digitale Bilder (Digital Images).*

Definition (Digitales Bild): Ein *digitales Bild (Digital Image) i* ist eine Funktion $i : 1 \dots r \times 1 \dots c \rightarrow P$, wobei P das sog. Pixel-Alphabet ist. Ein Pixel ist ein Tripel (r, c, p), so daß $i(r, c) = p$.

Definition (Piktorielle Sprache): Eine *piktorielle Sprache (Pictorial Language) PL* ist eine Teilmenge der möglichen digitalen Bilder I: $PL \subseteq I$. Es gilt: $I = \{i \mid \exists r, c \in \mathbb{N}\ (i : 1 \dots r \times 1 \dots c \rightarrow P)\}$. (Anmerkung: analog hierzu wird ja im Bereich der formalen Sprachen eine Sprache L als Teilmenge der möglichen Wörter über dem Alphabet A definiert, also $L \subseteq A^\star$. Da hier noch nichts über Semantik ausgesagt wird, scheint mir der Begriff „Bildersammlung" passender - auch die Bezeichung „formale Sprache" wurde schon vielfach kritisiert, s. z.B. [Sch87]).

Statt piktorieller Sprache wird oftmals der Begriff *konkrete Syntax* gebraucht: hierbei handelt es sich also um die Menge aller möglichen konkreten Bilder einer Sprache. Auch eine spezielles Bild i wird als konkrete Syntax bezeichnet.

Definition (Attributiertes Symbol): Ein *attributiertes Symbol (Attributed Symbol)* ist ein Tupel $as \in W = V \times P_{a1} \times P_{a2} \times \dots \times P_{am}$, wobei V ein Alphabet von Symbolen und $P_{a1} \dots P_{am}$ die Mengen der Attributwerte sind.

Definition (Beschreibung): Eine *Beschreibung (Description) d* ist eine Zeichenkette (der Länge g) von attributierten Symbolen: $d : 1 \dots g \rightarrow W$.

Definition (Beschreibungssprache): Eine *Beschreibungssprache (Description Language) DL* ist eine Teilmenge der Menge D aller möglichen Beschreibungen $DL \subseteq D = \{d \mid \exists\ g \in \mathbb{N}\ (d : 1 \dots g \rightarrow W)\}$.

Die genaue Form der Beschreibung ist im weiteren Verlauf irrelevant - daß eine Beschreibung hier als Sequenz attributierter Symbole definiert wird ist daher unwesentlich. Ebenso könnten z.B.

Graphen (deren Knoten und Kanten best. Objekte und Relationen zwischen diesen darstellen) zur Beschreibung von z.B. Diagrammen verwendet werden – natürlich lassen sich viele Darstellungen äquivalent ineinander überführen, und man kann einen Graphen sicherlich durch eine attributierte Zeichenkette beschreiben. Insbesondere können jedoch auch grafische Sprachen wiederum grafische Sachverhalte beschreiben, wie dies z.B. in den Palmer-Welten $\mathcal{B}$ bis $\mathcal{H}$ der Fall ist (s. S. 13). Aber auch Beschreibungslogiken können verwendet werden (s. z.B. [WS92, BMPS$^+$91]).

Definition (Semantik): Die *Semantik* eines Bildes i ist eine Funktion $sem : \mathcal{P}(i) \rightarrow U$, wobei U das Referenzuniversum heißt. $\mathcal{P}(i)$ bezeichnet die Potenzmenge der Menge i (ein Bild i wurde oben als Funktion definiert; eine Funktion ist jedoch auch eine Menge von Tupeln).

Definition (Interpretation): Eine *Interpretation int* eines Bildes i ist eine Semantikfunktion $int = sem : \mathcal{P}(i) \rightarrow DL$. Hier ist also die Beschreibungssprache DL das Referenzuniversum U.

Die Ermittlung einer passenden Beschreibung $d \in DL$ eines Bildes $i \in PL$, so daß $int(i) = d$ wird, ist Aufgabe des *visuellen Parsings.* Die entsp. Komponente heißt *visueller Parser.* Parsingalgorithmen bzw. konstruktive Beschreibungen von *int* existieren für diverse Grammatikformalismen.

Statt von der Beschreibungssprache DL wird oftmals von *abstrakter Syntax* gesprochen. Eine Beschreibung d eines Bildes i wird ebenfalls abstrakte Syntax der konkreten Syntax des Bildes i genannt. In der Regel wird anhand der abstrakten Syntax in einem weiteren Interpretationsschritt die Semantik ermittelt: offenbar kann die *int*-Funktion die Sematik aber auch direkt – ohne den Zwischenschritt über die abstrakte Syntax – ermitteln. In diesem Sinne kann DL dann auch als „Bedeutungssprache“ verstanden werden, und d als Bedeutung. Auf welcher „Ebene“ also die Beschreibungssprache DL anzusiedeln ist (Semantik oder eher abstrakte Syntax), wird hier nicht festgelegt und ist prinzipiell nicht relevant, da die Grenzen unscharf sind.

Definition (Materialisierung): Eine *Materialisierung mat* einer Bildbeschreibung d ist eine Funktion $mat : \mathcal{P}(d) \rightarrow PL$.

Die Generierung eines passenden Bildes $i \in PL$, so daß $mat(d) = i$ wird, ist Aufgabe der *Visualisierung.* Die entsp. Komponente heißt *Visualisierungs-* oder *Präsentationskomponente.* Sie wird in der Regel anhand einer abstrakten syntaktischen Repräsentation eine Visualisierung erzeugen. Ebenso wie für die *int*-Funktion sind hier in der Praxis mehrere Zwischenschritte vorzusehen. Oft werden Programme zum Lösen von Einschränkungssystemen (Constraint Solver) verwendet.

Definition (Visueller Satz): Ein *visueller Satz (Visual Sentence)* ist ein Tripel $(i, d, (int, mat))$, wobei i ein Bild, d eine Beschreibung dieses Bildes, *int* eine Interpretationsfunktion und *mat* eine Materialisierungsfunktion ist. Ein visueller Satz heißt

- *korrekt bzw. wahrheitsgetreu (faithfull)* g.d.w. $i = mat(d)$ ist.
- *vollständig (full)* g.d.w. $d = int(i)$ ist.
- *korrekt und vollständig bzw. „komplett“ (complete)* g.d.w. er korrekt und vollständig ist.

Ein visuelle Satz entspricht in der textuellen Programmierung einfach einem „Programm“, ist also als Wort bzw. Satz einer formalen Sprache mit zugeordneter Semantik zu betrachten. Ein visueller Satz ist also *vollständig,* wenn die ihm zugeordnete Beschreibung (bzw. Semantik) *anhand des Bildes vollständig ermittelt werden kann.* Hierfür müssen natürlich best. Konventionen und Maßnahmen bzgl. der möglichen Bilder ergriffen werden (im Rahmen der visuellen Syntaxfestlegung), so daß eine vollständige Interpretation durch *int* möglich wird, m.a.W., es sollte sich um einen visuellen Formalismus handeln. Werden nun anhand der Beschreibung d in der „Welt“ DL neue Ergebnisse ermittelt (z.B. durch Berechnung, Inferenz o.ä.), so sollten diese wieder *korrekt bzw. wahrheitsgetreu* unter Verwendung der Funktion *mat* visualisiert bzw. materialisiert werden können. Dies ist für visuelle Programmiersprachen anzustreben.

Definition (Visuelle Sprache): Eine *visuelle Sprache* ist eine Menge visueller Sätze. Ein visuelle Sprache heißt

- *korrekt bzw. wahrheitsgetreu (faithfull)* g.d.w. alle ihre visuellen Sätze korrekt bzw. wahrheitsgetreu sind.
- *vollständig (full)* g.d.w. alle ihre visuellen Sätze vollständig sind.
- *komplett (complete)* g.d.w. alle ihre visuellen Sätze komplett sind.

Prinzipiell sollte z.B. für ein System zur Zeichnungsinterpretation (s. „ADIK“) Vollständigkeit bzgl. der betrachteten Zeichungsdomäne angestrebt werden – alle dargestellten Aspekte sollten erkannt werden.

Generell sollten alle visuellen Programmiersprachen komplett sein: der Benutzer erstellt ein visuelles Programm, welches unzweideutig und *vollständig* in eine interne Repräsentation überführt werden kann (durch visuelles Parsen). Alsdann wird das Programm ausgeführt, und die berechneten Werte werden wieder *wahrheitsgetreu* – wahrscheinlich in derselben, leicht veränderten initialen Darstellung – visualisiert. Ein Bsp. für eine *komplette* visuelle Sprache ist *„Pictorial Janus“* ([KS90, Kah90]).

Da der obige Formalismus einfach eine Abbildung zwischen zwei verschiedenen Welten (nämlich der Welt aller möglichen Teilbilder und der Welt der Beschreibungen dieser Teilbilder) beschreibt, kann auch der Palmersche Repräsentationsbegriff (s.o.) verwendet werden: für ein spezielles Bild i würde man $\mathcal{W}_1 = \mathcal{P}(i)$ und $\mathcal{W}_2 = DL$ sowie $\rho = int$ setzen. Die Funktion ρ bildet lediglich Objekte (nämlich jedes mögliche Teilbild von i) auf symbolische Strukturen der Beschreibungssprache DL ab (auch die leere Beschreibung $\emptyset$ kann in DL enthalten sein). Die Isomorphismus-Forderung ist für komplette visuelle Sprachen erfüllt.

Der „Formalismus“ von Bottoni et al. ist zwar gut zur Präzisierung vieler Begriffe geeignet, gibt jedoch keine Hinweise zur *konstruktiven Realisierung* der Funktion *int* und/oder *mat*, so daß sein praktischer Wert vergleichsweise gering ist. Hierin ist aber auch die Allgemeinheit des Verfahrens zu sehen. Während es in der Literatur verschiedenste Formalismen gibt, die z.B. nur für ikonische Sprachen geeignet sind, ist dies hier nicht der Fall, da die zugrundeliegenden visuellen Entitäten auf einer „subsymbolischen“ Ebene (als Pixelhaufen) betrachtet werden. Konstruktive Beschreibungen von *int* setzen in der Regel jedoch auf semantisch höheren Ebenen auf, in denen bereits *symbolische Objekte* vorliegen; m.a.W., der *int*-Funktion wird hier „zuviel“ aufgebürdet.

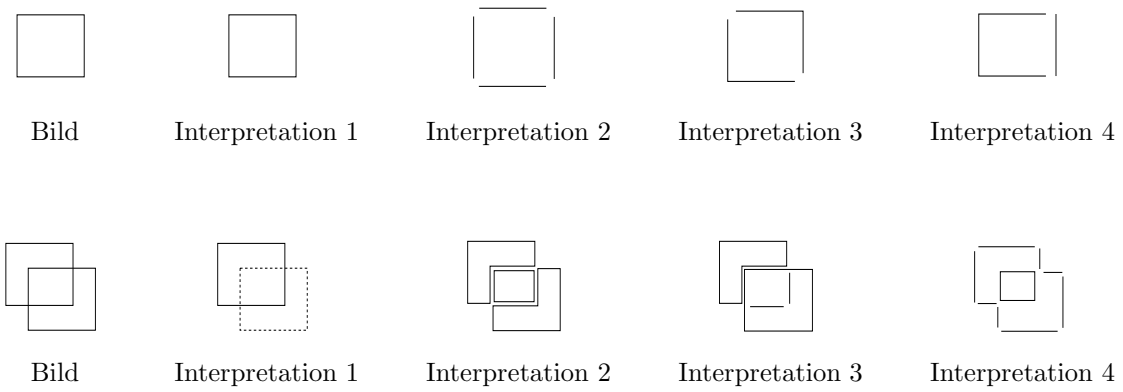


Abbildung 2.17: Mehrdeutigkeiten

Soll die Bildinterpretation tatsächlich auf der konkreten Syntax des Bildes i beginnen, so wäre hier – ähnlich einer Schriftzeichenerkennung (Optical Pattern Recognition, OCR) – zunächst eine *Segmentierung* des Bildes in einzelne Objekte vorzunehmen. Derartige Algorithmen werden in der Bildverarbeitung untersucht. Obwohl die Anzahl der möglichen Bilder im Bereich visueller Sprachen in der Regel durch die Syntax stark eingeschränkt wird (im Gegensatz zu moderner

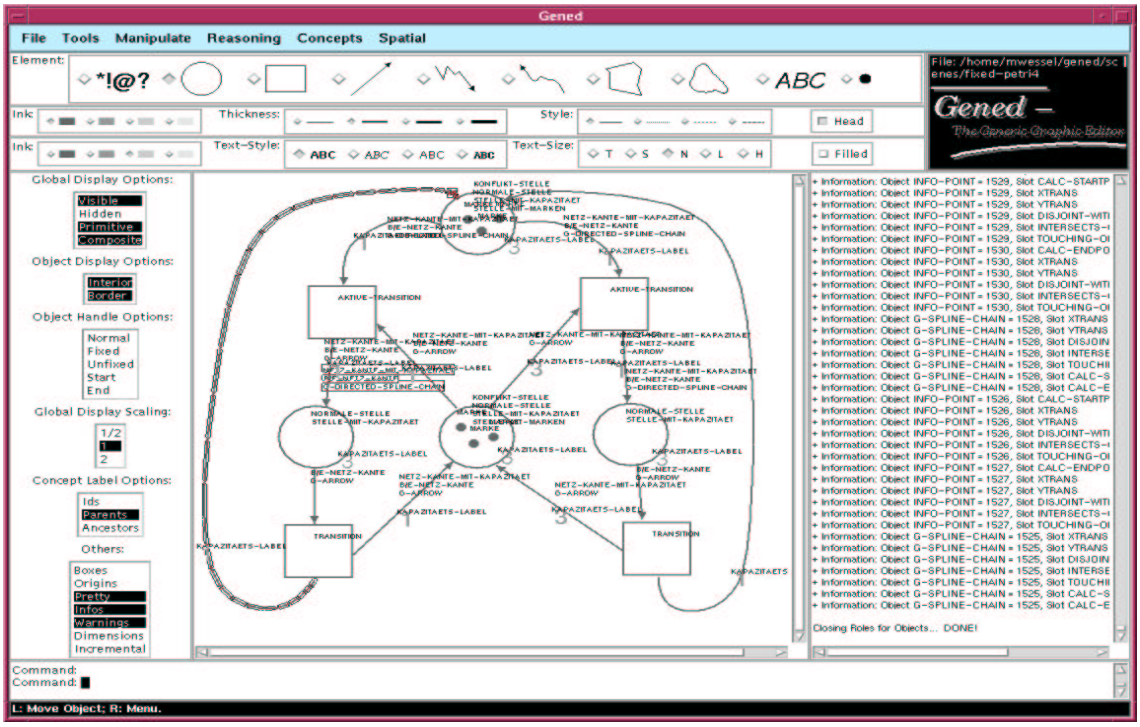


Abbildung 2.18: Der wissensbasierte Grafikeditor GENED

Kunst), so muß man doch mit potentiell vielen Mehrdeutigkeiten rechnen, die eine maschinelle Interpretation beginnend auf dieser Ebene sehr schwer machen.

Dies sei anhand von Abb. 2.17 verdeutlicht: ein visueller Parser soll nun anhand der gegebenen Bilder die abstrakte Syntax bzw. „Bedeutung“ erzeugen (z.B. in Form eines abstrakten Syntaxgraphen), wobei er sowohl Strecken, Streckenketten als auch Polygone erkennen soll. Offensichtlich sind hier nur einige wenige möglichen Interpretationen der dargestellten Bilder angegeben (die erkannten individuellen Objekte sind in den Darstellungen etwas auseinandergerückt, um die vorgenommene Segmentierung sichtbar zu machen).

Konventionen spielen hier eine wesentliche Rolle, da sie die *Anzahl möglicher Interpretationen stark einschränken.* Solche Konventionen werden zum einen von der Syntax, zum anderen aber auch von den *Gestaltprinzipien* – wie dem *Prinzip der guten Forsetzbarkeit* – vorgegeben. Wichtig ist auch der *Kontext* in dem das entsp. Bild auftaucht. Diese Punkte wurden ja bereits bei der Diskussion der Ikonen angesprochen.

Die meisten visuellen Sprachen vermeiden die Segmentierungsproblematik, indem sie ein spezielles erstellendes Werkzeug (wie z.B. einen syntaxgesteuerten Grafikeditor) verwenden. Große Teile der abstrakten Syntax können dann intern direkt anhand der Interaktionssequenzen des Benutzers aufgebaut werden, und der Schritt der Ermittlung der abstrakten Syntax anhand der konkreten Syntax entfällt somit weitgehend – natürlich können intern aber noch weitere Schritte anfallen, wie z.B. das Explizieren impliziter räumliche Relationen durch Berechnung.

Von Haarslev und Wessel wurde ein wissensbasierter Grafikeditor namens GENED vorgestellt ([Wes96, HW96, Haa96]), s. Abb. 2.18. Die vom Benutzer erzeugten Objekte und Konstellationen können hier durch eine *benutzeradaptierbare, beschreibungslogisch definierte Interpretationsfunktion* interpretiert bzw. klassifiziert werden. Die Interpretation geschieht durch A-Box-Klassifikation, wobei es sich um einen von beschreibungslogischen Systemen angebotenen Basisdienst handelt. In einer sog. T-Box (hier wird *terminologisches Wissen* kodiert) können domänenspezifische räumliche Konzepte anhand von notwendigen und evtl. auch hinreichenden Bedingungen definiert wer-

den. Wie auch „ADIK“ ist GENED generisch, da es für versch. Domänen durch einfaches Austauschen der Wissensbasis verwendet werden kann. Untersucht wurden Domänen wie Petrinetze und ER-Diagramme. Die Idee, Beschreibungslogiken zur konstruktiven Definition der *int*-Funktion zu verwenden, stammt von Haarslev (die erste Veröffentlichung diesbzgl. ist [HMS94]). In [Haa95] wird gezeigt, wie große Teile der Syntax der Sprache „Pictorial Janus“ mit Hilfe einer Beschreibungslogik definiert werden können.

Während GENED ein Freiformeditor ist (jedes mit den angebotenen Basisobjekten erstellbare Bild kann auch erstellt werden), findet man in der Literatur auch viele Vorschläge für *syntaxgesteuerte Grafikeditoren*. Ein syntaxgesteuerter Grafikeditor schränkt die mögliche Anzahl erstellbarer Bilder so ein, daß bzgl. der aktuellen Domäne stets nur syntaktisch korrekte Bilder entstehen. Oftmals werden *automatisches Layout* und *domänenspezifische Interaktionsformen* angeboten: so sollte z.B. für einen Petrinetz-Editor eine komfortable „Verbinde“-Funktionalität vorgesehen sein (der Benutzer selektiert die beiden zu verbindenden Objekte – genau eine Stelle und eine Transition – und das System erzeugt eine Kante zwischen diesen, deren genauer Verlauf automatisch bestimmt wird; gleichzeitig wird so sichergestellt, daß das Petrinetz auch tatächlich bipartit, also syntaktisch korrekt ist).

Oftmals werden spezialisierte Grafikeditoren (die z.B. als Konstruktionswerkzeuge für spezielle visuelle Programmiersprachen vorgesehen werden) anhand einer formalen Grammatikbeschreibung des entsp. visuellen Formalismus von einem *Synthesizer* generiert, also speziell erzeugt (vergleichbar einem Compiler-Compiler, der einen speziellen Compiler erzeugt, s. z.B. [CM95, RS96, MV95, Üsk94]).

Oben wurde bereits deutlich, daß die Funktion *int* (die also die Semantik eines Bildes i ermittelt) pragmatisch in mehrere Teilfunktionen aufgeteilt werden sollte: $int = int_n \circ int_{n-1} \circ \ldots \circ int_1$. Rekers und Schürr ([RS96, RS95]) verwenden z.B. $n = 3$:

1. Ein konventioneller Grafikeditor (z.B. MacDraw) erzeugt eine Sammlung bildhafter Elemente. Hierbei handelt es sich um die unterste betrachtete Repräsentationsebene (im Gegensatz zum Formalismus von Bottoni et al.), die auch Ebene des physikalischen Layouts genannt wird: lediglich Objekte und Attribute liegen explizit vor, räumliche Relationen sind implizit bzw. intrinsisch repräsentiert.

2. int_1 wird nun durch *grafisches Abtasten (Graphical Scanning)* den sogenannten „Spatial Relations Graph“ erzeugen. Die Kanten des Graphen sind räumliche Relationen wie „enthalten“, „berührt“ etc., so daß also lediglich die im physikalischen Layout implizit vorhandenen räumlichen Relationen explizit gemacht werden müssen. Die attributierten Knoten des Graphen entsprechen weitgehend den relevanten Objekten der Ebene des physikalischen Layouts: desweiteren können jedoch sog. *auftauchende Objekte (emergent objects)* – dies sind Objekte, die bisher lediglich implizit zu erkennen, aber nicht explizit als Objekte vorhanden waren – explizit gemacht worden sein. Sollen auftauchende Objekte automatisch erkannt und expliziert werden, so müssen ähnliche Probleme wie die in Abb. 2.17 dargestellten Mehrdeutigkeitsprobleme gelöst werden (s. z.B. das auftauchende Schnittrechteck der beiden sich schneidenden Rechtecke).

3. int_2 erzeugt durch „Low Level Parsing“ eine Repräsentation, die „Abstract Relations Graph“ genannt wird. Sowohl die Knoten als auch die Kanten dieses Graphen sind ihrer Semantik entsprechend weitgehend klassifiziert. Für die Petrinetz-Domäne würden Kreise zu Stellen, Rechtecke zu Transitionen und räumliche Relationen bzw. Kanten im „Spatial Relations Graph“ wie „touches(arrow,circle)“ und „touches(arrow,rectangle)“ gegen eine „linked-with(place,transition)“-Kante ersetzt.

4. int_3 führt ein sog. „High Level Parsing“ durch. Hier werden einige weitere Konsistenzbedingungen überprüft, die in Phase int_2 noch nicht geprüft werden konnten. Für ein Petrinetz muß z.B. geprüft werden, daß der Graph auch bipartit ist, etc.

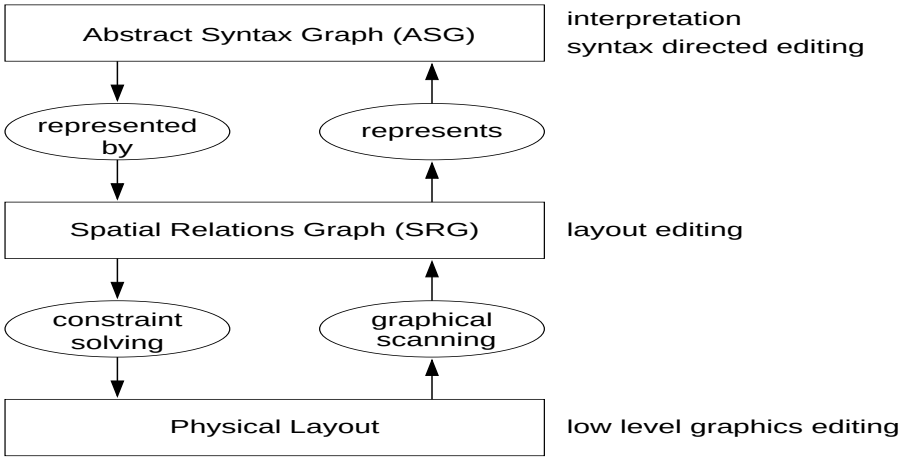


Abbildung 2.19: Visuelles Parsen nach Rekers und Schürr (reproduziert nach [RS96])

In der Regel bezeichnet man nun irgendeine interne Repräsentation wie den „Spatial Relations Graph“ oder den „Abstract Relations Graph“ als *abstrakte Syntax.* Der „Abstract Relations Graph“ ist jedoch schon weitgehend semantisch interpretiert. Letzlich sind die Übergänge zwischen „abstrakter Syntax“ und „Semantik“ also fließend. In Abb. 2.19 sind int_2 und int_3 zusammengefaßt, so daß nur drei Ebenen erscheinen.

Die Anzahl der vorzusehenden Zwischenstufen hängt u.a. von der Mächtigkeit der verwendeten Formalismen ab – z.B. wird die bei konventionellen textuellen Programmiersprachen vorgenommene Unterscheidung zwischen Syntax und statischer Semantik im wesentlichen aufgrund der Beschränkungen kontextfreier Grammatiken notwendig ([Mey90]).

In GENED wurden für *int* zwei Schritte notwendig:

1. Explizieren der räumlichen Relationen zwischen den Grafikobjekten. Hier wurden die acht qualitativen räumlichen Egenhofer-Relationen verwendet (die verwendeten Relationen tragen zwar die selben Namen wie die Egenhofer-Relationen, wurden jedoch anders definiert, s. [Wes96]).

2. Erzeugen von zu den Grafikobjekten korrespondierenden Individuen in der sog. „CLASSIC“-Beschreibungslogik und zusichern der zuvor berechneten Relationen. Die alsdann von „CLASSIC“ vorgenommenen Klassifikationen wurden einfach als entsp. Beschriftungen an den Grafikobjekten materialisiert.

Visualisierungs- und Interaktionsprobleme

Während bisher im wesentlichen *Interpretationsprobleme* betrachtet wurden, so tauchten im Rahmen dieser Arbeit nicht zuletzt auch zahlreiche *Visualisierungsprobleme* auf.

Da in dieser Arbeit u.a. Polygone und Strecken eine Rolle spielen, stellt sich die Frage, wie man z.B. ein *explizites Dreieck* (ein spezielles Polygon) von einem *impliziten Dreieck* unterscheiden kann (in Abb. 2.20 sind die Strecken des impliziten Dreiecks etwas auseinandergerückt dargestellt, damit ihre Individualität betont wird – letzlich wird hierdurch jedoch die Geometrie verfälscht, so daß dieses Visualisierungsverfahren nicht geeignet ist, denn bei der dargestellten Visualisierung des impliziten Dreieckes handelt es sich nicht mehr um eine Dreieck!).

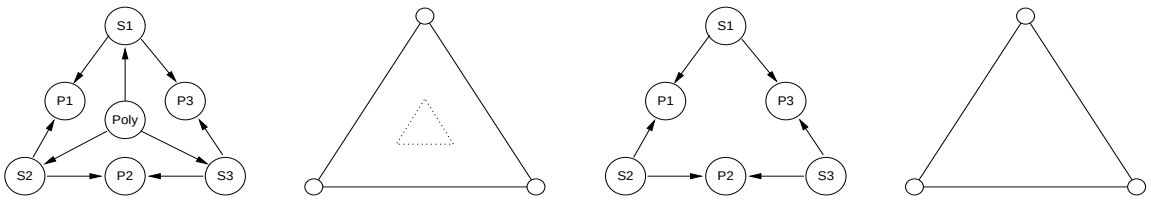
Ein Dreieck ist an sich stets ein *implizites Gebilde,* denn es wird vollständig durch seine drei Strecken definiert: es gibt kein explizites visuelles Anzeichen für das Dreieck *selbst,* sondern lediglich für die drei *Dreiecksseiten.*

(a) Explizites Dreieck (b) Impliztes Dreieck

Abbildung 2.20: Implizite und explizite Dreiecke

Implementiert man jedoch einen objektorientierten Grafikeditor, so stellt sich die Frage, wie man das auftauchende Objekt „implizites Dreieck“, welches durch drei günstig zueinander stehenden Strecken gebildet wird, von einem expliziten Dreieck (welches eine spez. Polygon ist) *unterscheiden* kann. In nahezu allen Grafikeditoren gibt es keine Unterscheidungsmöglichkeit: dies ist verständlich, da ja – sofern es um die Grafik selbst bzw. „What you see is what you get (WYSIWG)“ geht – ein Dreieck ein implizites Objekt ist. Oftmals will man aber das (implizite) auftauchende Objekt Dreieck zu einem expliziten Dreieck (also einem spez. Polygon) machen, um entsp. Operationen hierauf anwenden zu können. Auch die Bedeutung eines expliziten Dreieckes könnte sich von einem impliziten Dreieck unterscheiden (dies ist in der Sprache VISCO der Fall) – es muß also eine Möglichkeit vorgesehen werden, implizite und explizite Objekte zu unterscheiden und auch implizite Objekte explizit machen zu können. Während implizite und explizite Dreiecke in objektorientierten Grafikeditoren in der Regel ununterscheidbar sind, können jedoch meist Objektkonstellationen expliziert werden, indem Gruppenobjekte (Aggregate) gebildet werden (so im Grafikeditor „XFig“). Ein Aggregat ist jedoch kein Polygon (sondern ein allgemeineres Objekt) und kann deshalb auch nicht als solches behandelt werden: u.a. kann man ein Aggregat nicht mit Farbe füllen – diese Operation ist nur auf expliziten Polygonen sinnvoll anwendbar.

Selbst wenn intern anhand der abstrakten Syntax ein explizites Dreieck von einem impliziten Dreieck unterschieden werden kann, so sollte diese Unterscheidbarkeit doch auch dem Benutzer über die Funktion *mat* (die Visualisierungs- oder Materialisierungsfunktion) ermöglicht werden. Dies ist vor allem dann nötig, wenn der Benutzer sowohl auf das ganze Dreieck als auch auf seine einzelnen Seiten Bezug nehmen können muß. Denkt man an direkte Objektberührungen mit einem Mauszeiger, so wird klar, daß ohne einen Stellvertreter für das gesamte Dreieck entweder immer nur einzelne Seiten oder das gesamte Dreieck referenziert werden können, nicht jedoch abwechselnd mal das eine oder das andere. Daher sind kompliziertere Referenzierungsmechanismen notwendig.



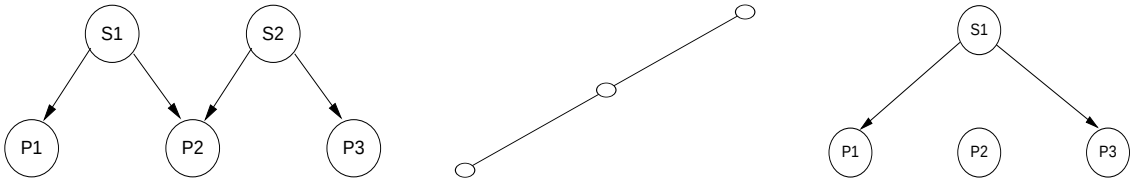
(a) ASG 1 (b) Visualisierung 1 (c) ASG 2 (d) Visualisierung 2

Abbildung 2.21: Unterscheidbare Visualisierungen

Zu Verdeutlichung zeigt Abb. 2.21 zwei abstrakte Syntaxgraphen und ihre zugehörigen Visualisierungen (die Kanten der Graphen repräsentieren die „Komponente von“-Relation). Damit die Visualisierungen unterscheidbar werden, wurde in der Visualisierung (Abb. 2.21(b)) von ASG 1 (Abb. 2.21(a)) ein *ikonischer Repräsentant als explizites visuelles Anzeichen für den Polygonknoten*

bzw. das Dreieck selbst eingezeichnet.[6] Nun sind implizite und explizite Dreiecke unterscheidbar. Auch das oben beschriebene Referenzierungsproblem könnte so gelöst werden: sowohl auf einzelne Seiten als auch auf das gesamte Dreieck kann nun Bezug genommen werden – eine Berührung einer Seite meint die Seite selbst, eine Berührung des Ikones jedoch das gesamte Dreieck.

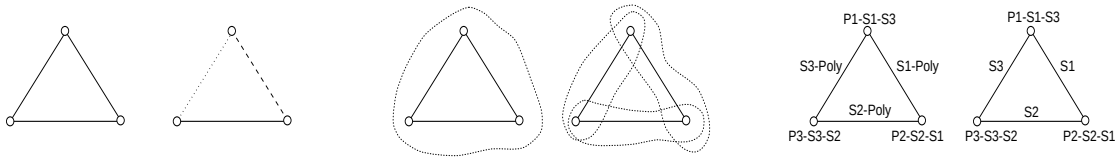
Daß es sinnvoll ist, auch visuelle Anzeichen für die Endpunktknoten von Strecken einzuführen, wird dann deutlich, wenn man an zwei Strecken denkt, die aneinander anschließen und die gleiche Steigung haben. Letzlich sind dann aber immer noch Mehrdeutigkeiten möglich: so haben die beiden ASGs in Abb. 2.22 die gleichen Visualisierungen. Verbietet man nun, daß Punkte auf Linien zum Liegen kommen, ohne auch Endpunkt zu sein – dies ist bei sog. *topologisch strukturierten Vektordaten (s. Kap. 3) der Fall* –, so kann die Visualisierung in Abb. 2.22(b) nur noch Materialisierung von ASG 1 sein (2.22(a)).



(a) ASG 1 (b) Visualisierung ASG 1/2 (c) ASG 2

Abbildung 2.22: Ununterscheidbare Visualisierungen

Andere Möglichkeiten der Disambiguierung („Handelt es sich um drei Segmente, die implizit ein Dreieck bilden oder handelt es sich um ein explizites Dreieck?“) der Darstellung wären:



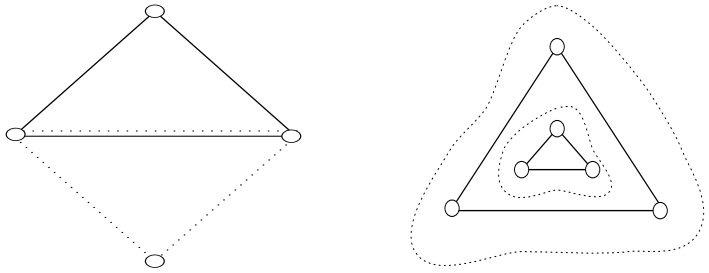
(a) Färbungen (b) Einkreisungen (Blobs) (c) Beschriftungen

Abbildung 2.23: Unterscheidbare explizite und implizite Dreiecke

- **Farben** (s. Abb. 2.23(a)): durch die Konvention, daß zwei topologisch zusammenhängende (verbundene) Objekte (wie z.B. zwei aneinanderstoßende Segmente eines Polygones) genau dann die gleiche Farbe haben, wenn sie Teil ein und desselben Elternkomplexobjektes sind (anderseits haben sie verschiedene Farben), kann die Eindeutigkeit bei der Interpretation wieder hergestellt werden. Dann kann die visuelle Variable Farbe jedoch nicht mehr zur direkten Repräsentation des Attributes Farbe genutzt werden. Dies ist in Grafikeditoren natürlich in der Regel unakzeptabel. Ein Problem ergibt sich zudem durch gemeinsam genutzte Knoten im ASG: ein Segment, daß z.B. Teil zweier Polygone ist, muß nach dieser Definition *zwei* Farben haben. Objekte müßten somit in mehreren Farben (z.B. leicht gegeneinander versetzt) dargestellt werden (s. Abb. 2.24(a)). Bei hinreichend komplizierten Konstellationen geht die Übersichtlichkeit sofort verloren. Das gleiche Problem entsteht auch für die Endpunkte – verbietet man auf Linien liegende Punkte, so ist die „Komponte von“-Relation jedoch auch für Punkte wieder eindeutig ablesbar, auch ohne Farbgebung. Zur Erzeugung der Farbgebungen muß zudem von einer Präsentationskomponente anhand des ASG ein aufwendiges Graphenfärbungsproblem (Graph Coloring Problem) gelöst werden (welches in der allgemeinen Form NP-vollständig ist).

[6] Dank an Ralf Möller für diese Idee!

- **Einkreisungen** („Blobs“, s. Abb 2.23(b)): alle Komponentenobjekte eines Objektes werden in einer geschlossenen Kurve eingeschlossen. Auch hier gibt es das Problem, daß die Übersichtlichkeit schnell verloren geht. Zusätzlich gibt es jedoch das Problem, daß aufgrund der Transitivität der „Enthalten in“-Beziehung in Bild 2.24(b) impliziert wird, daß das enthaltene Polygon Komponentenobjekt des anderen Polygones sei, was nicht der Fall sein soll. Somit ließe sich die Bildung der „Komponente von“-Beziehung nicht unabhängig von der räumlichen „Enthalten in“-Beziehung durchführen, da beide über die visuelle Repräsentation gekoppelt würden.
- **Hinterlegungen in verschiedenen Farben:** dieses Vorgehen ist vergleichbar mit Punkt 1, auch hier geht die Übersichtlichkeit schnell verloren.
- **Beschriftungen** (s. Abb. 2.23(c)): jedes Objekt wird mit einem Identifizierer versehen. Komponentenobjekte werden sowohl mit ihrem eigenen Identifizierer als auch mit den Identifizierern der direkten Elternobjekte beschriftet. Eventuell sollte das Objekt mit allen Vorgängern und nicht nur mit den direkten Vorgängern beschriftet werden.



(a) Probleme mit Farben (b) Probleme mit Einkreisungen

Abbildung 2.24: Weitere Probleme

Es wird deutlich, daß es keine wirklich befriedigende statische Lösung dieser Visualisierungsprobleme gibt: die entsp. Konventionen bzw. zusätzlichen metagrafischen Annotationen verkomplizieren die Darstellungen erheblich und kompensieren somit den Gewinn an Eindeutigkeit wieder. Generell sind solche oder ähnliche Probleme jedoch beim Entwurf kompletter visueller Programmiersprachen zu lösen. In der Regel werden kongruente oder sich verdeckende Objekte ausgeschlossen. Objektorientierte Grafikeditoren ignorieren die Problematik – die von ihnen generierten Darstellungen sind nicht *wahrheitsgetreu* und müssen es natürlich auch nicht sein, da den Grafiken keine maschinelle Bedeutung zukommt.

Unter Verwendung einer unterstützenden Oberfläche kann man sich jedoch eine *dynamische Inspektionsmethode* zur Lösung vieler dieser statisch nicht-lösbaren Visualisierungsprobleme vorstellen: hierzu nähert der Benutzer den Mauszeiger versuchsweise einzelnen Objekten. Das jeweils nächste Objekt wird dann hervorgehoben, und zudem werden alle Komponentenobjekte hervorgehoben. Durch *verschieden genaues Zeigen* kann der Benuzter nun auch auf einzelne Komponenten Bezug nehmen, indem er den Mauszeiger spezifischen Komponenten weiter nähert (s. Abb. 2.25).

Nähert sich der Mauszeiger einem Dreieck, so wird zunächst das gesamte Dreieck hervorgehoben. Je näher der Mauszeiger einzelnen Komponenten kommt, desto spezifischer ist die Referenz – im ASG wird somit „Top-Down“ hinabgestiegen. Somit können *dynamisch* auch implizte und explizite Dreiecke unterschieden werden, denn bei Annäherung an ein implizites Dreieck können (im Gegensatz zu einem expliziten Dreieck) nur einzelne Segmente, nicht jedoch das gesamte Gebilde hervorgehoben werden. Wie erwähnt, sollte das implizite Dreieck jedoch vom Benutzer *expliziert* werden können: in Bild 2.26(a) erzeugte der Benutzer zunächst ein explizites Dreieck, dann ein Weiteres in Abb. 2.26(b) (das eingeschriebene Dreieck). Von den fünf nun ersichtlichen Dreiecken liegen ausschließlich zwei explizit vor. Schließlich hat sich der Benutzer in Abb. 2.26(c)

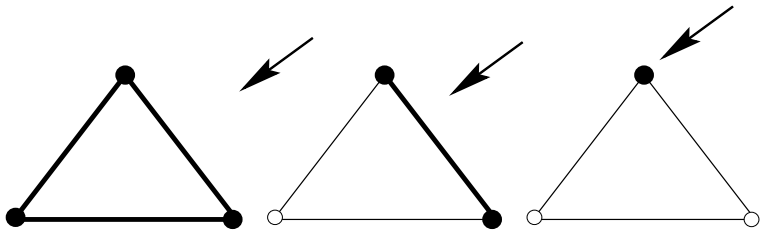
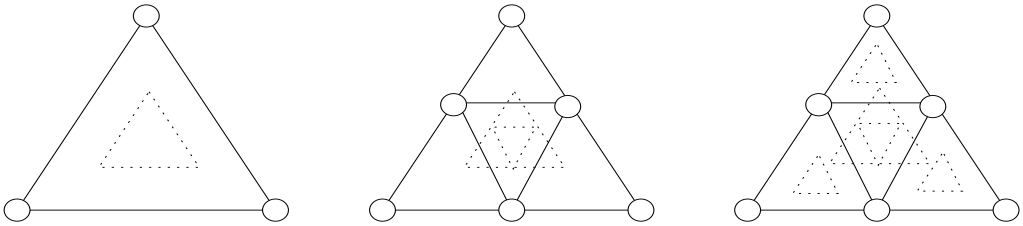


Abbildung 2.25: Disambiguierung durch dynamische Inspektion

entschieden, all diese explizit zu machen. Das nicht automatisch jedes implizite Polygon explizit gemacht werden sollte, wird bei einem Blick auf Abb. 2.27 deutlich.



(a) Ein explizites Dreieck

(b) Zwei explizite Dreiecke, fünf implizite

(c) Fünf explizite Dreiecke

Abbildung 2.26: Explizierung auftauchender Polygone

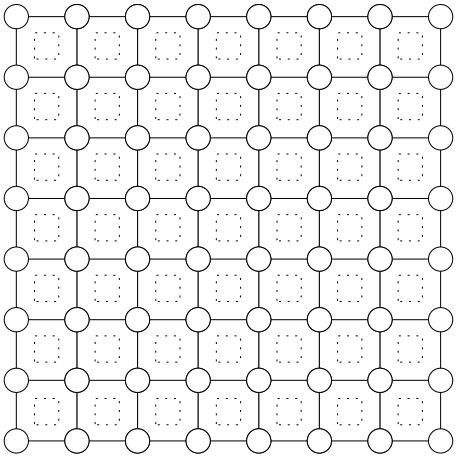


Abbildung 2.27: 49 explizite und sehr viele weitere implizite Polygone

Kapitel 3

Geoinformatik und Geographische Informationssysteme (GIS)

Als interessante Anwendungsdisziplin, in der räumliche Konzepte eine tragende Rolle spielen, soll die Geoinformatik etwas näher beleuchtet werden. Dabei möchte ich mich auf einige wenige Punkte beschränken, die sich im Verlauf der Arbeit als relevant herausgestellt haben – hierzu gehören vor allem die entwickelten *Verfahren zur Repräsentation und Organisation räumlicher Daten.* Schließlich möchte ich als konkretes Beispiel die Struktur der DISK-Daten des Hamburger Vermessungsamtes darstellen.

Ein hervorragendes Buch zum Thema Geoinformatik wurde von N. Bartelme geschrieben ([Bar95]): vieles des hier Dargestellten ist ihm entnommen.

3.1 Definitionen

Ein Informationssystem ist lt. Duden Informatik ([Dud93])

> Ein System zur Speicherung, Wiedergewinnung, Verknüpfung und Auswertung von Informationen. Ein Informationssystem besteht aus einer Datenverarbeitungsanlage, einem Datenbanksystem und den Auswertungsprogrammen.

Unter einem Datenbanksystem wird hier eine Datenbank verstanden: als Datenbank wird i.d.R. ein Datenbestand und ein diese Daten verwaltendes Datenbankverwaltungssystem (DBMS, Database Management System) verstanden ([Dat95]). Aus diesem Datenbestand kann nun „Information“ durch einen Benutzer unter Verwendung der Wiedergewinnungs-, Verknüpfungs- und Auswertungsprogramme extrahiert werden. Letzlich entsteht die Information jedoch im Kopf des Benutzers, indem er die von den Programmen aufbereiteten Daten interpretiert bzw. die Information mental rekonstruiert, wozu er sein Wissen verwendet. Die Auswertungsprogramme sollen letztlich dabei helfen, in den Daten verborgene Zusammenhänge zu extrahieren und somit Grundlagen bzw. Unterstützung für Entscheidungen bieten (Decision Support). Strenggenommen sind es also nicht „Informationen“, die in einem Informationssystem gespeichert, wiedergewonnen, verknüpft und ausgewertet werden, sondern Daten. Die im folgenden betrachteten Informationssysteme werden auch als Non-Standardanwendungen bezeichnet, da sie im Gegensatz zu z.B. den Personalinformationssystemen (PIS) nicht ausschließlich alphanumerische Daten enthalten.

Für den Begriff des *Geographischen Informationssystemes (Geographic Information System, kurz GIS)*[1] gibt es viele Definitionsversuche, einige wenige sind:

> Ein Geoinformationssystem dient der Erfassung, Speicherung, Analyse und Darstellung aller Daten, die einen Teil der Erdoberfläche und die darauf befindlichen technischen und adminstrativen Einrichtungen sowie geowissenschaftliche, ökonomische und ökologische Gegebenheiten beschreiben ([Bar95]).

> A geographic information system (GIS) is an information system that is designed to work with data referenced by spatial or geographic coordinates. In other words, a GIS is both a database system with specific capabilities for spatially-referenced data, as well a set of operations for working with the data ([SE90]).

> Ein Geo-Informationssystem ist ein rechnergestütztes System, das aus Hardware, Software, Daten und den Anwendungen besteht. Mit ihm können raumbezogene Daten digital erfaßt und redigiert, gespeichert und reorganisiert, modelliert und analysiert sowie alphanumerisch und grafisch präsentiert werden ([BF91]).

Wenn auch die unterschiedlichen Definitionen jeweils verschiedene Schwerpunkte setzen, so erwähnen sie doch alle (implizit oder explizit) den *Raumbezug* der relevanten Daten: so ist die Rede von „Daten, die einen Teil der Erdoberfläche ... beschreiben“, „spatially-referenced data“ und „raumbezogenenen Daten“. Während die zweite und dritte Definition den Rechnereinsatz betonen, könnte das GIS in der ersten Definition auch ein umfangreiches (analoges) z.B. von einer Behörde (mit unterschiedlichen Abteilungen für die einzelnen Aufgaben) verwaltetes Kartenwerk sein. So ist z.B. die Aufgabe des Hamburger Vermessungsamtes „die Bereitstellung von geographischen Basisdaten mit einheitlichem Raumbezug, im Klartext: Karten und Pläne ... Waren dies bisher ausschließlich analoge Produkte ... so sind es heute in zunehmenden Maße digitale Datenbestände in unterschiedlichen Formen.“ ([Fre96a]). Tatsächlich findet man in der GIS-Literatur häufig den Hinweis, daß es sich bei analogen Papierkarten um „Informationssyteme“ handele. Während diese Sichtweise aus der Informatikperspektive etwas ungewöhnlich erscheint, so ist sie jedoch nachvollziehbar, wenn man sich bewußt macht, daß Informationen nicht mit oder von Rechnern, sondern Menschen verarbeitet werden. So schreibt das Hamburger Vermessungsamt in [Fre87] über die seit 150 Jahren fortgeführte Flurkarte 1 : 1000:

> ... dient das Kartenwerk einer Vielzahl von Anwendern als Grundlage ihrer Fachaufgaben wie z.B.
>
> - Nachweis der Rechtsverhältnisse an Grundstücken in Verbindung mit dem Grundbuch,
> - Finanzierung und Genehmigung von Bauvorhaben,
> - Ableitung von Karten kleinerer Maßstäbe,
> - Entwurf und Planung von Hoch- und Tiefbaumaßnahmen,
> - Grün- und Landschaftsplanung,
> - Raumordnung und Bauleitplanung,
> - Bodenordnung und Stadterneuerung,
> - Wertermittlung von Grundstücken,
> - Verwaltung der Liegenschaften der Freien und Hansestadt Hamburg,
> - Dokumentation und Planung von Leitungen (Elektrizität, Gas, Wasser, Siel u.a)
> - Erhebung öffentlicher Abgaben (Steuern, Erschließungsbeträge).

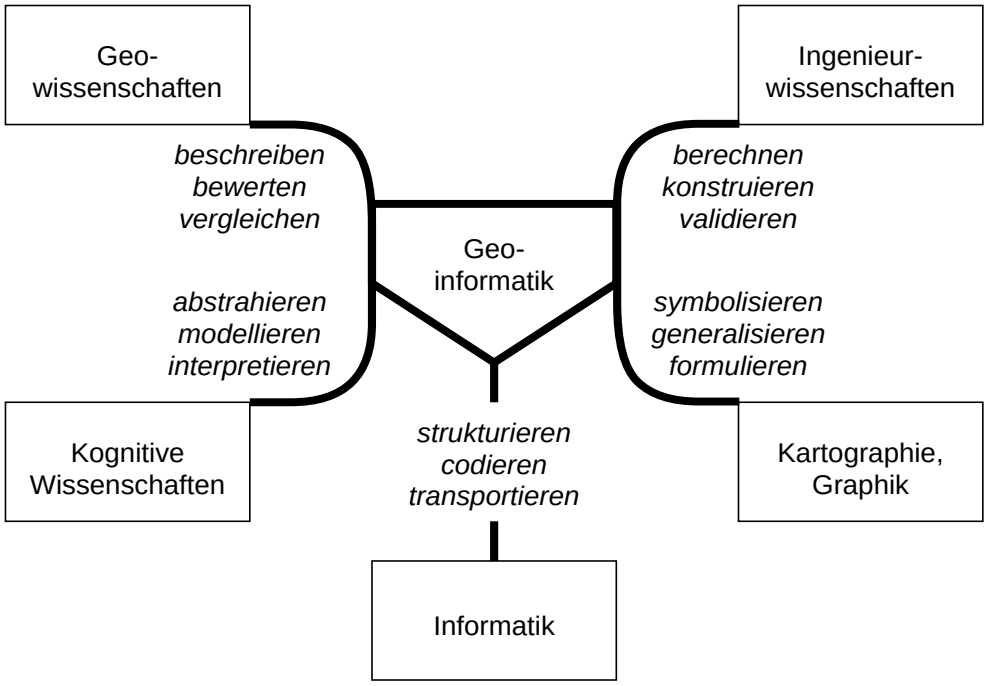


Abbildung 3.1: Geoinformatik im wissenschaftlichen Umfeld (reproduziert nach [Bar95])

Alle diese Aufgaben sollen in Zukunft mit Hilfe der „Digitalen Stadtgrundkarte (DSGK)“ gemeistert werden:

> Die „Digitale Stadtgrundkarte“ (DSGK) ist ein System, mit dem die raumbezogenen Sachverhalte und Erscheinungen umfassend in digitaler Form erfaßt, verarbeitet und grafisch präsentiert werden.

Offensichtlich soll hier ein spezielles GIS geschaffen werden. Die in obiger Liste aufgeführten Anwendungsbeispiele der Flurkarte 1 : 1000 sind somit potentielle GIS-Anwendungen in diesem Kontext. Typische Anwendungsbereiche von GIS sind u.a.

- Vermessungswesen
- Umwelt (z.B. Umwelt-Monitoring mit Mitteln der Fernerkundung)
- Städteplanung (z.B. Infrastrukturplanung, Bebauungsplanung, Grundstücksverwaltung)
- Planung und Betrieb von Netzen (z.B. Routenplanung, Verkehrssimulation, etc.)

Die große Heterogenität und Breite von GIS-Anwendungen läßt es sinnvoll erscheinen, eine Art „GIS-Grundlagenwissenschaft“ zu etablieren: Die sog. *Geoinformatik* ist eine sehr junge Wissenschaft und ein interdisziplinäres Unternehmen, welches eine große Anzahl von Teilbereichen unterschiedlichster Wissenschaftsdisziplinen zu vereinen sucht – hier finden sich Anteile aus der Informatik, den Geowissenschaften, aus der Kartographie, den Ingenieurwissenschaften, etc. (s. Abb. 3.1).

Bartleme beschreibt das Forschungsprogram der Geoinformatik folgendermaßen:

[1] Obwohl im Deutschen die Kurzform Geo-Informationssystem gebräuchlich ist, werde ich das Akronym „GIS“ verwenden.

> ... Abläufe im menschlichen Denkprozeß *[bezogen auf räumliche Daten]* besser verstehen zu lernen und Modelle für die Entscheidungsfindung in Situationen mit raumbezogenem Charakter zu finden, das Speichern, Wiedererkennen und Weitervermitteln solcher Situationen in gewissem Umfang nachvollziehen zu können, das ist die eigentliche Herausforderung an die Geoinformatik, die damit den Rang einer vollwertigen und eigenständigen Wissenschaftsdisziplin belegt. *(Hinzufügungen in Kursivschrift, d.V.)*

Während „Geo“ und „Informatik“ in erster Linie die Anteile von Geowissenschaften und Informatik hervorheben, klingt in obiger Beschreibung des Forschungsprogrammes „Geoinformatik“ eine Methodik an, die Assoziationen zur Kognitionsforschung weckt. Sicherlich ist die Berücksichtigung kognitiver Aspekte sehr wesentlich, wenn die entstehenden Systeme effektiv von Menschen benutzbar sein sollen. Fragen betreffend Design und Implementation von Geographischen Informationssystemen – oder allgemeiner: *räumlicher Informationssysteme* – werden auch in der *räumlichen Informationstheorie (Spatial Information Theory)* untersucht. Hier ist die „COSIT“-Konferenz (Conference on Spatial Information Theory) zu erwähnen, die von A. Frank ins Leben gerufen wurde, um eine solide theoretische Basis für die GIS-Technologie aufzubauen.

Historisch spielt die *Kartographie* (die Wissenschaft von der Erstellung von Karten) für die Geoinformatik eine herausragende Rolle: so sind die von dieser Wissenschaft in jahrhunderterlanger Tradition entwickelten Techniken zur Visualisierung komplexer raumbezogener Sachverhalte mit Hilfe des analogen Informationssytems „Karte“ unbedingt auch für die digitalen Nachfolger „GIS“ nutzbar zu machen. Durch den Rechnereinsatz ergeben sich für die digitalen Karten „GIS“ jedoch Möglichkeiten, die analoge Karten nicht bieten: u.a. können die Daten beliebig kombiniert, aktualisiert, präsentiert und analysiert werden und stellen somit die Möglichkeiten konventioneller Karten weit in den Schatten. Das *Überlagern* (transparenter) Karten (auch *Verschneiden* genannt) war (und ist) eine der wichtigsten Analysetechniken. Sucht man z.B. alle Gewässer in Deutschland, so überlagert man die transparenten Karten – auch Layer oder Themen genannt – „Gewässer“ und „Länder“ (s. Abb. 3.4). Karten (und auch Pläne, z.B. Netzpläne) spielen im Rahmen eines GIS sowohl für die zu erwartenen Präsentationen der Datenbestände, die Datenbestände selbst aber auch für die Datenerfassung, -pflege und -manipulation eine große Rolle. Manchmal wird ein GIS als Sammlung digitaler Karten definiert.

Gut werden beispielhaft die Möglichkeiten eines GIS im Bereich Stadtplanung anhand einiger der an die DSGK (s.o.) gestellten Anforderungen deutlich, denn zusätzlich zu den Möglichkeiten der analogen Flurkarte soll die DSGK folgendes bieten ([Fre87]):

- Selektions- und Kombinationsmöglichkeiten der Karteninhalte
- Flexible Ausgabe hinsichtlich Maßstab und Abgrenzung
- Verknüpfbarkeit mit anderen Sachdaten (z.B. mit dem Automatisierten Liegenschaftsbuch oder dem Informationssystem der HEW)
- Erweiterung der Karteninhalte
- Verbesserte Genauigkeit

3.2 Struktur von Geographischen Informationssystemen

GIS werden in der Regel in vier Komponenten aufgeteilt (s. Abb. 3.2): Erfassungssystem, Verwaltungssystem, Analysesystem und Präsentationssystem. Obwohl frühe GIS keine echten Datenbanken verwendet haben, impliziert das Wort GIS heute die Benutzung einer (räumlichen) Datenbank (Spatial Database). Dies kommt ja auch in der obigen Definition eines Informationssystems zum Ausdruck.

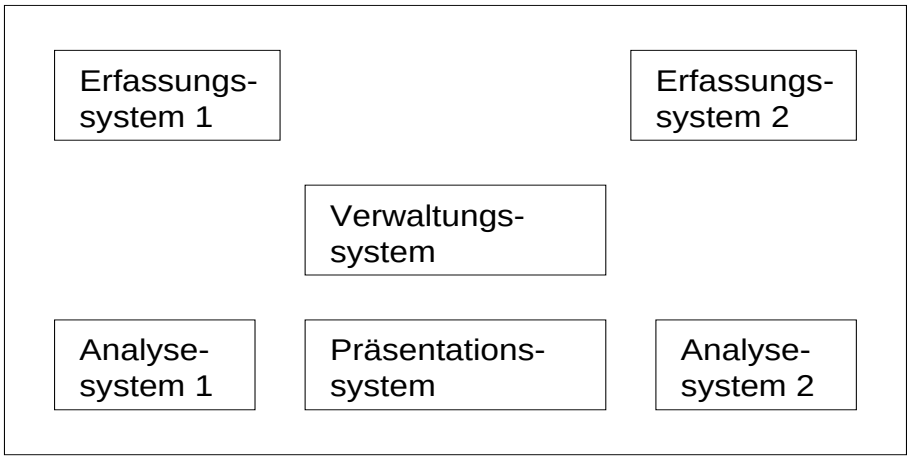


Abbildung 3.2: Struktur eines GIS (reproduziert nach [Bar95])

Erfassungssystem: Die Datenerfassung bzw. Daten selbst stellen einen besonders kostenintensiven Teil eines GIS dar. Letzlich ist ein GIS ohne die Daten wertlos, und für die Daten sind (je nach Anwendungskontext) Kriterien wie Fehlerlosigkeit, Genauigkeit, Aktualität und Vollständigkeit entscheidend – natürlich müssen die Daten kontinuierlich gewartet und gepflegt werden. GIS-Datenbestände werden oftmals anhand analoger Karten durch Scannen oder (manuelles) Digitalisieren erstellt. So wurde z.B. die Aufgabe der Digitalisierung der historisch gewachsenen Kartenbestände des Hamburger Vermessungsamtes an Ingenieurbüros vergeben. Dabei entstehen oft Probleme mit unterschiedlicher Genauigkeit und Aktualität der einzelnen Papierkarten. So schreibt das Hamburger Vermessungsamt über seine Flurkarte 1 : 1000: „Etwa die Hälfte der Karten erfüllt die heutigen Qualitäts- und Inhaltsanforderungen nur eingeschränkt."

In anderen Anwendungskontexten können Daten aber auch direkter gewonnen werden, z.B. durch Satellitenphotos oder Luftbilder (Orthophotos). Multispektralbilder können weitgehend automatisch klassifiziert werden.

Verwaltungssystem: In der Regel ist es mit einer einmaligen Erfassung aller Daten nicht getan: Daten müssen gepflegt und aktualisiert werden. Lediglich für spezielle Anwendungen (z.B. in den Geowissenschaften) wird man es mit überwiegend unveränderlichen Strukturen zu tun haben. Sobald es jedoch um die Verwaltung menschlicher Aktivitäten geht, ist die Frage der Aktualität der Daten von größter Bedeutung: z.B. beginnt die Anforderungsliste zur DSGK mit den Punkten „Flächendeckende Einrichtung in einem überschaubaren Zeitraum" und „Hohe Aktualität bestimmter Karteninhalte", wodurch also *zeitliche Aspekte* betont werden. Natürlich ist hierin ein wesentlicher Vorteil rechnergestützter Methoden zu sehen: die Wartung eines analogen Kartenbestandes stellt sich sehr viel zeitaufwendiger dar – u.U. müssen Karten *komplett neu erstellt werden.* Ist jedoch ein digitaler Grunddatenbestand erst einmal in einem GIS vorhanden, so können umfangreiche Änderung relativ schnell (und „lokal") durchgeführt werden. Aber auch vor Rechnergläubigkeit muß gewarnt werden: oftmals werden Daten lediglich aufgrund ihrer digitalen Rechner-Präsenz Attribute wie Objektivität, Aktualität, Genauigkeit etc. zugesprochen, die sie evtl. nicht haben. Hier wird deutlich, daß Metadaten (in Form von Dokumentation über die Daten) eine wichtige Rolle spielen. Im Vermessungswesen muß z.B. für jeden vermessenen Punkt Genauigkeit, Tag der Erfassung, verwendetes Werkzeug, Erfasser etc. mit katalogisiert werden. Diese Metadaten gehören zur *Thematik* (s.u.) eines vermessenen Punktes.

Analysesystem: Während die Komponenten Erfassungssystem und Verwaltungssystem in der Regel nicht in jeder GIS-Anwendung zu finden sein werden (z.B. bietet das Hamburger Vermessungsamt seine Daten zum Verkauf an), so wird jedoch jeder Anwender zumindest ein Analyse-

und/oder Präsentationssystem auf den verwalteten Daten benutzen. Sowohl Analyse als auch Präsentation dieser Daten ist hochgradig anwendungsspezifisch. Während also Erfassung und Verwaltung (in größeren Bereichen) relativ anwendungsunabhängig geschehen können, so ist dies für Analyse und Präsentation nicht der Fall. Die DSGK (und auch ihr Vorgänger, die Flurkarte) ist so ausgelegt, daß eine Vielzahl unterschiedlichster Anwendungen mit ihr realisiert werden kann (s.o.).

Auch hier werden wieder die Vorteile digitaler Karten deutlich: ein Anwender muß lediglich die für ihn relevanten Daten selektieren und kann den Rest der verwalteten Daten ignorieren. Alsdann kann er diese Daten beliebig kombinieren: so können versteckte Zusammenhänge deutlich werden (Data Mining). *Wesentlich ist also, daß Daten in Zusammenhänge gebracht werden können, die weder voraussehbar waren noch vorausgesehen werden sollten.* Wesentlich ist jedoch auch, daß die so erhaltenen Resultate angemessen interpretiert werden und ihnen somit nicht Attribute zugesprochen werden, die sie gar nicht haben. Typische Analysen sind z.B.

- Routensuche (Welches ist der schnellste Weg von Hamburg nach München?)
- Ortung (Wo liegt Hamburg?)
- Standortwahl (Bestimmte den besten Standort für eine neue Schule!)

Hier sind insbesondere auch die *Anfragesprachen (Query Languages)* hervorzuheben: sie dienen ganz wesentlich zur Analyse bzw. Informationsvermittlung über einen Datenbestand.

Heutzutage sollten Analysen mit grafisch-interaktiven Methoden, also mit Hilfe von Visualisierungen und direkt-manipulativen Benutzungsoberflächen erfolgen. Dies ist im GIS-Kontext umso verständlicher, als daß es hier oft um Karten und Pläne geht, also inhärent visuelle Objekte. So stellt sich auch die Frage, ob Analyse- und Präsentationskomponente als (logisch) getrennt betrachtet werden sollten – sicherlich handelt es sich um einen wechselseitigen, interaktiven Prozeß ([Ege90]). Natürlich können Programme die Datenbestände ohne jegliche Visualisierung analysieren. Für einen menschlichen Benutzer sollten Analyse- und Präsentation jedoch im Rahmen einer integrierenden Benutzungsoberfläche gestaltet werden. Schließlich ist Kap. 4 dieser Arbeit den *visuellen räumlichen Anfragesprachen* gewidmet.

Präsentationssystem: Die zu erwartenen Präsentationen sind ebenso anwendungsspezisch wie die vorzunehmenden Analysen selbst. So mag es z.B. für die Überwachung und Analyse eines Leitungsnetzes vielleicht ausreichen, lediglich gewisse Daten (wie Belastung o.ä.) in Form von Zahlen oder Balkendiagrammen planartig verteilt darzustellen. Oftmals werden jedoch kartenartige Darstellungen benötigt, insbesondere dann, wenn räumliche Anordnungen und Zusammenhänge im Raum selbst von Interesse sind bzw. erkannt werden sollen. In die automatische Generierung von Karten anhand räumlicher Datenbestände wurde bereits viel Forschungskapazität investiert: die Generierung ansprechender Karten ist jedoch dermaßen komplex (und zudem schlecht formalisierbar), so daß rechnererzeugte Karten auch heute noch in der visuellen Qualität beträchtlich ihren analogen Vorfahren unterlegen sind. Heutige Präsentationssysteme können keine ansprechenden Karten erzeugen – Karten werden weiterhin (wenn auch grafisch-interaktiv am Rechner) manuell von einem Menschen erstellt. Insbesondere wurde in der Vergangenheit versucht, mit KI-Techniken (Expertensysteme u.ä.) Fortschritte in dieser Richtung zu erzielen (s. die Übersicht in [KFAF89]).

Da die Datenbestände inhärent räumlich sind, könnten die Daten stets 1-zu-1 präsentiert werden (z.B. könnte in einem vektorbasierten GIS eine Linienzeichnung anhand der Geo-Objekte erstellt werden). Eine solche Darstellung hat jedoch nichts mit den ausgefeilten Visualisierungsmethoden der Kartographie zu tun: eine Karte soll sowohl informativ als auch ästhetisch sein. Insbesondere letzerer Punkt läßt sich wohl kaum formalisieren. Eine Formalisierung ist jedoch Voraussetzung für eine Implementierung.

Visualisierung läßt sich folgendermaßen definieren (aus [Bar95], frei nach B. P. Buttenfield und W. A. Mackaness):

> Visualisierung ist der Prozeß der ganzheitlichen Darstellung zum Zweck des Erkennens, der Vermittlung und Interpretation von Strukturen. Darstellungen sind oft grafisch-symbolischer Natur und unterscheiden sich dadurch von einer textlichen, sprachlichen oder formelhaften Ausdrucksweise.

Da eine Visualisierung nicht zweckfrei ist, müssen gewisse Richtlinien beachtet werden. Insbesondere zwei Fragen sind zu beantworten:

1. **Auswahl:** Welche Informtionen soll zur Zeit vermittelt werden?

2. **Betonung:** Wie kann diese Information ansprechend hervorgehoben werden?

So soll es dem Menschen ermöglicht werden, die relevanten Informationen herauszulesen bzw. die „Spreu vom Weizen“ zu trennen. Darstellungen dürfen nicht mit irrelevanten Aspekten überladen werden. Wichtige Fragen bzgl. der Kartengestaltung sind auch Farbgestaltung, Wahl von Texturen, Wahl von Symbolen und Schriftarten, Größe von Symbolen und Schriftarten. Hierbei werden also die sog. *visuellen Variablen* Größe, Helligkeitswert, Textur, Farbe, Richtung und Form (nach Bertin) möglichst geschickt kombiniert. Die *Legende* listet die Bedeutung der entsp. Symbole, Farben und Texturen auf. Ein gute Übersicht über diese Visualisierungsmethoden ist [McC83] - ähnliche Fragen müssen bei der Gestaltung jeglicher visueller Darstellungen (u.a. für Benutzungsoberflächen, s. [Cha90, Kap. 2]) behandelt werden.

Ein wichtiges Mittel ist auch die *Generalisierung:* je nach zu vermittelnder Information bzw. Zweck der Karte kann ein in einem GIS gespeichertes Haus einmal als Symbol, ein anderes mal jedoch als Grundrißdarstellung präsentiert werden. Hier spielt natürlich auch der Maßstab und der Zweck der Karte eine wichtige Rolle: eine Straße wird manchmal als Linie und manchmal als langgestreckte Fläche erscheinen. Verläufe von Flüssen werden begradigt, kleinere Schlaufen erscheinen nicht. Autobahnen werden als rote dicke Linien dargestellt, normale Straßen als dünne schwarze Linien. Die Dicke der Linie hat meist wenig mit der Breite der Straße zu tun. Die Einwohnerzahl von Städten wird in einer Europakarte z.B. durch entsp. große Kreise o.ä. visualisiert. Die Größe der Kreise hat nichts mit der räumlichen Ausdehnung und Form der Städte zu tun. Insofern wird hier (in Teilen) die *Geometrie der Karte* durch den darzustellenden Sachverhalt bzw. die Thematik geprägt – die durch die Karte repräsentierte Geometrie wird teilweise verfälscht ([BF97]). Wesentlich ist hier wieder die Kenntnis der Interpretationsfunktion und die Berücksichtigung der Legende – anderseits besteht die Gefahr der Fehlinterpretation.

3.3 Datenmodelle für GIS

Prinzipiell gibt es zwei Klassen von GIS: *raster-* und *vektorbasierte GIS.* Die diesen GIS zugrundliegenden *Datenmodelle* werden *Raster-* und *Vektormodelle* genannt. Hierbei handelt es sich um zwei verschiedene Repräsentationsmöglichkeiten der Erdoberfläche (bzw. des Interessenbereiches). Die Wahl zugunsten eines dieser Modelle ist ganz wesentlich von den beabsichtigten Anwendungen des GIS abhängig: wie jede Repräsentationsform haben auch diese ihre Stärken und Schwächen. Mit einer Modellierung oder Repräsentation ist natürlich immer eine Abstraktion gewisser Aspekte gegeben. GIS-Anwendungen sind in der Regel (wie Karten) zweidimensional. Lediglich im Rahmen *Digitaler Geländemodelle (DGM)* wird manchmal auch die dritte Dimension berücksichtigt. Im allgemeinen spielt sie jedoch eine bescheidende Rolle und wird allerhöchstens als *zusätzliches Attribut* mitgeführt. Insofern sind auch die verwendeten Datenmodelle zweidimensional, im Gegensatz zum CAD-Bereich. Da die dritte Dimension manchmal als Attribut mitgeführt wird, bezeichnet man GIS auch als 2,5-dimensional.

Zunächst muß noch der wichtige Begriff der *Thematik* geklärt werden. So schreibt Bartelme ([Bar95]):

> Geoinformation stützt sich auf zwei Säulen: Eine dieser beiden Säulen wird von der *Geometrie* der Daten gebildet; die andere Säule ist die *Thematik* bzw. *Semantik.*

Beide Aspekte müssen in den entsp. Datenmodellen für GIS berücksichtigt werden.

3.3.1 Thematik

Der Begriff „Thematik“ kommt aus der Kartographie: dort unterscheidet man u.a. zwischen

- topographischen und
- thematischen Karten.[2]

Topographische Karten dienen hauptsächlich der Darstellung von Teilen der Erdoberfläche. Auf ihnen findet man wesentliche Charakteristika der Erdoberfläche, wie Gewässer, Küstenverläufe, Landhöhen und Gebirge, Verkehrswege und Städte, Ländergrenzen etc. *Thematische Karten* stellen eines oder einige wenige *Themen* in der Vordergrund: sie stellen die primär zu vermittelnden Informationen dar, wozu besondere Visualisierungstechniken verwendet werden. Politische Karten stellen z.B. das Thema „Staatenzugehörigkeit“ durch Farben gesondert dar. Die *Geometrie* topographischer Karten bildet dabei den *Visualisierungshintergrund* für die Darstellung des Themas.

Primär geht es beim Begriff der Thematik also um die Bedeutung (Semantik) von Darstellungen. Die Datenmodelle für GIS müssen somit nicht nur die Geometrie, sondern auch die Semantik der repräsentierten Objekte beschreiben können – diese semantischen Attribute müssen dann von einer Präsentationskomponente adäquat visualisiert werden.

3.3.2 Rastermodelle

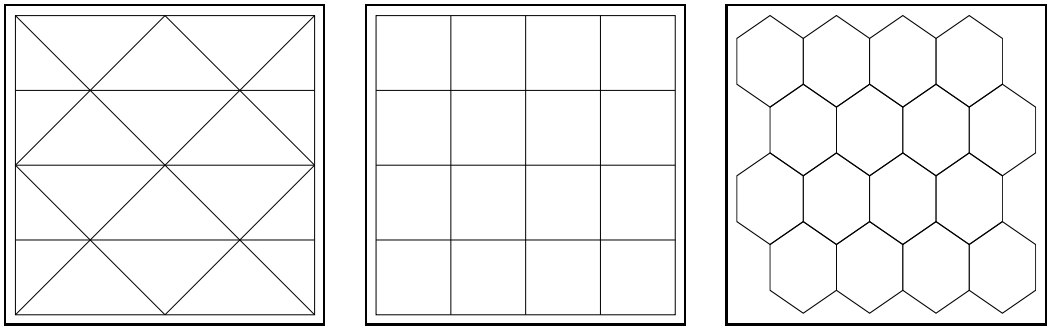


Abbildung 3.3: Einige Parkettierungen

Beim *Rastermodell* handelt es sich konzeptionell um eine *Parkettierung* der Ebene (s. Abb. 3.3); man spricht auch von *Mosaik* oder *Tesselation*, die einzelnen Teilstücke des Parkettes werden Kacheln genannt. Verschiedene Ausprägungen dieses Modelles sind denkbar: u.a. läßt sich die Ebene mit Rechtecken, Sechsecken und Dreiecken parkettieren. Für jede einzelne Kachel gilt die Forderung, daß sie homogene Thematik hat; die einzelnen Kacheln sind kongruent zueinander. Unregelmäßige Parkettierungen lassen sich im Vektormodell darstellen.

[2] Eine vollständige Taxonomie von Karten scheint es nicht zu geben – u.a. gibt es Wetterkarten, politische Karten, Straßenkarten, Topographische Karten, Seekarten, Sternenkarten, Wirtschaftskarten, etc.

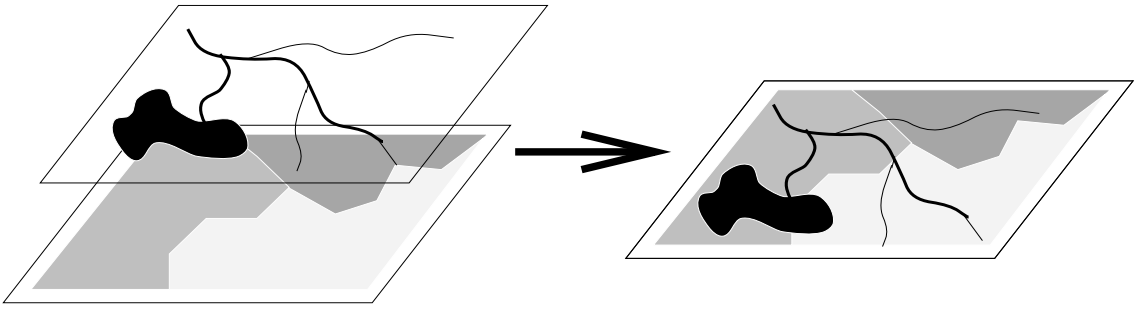


Abbildung 3.4: Verschneidung (Überlagerung) der Themen „Gewässer“ und „Staaten“

Die häufigste Ausprägung des Rastermodelles ist einfach das regelmäßige Gitter; die einzelnen Kacheln heißen dann auch Pixel (Picture Elements). Die einfachste Implementierung dieses konzeptionellen Datenmodelles wird daher ein Speicherraster (Array) sein – der *Quadtree* stellt eine sehr speicherökonomische Implementierungsvariante dar.

Der *Farb-* oder *Grauwert eines Pixels* wird nun einfach als dessen *Thematik* definiert: im Gegensatz zur Computergrafik sind die Begriff „Farbe“ und „Grauwert“ jedoch nicht wörtlich zu nehmen. Die Thematik jeder Kachel kann nun durch Nachschlagen in einer Tabelle ermittelt werden: so könnte 0 für „Weder Wasser noch Wald“ stehen, 1 für „Wasser“, 2 für „Wald“, etc. Ein Pixel repräsentiert also ein Stück der Erdoberfläche bzw. des Interessenbereiches. Da in der Regel vorher Transformationen stattgefunden haben (z.B. *Entzerrungen*), hat das repräsentierte Stück der Erdoberfläche nicht unbedingt die gleiche Form wie das Pixel. Der Farbwert eines Pixels muß richtig interpretiert werden, denn natürlich handelt es sich bei der Thematik um eine Abstraktion bzw. Generalisierung: zum einen könnte es sich hierbei um die *gemittelte Thematik*, zum anderen aber auch um die Thematik des Mittelpunktes dieses Ebenenstückes der „Welt“ handeln. In der Regel wird ersteres der Fall sein (man denke z.B. an ein Luftbild).

Mehrere Themen pro Pixel können nun definiert werden, indem mehrere sog. *Schichten oder Ebenen (Layer)* angelegt werden. Jede Ebene definiert dabei ein Thema (oder einige wenige) für seine Pixel: durch *Überlagerung* dieser Ebenen können dann durch die *gemeinsame Geometrie* Themen *kombiniert* werden, s. Abb. 3.4. Hierbei handelt es sich um ein klassisches Vorgehen aus der Kartographie, die analoge Karten einst durch Überlagerung von transparenten *Farbauszügen* herstellte.

Mehrere Themen können aber auch auf einer einzigen Schicht untergebracht werden: während dies für „disjunkte“ Bedeutungen klar ist (die Geometrie der Themen „Wasser“ und „Land“ ist disjunkt), so können einem Pixel im allgemeinen einfach mehrere Farben bzw. Themen zugewiesen werden – hier würden dann mehrere Schichten wieder zu einer einzigen verschmolzen. Natürlich können auch mehrere Kombinationen von Themen wieder als einzelne Zahlen kodiert werden (ebenso, wie dies mit den Farben Rot, Grün und Blau im Speicher einer Grafikeinheit geschieht).

Oftmals findet man in der Literatur die Aussage, daß das Rastermodell (gegenüber dem Vektormodell) den Vorteil hat, daß es keine Stellen im Interessenbereich ohne Thematik gibt – in Bezug auf die verwendeten Themen herrscht für jedes Pixel Klarheit darüber, ob ein entsp. Thema gilt oder nicht.[3] Die vollständige einheitliche Bedeckung der Ebene durch eine Schicht wird auch durch den Begriff *Überdeckung (Coverage)* zum Ausdruck gebracht. Schließlich sei noch darauf hingewiesen, daß eine vollständige Bedeckung bzw. Parkettierung der Ebene natürlich auch mit Polygonen im Vektormodell geschehen kann.

Die *Hauptvorteile* des Rastermodelles sind:

[3]Denkt man aber an die Thematik „Unbekannt“, die z.B. für noch nicht klassifizierte Bereiche der Ebene vergeben werden könnte, so herrscht zwar an einem Punkt Klarheit darüber, ob das Thema „Unbekannt“ gilt oder nicht, dennoch handelt es sich hier um einen „weißen Fleck“ auf der Karte.

Vollständige Flächenüberdeckung: Es gibt keine Flächen mit unbekannter Thematik (bzgl. des Interessenbereiches). Das Rastermodell ist für die Repräsentation flächiger Zusammenhänge besser geignet als das Vektormodell – die konzeptionelle Einfachheit des Modelles macht zudem die Analyse dieser flächigen Zusammenhänge sehr einfach.

Einfachheit: Für viele Anwendungen (z.B. die Fernerkundung) ist das Rastermodell ideal. Viele Operationen lassen sich sehr einfach implementieren – das *Überlagern* oder *Verschneiden* von Schichten ist trivial: der Grauwert eines Pixels im Ergebnislayer ergibt sich durch die Anwendung einer logischen Operation (meist UND) auf den Grauwerten der beiden Pixel der zu verschneidenen Layer. Andere oft benötigte Operationen wie das *Reklassifizieren* setzen einfach alle momentan nicht relevanten Themenkodierungen von Pixeln auf 0, so daß im Raster nur noch das relevante Thema erscheint (alle anderen Pixel sind auf 0 gesetzt). C. D. Tomlin hat eine sog. *Map Algebra* geschaffen: er zeigte, daß mit der minimalen Liste von 64 Operatoren alle Wünsche bzgl. Rasteroperationen erfüllt werden können. Auch die *räumlichen Selektionen* von relevanten Teilen des Rasters sind einfach zu implementieren: im Vektormodell sind hierfür aufwendige Indizierungsstrukturen vorzusehen (s.u.). Eine räumliche Selektion wird in der Regel durch Vorgabe eines räumlichen Objektes und einer Reihe von Bedingungen vorgenommen. So könnte der Benutzer ein Rechteck und die Bedingung „Enthält Pixel“ vorgeben. Als Selektionsergebnis erhielte er den durch das Rechteck ausgeschnittenen Teil des Rasters. Erwähnenswert ist auch die *Pufferzonenbildung*: hierzu wird vom Benutzer z.B. eine Linie vorgeben, und es wird dann der Teil des Rasters selektiert, dessen Pixel nicht weiter als r Meter von der vorgegebenen Linie entfernt sind.

Einfache Datenerfassung: Die Datenerfassung ist in der Regel kein so aufwendiger Prozeß wie bei den Vektormodellen, da die Daten z.B. in Form von Satelliten- oder Luftbildern vorliegen. Konventionelle Papierkarten können gescannt werden – bei Vektormodellen müssen diese Karten dem sehr aufwendigen (und teuren) Prozeß der manuellen Digitalisierung unterzogen werden: automatische Verfahren können hier zwar eingesetzt werden, anschließend ist jedoch eine sehr aufwendige Nachbearbeitung notwendig.

Hauptnachteile des Rastermodelles sind

Speicherbedarf: Größenordnungen für einzelne Themen liegen sehr schell im Mega- und Gigabyte-Bereich. Einige globale (sinnvoll verwendbare) Weltkarte würde im Terabyte-Bereich liegen. Relationale Datenbanken können nicht verwendet werden. Meist werden die einzelnen Layer in Dateien oder speziellen Bilddatenbanken gespeichert: relationale Datenbanken archivieren dann lediglich Verweise und Metainformationen über diese.

Keine Dynamik: Die einmal vorgenommene Einteilung der Layer in Themen im Sinne eines „Top-Down“-Vorgehens kann schlecht rückgängig oder adaptiert werden. Individuelle Zusammenstellungen sind aufwendiger als im Vektormodell, da verschiedene Layer kombiniert und neu erzeugt werden müssen.

Keine Objekte: Oftmals müssen die dargestellten geograpischen Entitäten *individuell* angesprochen und behandelt werden – da jede Entität aber über viele Pixel verteilt ist, muß man entweder pro Entität ein eigenes Thema definieren (wobei man sich dann wieder einem „Bottum-Up“-Vorgehen nähert), oder aber z.B. mit Methoden der Bildverarbeitung Segmentierungen o.ä. vornehmen.

Homogene Auflösung: Die einheitliche Größe der Pixel bedingt immer eine Ungenauigkeit bzgl. thematischer Informationen. Je feiner die Rastermaschen sind, desto geringer ist dieser Effekt. Diese *Quantisierung* tritt natürlich auch für die Geometrie auf, die sich hier ja in gewisser Weise erst aufgrund der Thematik ergibt (nämlich durch die thematischen Färbungen der Pixel bzw. Aufteilungen auf Layer).

Schwierige Implementierbarkeit bestimmter Operationen: Transformationen (Skalierungen, Rotationen etc.) sowie Flächen- und Umfangsberechnungen sind sowohl aufwendig zu implementieren als auch mit (im Vergleich zum Vektormodell) größerer Ungenauigkeit behaftet.

Vergleichbare Datenbestände: Damit aus evtl. verschiedenen Datenquellen kommende Layer überhaupt kombiniert werden können, müssen sowohl die verwendeten Koordinatensysteme, Auflösungen (Genauigkeit) als auch Rastermaschenweiten vergleichbare Werte haben (nur in kleineren Abweichungen können hier Transformationen helfen). Diese Forderungen sind in der Praxis häufig nicht erfüllt, da es sich um historisch gewachsene, heterogene Datenbestände aus verschiedenen Quellen handelt. So variiert z.B. die räumliche Auflösung eines Multispektralbildes der Erdoberfläche mit der Wellenlänge (also den einzelnen Kanälen). Hiermit einher gehen Fragestellungen der *Genauigkeit* und *Qualität* von Geodaten. Werden komplizierte Operationen auf verschiedenste Daten angewendet, so müssen die entstehenden Ergebnisse besonders kritisch interpretiert werden (nach Möglichkeit sollte die *Fehlerfortpflanzung* berücksichtigt werden).

Rastermodelle sollen im weiteren Verlauf dieser Arbeit nicht mehr betrachtet werden, da die Sprache VISCO ein Vektormodell voraussetzt.

3.3.3 Vektormodelle

Das Vorgehen bei der Vektormodellierung kann in gewisser Weise als *diametral* zum Vorgehen bei der Rastermodellierung bezeichnet werden: während die Rastermodelle den Interessenbereich im Sinne einer „Top-Down"-Unterteilung in einzelne Themen aufteilen, werden hier aus individuellen Entitäten „Bottom-Up" komplexere Gebilde aufgebaut.

Die hier relevanten Objekte bzw. Bausteine liegen ja auf der Hand: *Objekte wie Punkte, Linien und Flächen bzw. Polygone können als fundamentale Abstraktionen räumlicher Gegebenheiten bezeichnet werden ([Güt94]).* Diese Objekte werde auch als *SDTs (Spatial Data Types)* bezeichnet – die Objekte und die auf ihnen definierten Operationen werden als *räumliche Algebra* bezeichnet (im Sinne der Theorie der *Abstrakten Datentypen, ADTs*). Oftmals werden diese Operationen unter Verwendung von Algorithmen aus dem Bereich der algorithmischen Geometrie (Computational Geometry) realisiert: so müssen Algorithmen zur Berechnung räumlicher Relationen, zur Berechnung von Schnittobjekten etc. effizient implementiert werden. Von großem Interesse sind hier *Sweep Line*-Verfahren ([SDK96]).

In der Praxis wird die Implementierung von Operationen einer räumlichen Algebra vor allem durch numerische Probleme erschwert: in [GS93, Sch96] wird eine diskrete geometrische Basis (sog. „REALMS") als Grundlage einer komplexen Algebra („ROSE") vorgestellt, wodurch Probleme, die sich aufgrund der Repräsentation von in der Ebene $\mathbb{R}^2$ eingebetteten Objekten durch Fließkommazahlen ergeben, verschwinden.

Geometrie im Vektormodell

Während im Rastermodell einzelne Flächen der „Welt" durch Pixel repräsentiert werden, werden hier zu Punkten abstrahierte Objekte der Welt durch *Punktentitäten* oder *Punktobjekte* repräsentiert. Punktobjekte sind hier das primitive Element – alsdann können Linienobjekte anhand von Punktobjekten, Polygonobjekte anhand von Punkt- oder Linienobjekten und schließlich auch Komplexobjekte (beliebige Aggregate) gebildet werden. Punktobjekte haben die geometrische Position des repräsentierten Punktes als *Attribut. Die Geometrie ist somit im Vektormodell nur eines unter vielen Attributen.*

Unter einer *Linie* wird meist eine geometrische *Strecke* (eine geradlinige Verbindung zwischen zwei Punkten) verstanden: viele GIS betrachten eine Linie jedoch im Sinne der *Topologie* bzw. als

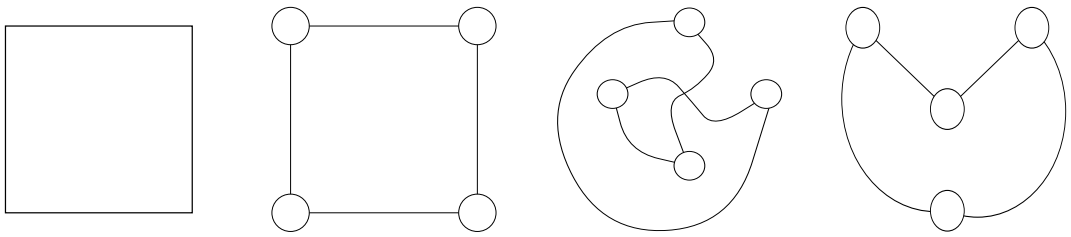


Abbildung 3.5: Geometrie und Graphen

Kante, die zwischen sog. *Knoten (Endpunkten)* verläuft.[4] Hierbei handelt es sich dann um eine Knoten-Kanten-Struktur, auch als (planarer) Graph bezeichnet: Graphen werden im Rahmen der *Graphentheorie* oder *kombinatorischen Topologie* untersucht.

Der genaue Verlauf einer Linie kann durch eine (evtl. leere) Liste von Zwischenpunkten repräsentiert werden, und eine Berechnungsvorschrift wird zwischen diesen Punkten *interpolieren* (z.B. Splines). Hier gibt es dann zwei Klassen von repräsentierten Punkten: Punktobjekte und Zwischenpunkte. Die Zwischenpunkte werden meist nicht als eigenständige Entitäten modelliert.

Jedes Vektormodell muß zumindest Punkte und Linien explizit machen: natürlich ist es nicht sinnvoll, auf Linien-Entitäten zu verzichten und stattdessen nur Punkte zu verwenden, wovon man sehr sehr viele benötigen würde. Eine sehr grobe Quantisierung und gleichmäßige Verteilung von z.B. „Meßpunkten“ würde uns dann wieder zum Rastermodell bringen. Hier wird ja der topologische Begriff der *Nachbarschaft* und somit auch die *Verbundenheit von Punkten*[5] implizit durch die Anordnung der Rasterzellen vorgegeben. Schließlich sei noch erwähnt, daß jedes Raster natürlich auch wieder durch einen Graphen repräsentiert werden kann: pro Pixel wird ein Knoten eingeführt, und je nach verwendeter Nachbarschaftsbeziehung (4er- oder 8er-Nachbarschaft bei rechteckigen Pixeln) werden entsp. Kanten eingeführt. Zwei Knoten sind genau dann benachbart, wenn zwischen ihnen eine Kante verläuft. Hier treffen sich also die beiden Modelle: das Rastermodell kann auf das Vektormodell abgebildet werden. Betrachtet man das Rastermodell als spezielles Vektormodell, so werden auch Vergleiche zwischen beiden möglich.

Prinzipiell unterscheidet man nun zwischen

- topologisch strukturierten Vektordaten und
- nicht topologisch strukturierten Vektordaten (auch „Spaghetti“ genannt).

Grob gesagt liegt den ersteren im Gegensatz zu letzteren eine Graphenstruktur zugrunde. An jeder „Stelle“ des Raumes gibt es nur einen Punkt: diesem Umstand wird bei topologisch strukturierten Vektordaten dadurch Rechnung getragen, daß keine zwei Knoten die gleiche Position haben dürfen. Die Position ist hier ein Attribut der Knoten des Graphen. Ein weiteres Attribut der Kanten könnte die bereits erwähnte Liste von Zwischenpunkten und die verwendete Interpolationsvorschrift sein. Es mag jedoch auch Anwendungen geben, wo überhaupt keine geometrischen Attribute benötigt werden (z.B. für Netzwerkanalysen). Wenn man die „Position“ eines Knoten in der *Visualisierung* eines Graphen als Position des repräsentierten Punktes und den Verlauf der Kante als Verlauf der repräsentierten Kante selbst auffaßt, so sieht man, daß die topologische Graphenstruktur dann auch die *Geometrie* wieder perfekt beschreibt. Dann handelt es sich um eine *direkte Repräsentation* (s. Kap. 2). In Abb. 3.5 wird dargestellt, wie die Geometrie eines Vierecks durch einen Graphen repräsentiert werden kann. Eine der vier äquivalenten Darstellungen

[4] Der Begriff „Vektor“ bringt ungerechtfertigter Weise wieder die Geradlinigkeit der Verbindungen zum Ausdruck.

[5] Es gibt jedoch im Rahmen der Rastermodelle Probleme mit diesen Begriffen: sie werden von der *Digitalen Topologie* untersucht.

dieses Graphen hat wieder die Geometrie des Viereckes selbst – zur Erzeugung dieser Darstellung werden einfach die Geometrieattribute der Knoten und Kanten genutzt. Normalerweise führt die geometrische Interpretation einer Graphendarstellung jedoch zur Überinterpretation.

Linien werden nun als Kanten zwischen Knoten repräsentiert, so daß ein Graph entsteht: zwischen zwei Knoten verläuft höchstens eine Linie (kein Multigraph). Da es sich um einen planaren Graphen handelt, dürfen sich die Kanten nicht schneiden: tun sie es, so muß ein weiterer „Schnittknoten" aufgenommen werden und die Kanten werden entsp. unterteilt.

Polygone können nun explizit sein oder nicht: ein explizites Polygon wird eine Reihe von Kanten (die – im Sinne der Graphentheorie - einen sog. *Kreis* bilden) *zusammenfassen.* Durch weitere Zusammenfassungen (Aggregierungen) können beliebige Komplexe aufgebaut werden: zusätzlich zur Kante „Direkt verbunden mit" (die ja eine Linie repräsentiert) wurde bereits beim Polygon die Kante „Komponente von" notwendig.[6] Die „Komponente von"-Struktur bildet einen DAG (engl. für „directed acyclic graph", azyklischer gerichteter Graph).

Ob es sinnvoll ist oder nicht, Polygone (also Kreise oder Zyklen) explizit durch einen Knoten auszuzeichnen, hängt wieder von der speziellen Anwendung ab. Letzlich könnte ein Suchprogramm Zyklen und somit (einfache) Polygone im Graphen erkennen. Dies erfordert aber in der Regel einen beträchtlichen Aufwand, und zumal kann oft algorithmisch nicht ohne weiteres geklärt werden, welches denn nun die relevanten Polygone sind (s. Kap. 2). So schreibt das Hamburger Vermessungsamt in [Fre96a] über die DSGK (Digitale Stadtgrundkarte):

> Dabei findet in einigen Bereichen eine Objektbildung statt, die jedoch nur den unmittelbaren Kundenwünschen folgt und keineswegs versucht, Inhalte bis ins letzte Detail durchzustrukturieren und zu schwerer fortzuführenden Objekten zusammenzufassen.

Natürlich ist die gesamte Geometrie einer Karte bereits durch Punkte und Linien beschrieben: alle weiteren *Zusammenfassungen* dienen lediglich der *Strukturierung von Inhalten, also der Thematik.*

Der große Vorteil topologisch strukturierter Vektordaten liegt daran, daß man einen leistungsfähigen Formalismus in Form der Graphentheorie zur Verfügung hat. So lassen sich z.B. viele *Konsistenzbedingungen* für die Daten recht einfach formulieren: ein Graph soll planar sein, soll evtl. keine isolierten Knoten enthalten, soll evtl. keine Zyklen haben, etc. Viele Anwendungen profitieren von einer Graphstruktur: eine Routenplanung ist auf topologischen Daten unmittelbar durchführbar – Graphenverfahren nehmen im sog. *Operations Research* eine wesentliche Rolle ein.

Zudem liegen die wesentlichsten topologischen Beziehungen *explizit* vor: durch die Graphstruktur können daher viele Fragen ohne Suche in den Daten beantwortet werden. Während die Frage „Welche Linien haben Punkt xyz als Endpunkt?" in topologisch strukturierten Vektordaten einfach durch Auflistung aller abgehenden Kanten (die ja explizit gespeichert sind) beantwortet weden kann, so erfordert die Beantwortung dieser Frage in einem nicht-topologisch strukturierten Datenbestand pro vorhandenem Linienobjekt zwei Koordinatenvergleiche. Da jedes Vektormodell die Geometrie explizit machen muß, ist natürlich die Topologie stets (anhand einer entsp. Metrik) *rekonstruierbar*, s. Kap. 2. Der Unterschied liegt also im wesentlichen zwischen expliziter und impliziter Topologie.

Während die Begriffe

- Punkt (Point Feature)
- Linie (Line Feature)
- (einfache) Fläche oder Polygon (Area Feature)

allgemein sowohl für topologisch als auch nicht-topologisch strukturierte Vektordaten verwendet werden, werden die Elemente topologisch strukturierter Vektordaten auch als

[6] Betrachtet man Polygone und Komplexe als Knoten und die „Komponente von"-Relationen als weitere Kanten in diesem Graphen, so bekommt man allerdings Probleme mit dem Konzept des planaren Graphen.

- Knoten (Nodes, Vertices)
- Kanten (Edges)
- Maschen (Faces)

bezeichnet. Es scheint mir jedoch konsequenter zu sagen, daß es sich bei Punkten, Linien und Polygonen um räumliche Objekte bzw. Konzepte handelt, die durch Knoten, Kanten und Maschen *repräsentiert* werden können.

Die in der Praxis benutzten Datenstrukturen sind oftmals nicht so einfach, wie es obige Diskussion vermuten lassen würde: so speichern z.B. Maschen meist ihren Umlaufsinn, Kanten speichern jeweils linke und rechte Masche, etc. All dies dient zum einen der Überprüfung topologischer Konsistenzbedingungen, zum anderen aber auch der effizienten Bearbeitung bestimmter Aufgaben. Hierfür sind u.a. wieder spezielle Indexstrukturen vorzusehen (s.u.).

In der Praxis werden auch *Textobjekte* und *kartographische Symbole* als weitere Primitive benötigt. Sie spielen nur für die Grafik, also die Visualisierung des Datenbestandes bzw. die Kartengenerierung eine Rolle. Weitere grafische Attribute (wie z.B. Füllmuster und Farbe von Polygonen) werden in der Regel als zusätzliche Attribute modelliert, oder aber von einer Präsentationskomponente z.B. anhand thematischer Sachverhalte ermittelt: „Autobahn“ ⇒ 3 mm dicke durchgezogen Linie. Bei einer solchen „Look Up“-Tabelle handelt es sich um die einfachste Form einer Präsentationskomponente.

Thematik im Vektormodell

Nachdem nun geklärt wurde, wie die Geometrie im Vektormodell repräsentiert wird, fehlt noch die *Thematik:* ebenso wie die Position als Attribut eines Punktobjektes definiert wurde, kann auch die Bedeutung (Semantik) als Attribut modelliert werden. Hier bedient man sich in der Praxis einer sog. *Objektschlüsseltabelle:* entsp. Themen werden wieder (wie auch im Rastermodell) durch bestimmte Zahlen kodiert. So wird z.B. in der DISK (s.u.) für Linienobjekte vom Typ „linienhafter Bach, nicht schiffbar“ die Nummer 5211 vergeben.

Man sollte evtl. auch an eine *Spezialisierungshierarchie von Bedeutungen bzw. „Objektschlüsseln“* denken: so könnte ein „linienhafter Bach, nicht schiffbar“ ein spezieller „Bach“ sein. Durch eine *objektorientierte Modellierung* könnte dies repräsentiert werden: objektorientierte Modelle werden erst seit kurzem in der GIS-Gemeinde untersucht. Vor allem die Möglichkeit, *unterschiedliche Verhaltensrepertoires für thematische Objekte* (in Form von Methoden) zu Verfügung zu stellen erregt hier Aufmerksamkeit. Bei der Kartendarstellung wird oftmals eine Generalisierung von Objekten vorgenommen – man könnte also bedeutungsspezifische Visualisierungsmethoden entwickeln (eine Straße wird anders generalisiert als ein Fluß, beide sind aber „Linien“).

Überlegungen zum Einsatz von *Beschreibungslogiken* (s. [WS92, BMPS$^+$91]) im GIS-Kontext findet der Leser bei Haarslev und Möller ([HM97a, MHL97]). Ziel dieser Untersuchungen ist es, anhand geometrischer und anderer bekannter thematischer Eigenschaften von Objekten (weitere) Bedeutungen bzw. Themen von anderen Objekten zu *inferieren.* Dies geschieht durch die von Beschreibungslogiken unterstütze dynamische Klassifikation von Objekten, was in der sog. A-Box geschieht. Die relevanten Konzepte oder Begriffe – die den Objekten dann dynamisch zugeordnet werden – werden dabei (ähnlich einer objektorientierten Modellierung mit Mehrfachvererbung) in einer terminologischen Taxonomie (die sog. T-Box ist ein DAG) angeordnet: das Konzept „linienhafter Bach, nicht schiffbar“ wäre ein Unterkonzept von „Bach“. Da die von Standardbeschreibungslogiken zur Verfügung gestellten Sprachmittel für die angestrebten Anwendungen nicht ausreichen, werden in [LHM97, MHL97, LM97] Spracherweiterungen und Fragen bzgl. Komplexität und Entscheidbarkeit untersucht.

Ein und dasselbe geometrische Objekt kann oftmals *mehrere Bedeutungen* bzw. Thematiken haben: Ein Punkt kann sowohl Teil einer Hausbegrenzung als auch einer Grundstücksbegrenzung sein,

womit er in verschiedenen *Rollen* auftritt. Somit sollte es möglich sein, einem geometrischen Objekt mehrere Themen zuzuweisen. Im Sinne einer objektorientierten oder beschreibungslogischen Modellierung könnte man Mehrfachvererbung nutzen. Ein geometrisches Objekt ist somit von einem thematischen Objekt zu unterscheiden: ein Grundstück ist nicht identisch mit der Fläche, die es einnimmt. Eine Möglichkeit wäre, einem thematischen Grundstücksobjekt als Attribut ein geometrisches Flächenobjekt zu geben: mehrere andere thematische Flächenobjekte könnten dieses geometrische Flächenobjekt ebenfalls nutzen. So könnte z.B. ein thematisches Komplexobjekt Verweise auf alle Flächenobjekte enthalten, die einer best. Person gehören – hierbei handelt es sich um das (sehr spezielle Thema) „Grundbesitz von XYZ“. Es wurde ja bereits erwähnt, daß die Aggregierungen zu weiteren Objekten im wesentlichen durch thematische Gesichtspunkte geleitet wird. Während die Geometrie bereits durch Punkte und Linien vollständig repräsentiert wird, ist dies für thematische Zusammenhänge eben nicht der Fall. Bartelme benutzt die treffende Metapher von „thematischen Fäden“, die geometrische Objekte zusammenhalten bzw. miteinander verbinden.

Schließlich sei noch erwähnt, daß es auch im Vektormodell Sinn macht, von Ebenen oder Schichten (Layern) zu sprechen: im Gegensatz zum Rastermodell hat man hier den Vorteil, daß beliebige neue Ebenen durch Selektion individueller Objekte gebildet werden können. So kann z.B. eine neue Ebene durch Selektion aller Objekte mit dem Objektschlüssel 2224 gebildet werden - es wäre dies eine Ebene ausschließlich für das Thema „Park, mit Einzelsymbolen“ (s. auch den in Kap. 6 vorgestellten Map Viewer).

Vor- und Nachteile des Vektormodelles

Hauptvorteile des Vektormodelles sind

Objekte: In vielen Kontexten benötigt man individuell ansprechbare Objekte. Im Rahmen einer objektorientierten Modellierung kann man auch das Verhalten der Objekte definieren. Individuelle Objekte können viele Attribute und auch Metadaten direkt im Objekt speichern (z.B. Erfasser eines Vermessungspunktes, Datum der Erfassung, etc.), wofür man im Rastermodell entweder separate Themen anlegen muß oder eine spezielle Attributdatenbank verwalten muß. Zudem gibt es dann das Problem der wechselseitigen Referenzen zwischen diesen Datenbeständen.

Speichersparsamkeit: Obwohl sich Rasterdaten manchmal effizient komprimieren lassen, benötigen Vektordaten in der Regel sehr viel weniger Speicher.

Genauigkeit: Objektkoordinaten lassen sich im Gegensatz zum Rastermodell mit beliebiger Genauigkeit angeben.

Einfache Implementierbarkeit bestimmter Operationen: Insbesondere Transformationen lassen sich viel einfacher als im Rastermodell durchführen (und zudem ohne Genauigkeitsverlust).

Dies ist mit folgenden *Nachteilen* zu kontrastieren:

Schwierige Implementierbarkeit best. Operationen: Die in der Praxis sehr wichtige Verschneidungs- bzw. Überlagerungsoperation (s.o.) ist im Vektormodell sehr kompliziert (s. z.B. den Algorithmus zur Bestimmung des Schnittpolygones zweier Polygone in [SDK96]), im Rastermodell hingegen trivial. Aufwendig sind auch *räumliche Selektionen*: hier sind spezielle räumliche Indizierungsverfahren vorzusehen (s.u.).

Keine Flächendeckung: Während Rastermodelle die gesamte Fläche des Interessenbereiches homogen repräsentieren, ist dies bei Vektormodellen nicht der Fall – es kann „weiße Flächen“ geben. Natürlich könnte man auch im Vektormodell eine Flächendeckung mit Polygonen

erreichen, so daß sich dieser oft in der Literatur zu findende Einwand wohl eher auf den Aufwand bezieht, der in beiden Modellen zur Erreichung der Flächendeckung getrieben werden muß.

Aquisitionskosten: Während im Rastermodell oftmals z.B. Multispektralbilder automatisch klassifiziert und somit in Ebenen bzw. Themen aufgeteilt werden können, ist dies für Vektordaten nicht der Fall. Hier werden in der Regel umfangreiche manuelle Digitalisierungsarbeiten notwendig. Aber auch die Daten vieler Rasterquellen müssen von einem menschlichen Bearbeiter klassifiziert werden.

3.3.4 Rastermodelle vs. Vektormodelle

Letztlich hängt die Wahl zugunsten eines Modelles immer vom beabsichtigten Anwendungskontext eines GIS ab, so daß ein anwendungsunabhängiger Vergleich von Plus- und Minuspunkten hier schwer fällt. So schreibt Bartelme:

> Rastermodelle sind für die Beschreibung flächiger Sachverhalte weit besser geeignet als Vektormodelle (die ihrerseits wieder eine Stärke für linienhafte Verbindungen aufweisen). Man bezeichnet Rastermodelle deshalb auch als *areale Modelle*, im Gegensatz zu den *linealen (Vektor-) Modellen*.

Will man Waldgebiete abbilden, so sind daher Rastermodelle besser geeignet als Vektormodelle. Will man hingegen Routenplanung oder Netzwerkanalysen durchführen, so wird ein solches Netzwerk natürlich besser durch topologisch strukturierte Vektordaten repräsentiert. Oftmals will man beides: in diesem Fall kann ein hybrides Modell helfen.

In gewisser Weise sind Vektor- und Rastermodelle diametral zueinander: was im Vektormodell einfach ist, ist im Rastermodell schwierig, und andersrum. Dies kommt auch im Gegensatz „Top-Down“ (Raster) vs. „Bottom-Up“ (Vektor) gut zum Ausdruck.

Spricht man in diesem Kontext von „(quasi-)analogischen Repräsentationen der Ebene“, so stellt sich die Frage, welches die direkt repräsentierten Eigenschaften und Relationen sind und auf welcher Stufe die Analogie (noch) vorliegt: im konzeptionell „eindimensionalen“ Speicher des Rechners bleibt von der Räumlichkeit der Modelle letzlich nicht mehr viel übrig – dies gilt sowohl für Raster- als auch Vektordaten. Scheinbar liegen Rastermodelle aber noch eine Stufe höher in der Analogizität als Vektormodelle. Hierfür spricht die Beobachtung, daß räumliche Selektionen im Rastermodell trivial, im Vektormodell jedoch aufwendig sind. Dies liegt natürlich daran, daß Position im Rastermodell in gewisser Weise *direkt* durch Position repräsentiert wird. Vektordaten werden daher oftmals mit einem zusätzlichen Gitter zur räumlichen Indizierung versehen (s.u.).

Interessant ist auch, daß Raster wiederum als spezielle Graphen und somit Vektormodelle betrachtet werden können. Während Rastermodelle die Position von Objekten in gewisser Weise direkt repräsentieren, ist dies für Graphen nicht unmittelbar der Fall, da die Position nicht primär als wesentlicher Bestandteil des Datenmodelles, sondern als Attribut betrachtet wird. Aber auch Graphen können bzgl. der Geometrie der Ebene als analogische Repräsentationen betrachtet werden, wenn man die Positionsattribute der Knoten als wesentlich ansieht. In jedem Fall repräsentieren Graphen die topologischen Eigenschaften direkt (also analogisch).

Die geführte Diskussion weckt Assoziationen mit der Konnektionismus-Debatte (s. auch [Fre91] für eine Diskussion weiterer „Zwei-Seiten-einer-Medaille“-Debatten).

3.3.5 Die Digitale Stadkarte des Hamburger Vermessungsamtes

Die Digitale Stadtkarte (DISK) wird vom Hamburger Vermessungsamt folgendermaßen beschrieben ([Fre96a]):

> Grob gefaßt kann man die DISK mit dem Begriff „Stadtplan“ oder „Stadtübersichtskarte“ umschreiben. Es handelt sich . . . um eine digitale Karte, die . . . in ca. 250 verschiedene Elementtypen strukturiert wurde, um möglichst vielen Anwendungsmöglichkeiten gerecht zu werden. . . . bildet bei der DISK ein kartographisch generalisiertes Modell den Kern des Datenbestandes, so daß zum Beispiel Straßen in ihrer Widmung entsprechenden einheitlichen Darstellung (Objektschlüssel, Breite, Ausgestaltung) nachgewiesen werden, ohne Rücksicht auf ihre tatsächliche Breite.

Der Basismaßstab der DISK ist 1 : 20 000 – sie wird entweder als Vektordaten im „DXF“ oder „SQD“-Format (je nach Datenvolumen von DM 2,- bis 35,- pro km^2) oder als TIFF-Raster (für DM 1,42 pro km^2) abgegeben.

Es handelt sich also im wesentlichen um grafische Daten, die für eine direkte Kartendarstellung gedacht sind. So findet man in der DISK u.a. Themen wie

- Straßenverkehr (klassifiziertes Netz inkl. Straßennamen und ausgewählten Hausnummern),
- Schienenverkehr,
- Flug- und Schiffsverkehr,
- Siedlungsflächen inkl. Darstellung öffentlicher Gebäude (teilw. durch Symbole),
- Nutzungsarten und Vegetation,
- Gewässer,
- Landes-, Bezirks, Stadt- und Ortsteilgrenzen sowie deren Bezeichnungen bis zum Baublock.

Diese Themen werden auf separaten Ebenen (Layern) gehalten: Eine Ebene (Layer) ist dabei eine thematische Zusammenfassung bestimmter Objekte. Kongruente geometrische Objekte finden sich auf vielen Ebenen – es wurde ja bereits erwähnt, daß die Aggregierung bzw. Bildung von Komplexen im wesentlichen durch thematische Gesichtspunkte bestimmt wird. So findet man Ebenen wie Gitternetz, Hauptverkehrstraßen 1 – 4, Sonstige Straßen 1 & 2, Parkplatzflächen, Bahnen, Siedlung, Vegetation, öffentliche Gebäude, etc.

In einem weiteren Schritt werden eine Reihe Ebenen zu *Tafeln* zusammengefasst – stellt man sich Ebenen als Farbauszüge vor, deren Überlagerung von Themen eine komplette Karte ergibt, so kann man sich Tafeln am ehesten als thematisch verschiedene Karten vorstellen, die unterschiedlichen Anwendungen dienen, aber eine gemeinsame Geometrie haben. Die wichtigste Tafel trägt den Namen „Grunddaten“: optional können nun die Tafeln „Verwaltungseinteilung“, „Buslinien des HVV“ und „Schrift und Symbole für den Maßstab 1 : 60 000“ hinzugekauft werden. Letztere Tafel ist notwendig, wenn man die DISK im kleinerem Maßstab 1 : 60 000 darstellen will (denn eigentlich ist sie für den Maßstab 1 : 20 000 gedacht): Symbole und Schrift können in der Regel nicht beliebig skaliert werden. Hierbei handelt es sich um eine typische *Generalisierungsproblematik* – Karten können nicht beliebig skaliert werden. So wird der *Verwendungsbereich* der DISK auch mit 1 : 10 000 bis 1 : 60 000 angegeben.

Wie bereits erwähnt, hat nun jedes Objekt in der DISK einen Objektschlüssel (OS), die im Objektschlüsselkatalog [Fre96b] dokumentiert sind.

Das SQD-Datenformat der DISK

Prinzipiell liegt den SQD-Daten der DISK eine dreistufige Hierarchie zugrunde (das SQD-Format wird in [Fre92a] beschrieben). Es wird u.a. zwischen Punkten (PG), Strecken (LI), Bögen (BO) und Splines (SN) sowie Flächen (FL) unterschieden. Strecken, Bögen und Splines sind spezielle Linien. Die Fläche ist ein Aggregat von einfachen Polygonen, also ein Komplexobjekt: die einzelnen

Polygone der Fläche werden durch hintereinandergereihte Linien (also eine Reihe von LI-, SN- und/oder BO-Objekten) repräsentiert. Eine Fläche kann natürlich auch ein einzelnes Polygon repräsentieren. Zudem findet man Textobjekte (TX) und Symbole (SY) in der DISK – alle anderen Elemente finden hier keine Erwähnung.

Jedes dieser SiCAD-Elemente wird nun durch einen „Datenblock“ beschrieben: ein Datenblock besteht aus einer Reihe von „Datensätzen“. Ein einzelner Datensatz beschreibt dabei in der Regel ein Attribut des entsp. Objektes und entspricht genau einer Zeile in der SQD-Datei. Ein Datenblock besteht aus mehreren Zeilen, also Datensätzen. Je nach Element enthält der dieses Element beschreibende Datenblock unterschiedliche Datensätze für die verschiedenen Attribute des Elementes.

Objekte werden nun hierarchisch gebildet, indem die entsp. Datenblöcke einfach hintereinander geschrieben werden: auf einen Datenblock LI (Strecke) müssen zwei Datenblöcke PG (Endpunkte) folgen. Die Baumebene kann durch die sog. Stufennummer im Kopf des Datenblocks abgelesen werden. Ein komplexes Objekt ist komplett, wenn der nächste Block wieder die Stufennummer 1 hat. Diese Regelung ist vor allem für die Flächen notwendig, die ja aus beliebig vielen Linien bzw. Polygonen bestehen können. Eine SQD-Datei beschreibt also einen Strom von Bäumen (Wald): die Datei besteht aus einem Kopf und einer Sequenz von Datenblöcken, wobei die Reihenfolge und die Stufennummer der Blöcke den Aufbau der Hierarchie regeln. Da es sich um eine strenge Hierarchie und keinen DAG handelt, kann das Format nicht als topologisch strukturiert bezeichnet werden. Gemeinsam genutzte geometrische Elemente sind mehrfach vorhanden. Hier taucht z.B. eine Linie, die einmal den Rand eines Teiches, ein anderesmal jedoch die innere Begrenzung eines Parkes beschreibt, doppelt auf, und zwar mit verschiedenen Objektschlüsseln.

Offensichtlich hätte diese Redundanz in den SQD-Dateien durch die Einführung von *Identifizierern* für geometrische Objekte verhindert werden können. Dies hätte aber auch die Zweiteilung in thematische und geometrische Objekte erfordert, da ein geometrisches Objekt ja in mehreren thematischen Rollen auftreten kann. Eine Referenz auf ein geometrisches Objekt hätte dann die Form (*thematische_Rolle*, *geom_id*) haben können.

Das Problem ist also, daß im SQD-Format die Thematik eines geometrischen Objektes einfach als Attribut definiert wurde – man hätte sowohl thematische als auch geometrische Datenblöcke und obige Referenzen benötigt. Die Daten sind aber auch deshalb nicht topologisch strukturiert, weil sich schneidende Linien (Kanten) zugelassen werden: so wurde z.B. mit VISCO ein Haus gefunden, dessen eine Ecke auf einer Hauptverkehrstraße liegt. Dies ist natürlich unkritisch, da es sich lediglich um *grafische Daten* handelt. In der DSGK hätte man hier jedoch ein ernstes Problem.

Letzlich sagt das externe Datenformat der SQD-Datei natürlich nichts über die intern im SiCAD (Siemens CAD) verwendeten Datenmodelle aus: hierbei könnte es sich durchaus um ein topologisch strukturiertes Vektordatenmodell handeln. So bezieht sich die hier geäußerte Kritik auch nur auf das Format der SQD-Dateien: schließlich kann die Topologie anhand der Geometrie der SQD-Daten wieder rekonstruiert werden. Dies ist für den Anwender jedoch relativ mühsam (s. Kap. 6). Schließlich sei hier noch ein kleiner Ausschnitt aus einer SQD-Datei abgedruckt:

```
**********************************************************************
**** SQD.A1-GDB.GRDA.A.001                                        ****
**** 01.07.93       08:32:16       AUFTRAGSNR=                    ****
**** XLU=74000      YLU=34000      XRO=76000      YRO=36000       ****
**********************************************************************
ETYP=LI1 STU=1 ENUM=24934/1B000004 EB=1 ST=3
OS 1111
**********************************************************************
ETYP=PG1 STU=2 ENUM=43/1B000004 EB=1 ST=3
X Z4512110000000000     74000.0000
Y Z4484D00000000000     34000.0000
PKZ S
PNR 0
OS 1119
```

```
**********************************************************************
ETYP=PG1 STU=2 ENUM=4234/1B000004 EB=1 ST=3
X Z4512110000000000     74000.0000
Y Z4480E80000000000     33000.0000
PKZ S
PNR 0
OS 1119
**********************************************************************
```

Als erstes erscheint der Kopf der Datei, dann ein Datenblock für ein Streckenelement (`LI`) gefolgt von den beiden Datenblöcken der beiden Endpunkte (`PG`). Der Kopf eines Datenblocks listet in der ersten Zeile den Elementtyp (`ETYP`), die Stufennummer (`STU`), die Elementnummer (`ENUM`) – wobei es sich nicht um einen Identifizierer handelt! – sowie die Ebene (`EB`) und die Strichstärke (`ST`) auf. Alsdann folgen eine Reihe von Datensätzen (bis zum nächsten Block), die weitere Attribute des Objektes beschreiben. Der Objektschlüssel (`OS`) der Linie ist 1111, was die Linie als „GK-Gitterline 2km“ identifiziert. Die Punkte tragen die Thematik 1119, was einen „Gitterpunkt für das Einpassen von Vorlagen“ bezeichnet. Diese Linie beschreibt also einen Teil des Koordinatengitters der Karte. Die Koordinaten der Punkte (`X Y`) sind im Gauß-Krüger-Koordinatensystem und sowohl hexadezimal als auch dezimal angegeben (in Metern).

3.3.6 Standardformate für Geodaten

Hier sollen nur einige Begriffe erwähnt werden (s. jedoch [Pol97]): Standardformate für Geodaten sind u.a. „Digital Elevation Models (DEMs)“, „Digital Line Graphs (DLGs)“, „Geographic Information Retrieval and Analysis System (GIRAS)“-Daten, das „Vector Product Format (VPF)“ als Teil des „Digital Geographic Information Exchange Standard (DIGEST)“, der „Spatial Data Transfer Standard (SDTS)“ und auch „Dual Independent Map Encoding (DIME)“-Daten. In Deutschland soll in Zukunft vor allem das „Austauschformat des Amtlichen Topographischen-Kartographischen Informationssystems der Bundesrepublik Deutschland (ATKIS-EDBS)“ eingesetzt werden. Schließlich sind noch Quasi-Standards wie „DXF (AutoCAD)“, „SQD (SiCAD)“, „IGES (INTERGRAPH)“ und „GENERATE (ARC/INFO)“ zu erwähnen.

Einigen obiger Formate liegt das Rastermodell, anderen das Vektormodell zugrunde. Die aufgelisteten Datenformate sind vor allem zum Zweck des Austausches von Geodaten von Interesse („Marktplatz Geoinformation“). Einige Standards können sowohl Raster- als auch Vektordaten behandeln: ein solches Datenmodell wird hybride genannt; man spricht auch von *hybriden GIS.*

3.4 Räumliche Datenorganisation

Nachdem nun die beiden grundsätzlichen konzeptionellen Datenmodelle für GIS diskutiert wurden, sollen in diesem Kapitel Realisierungsmöglichkeiten (in Form konkreter Datenstrukturen) diskutiert werden. Dabei beschränke ich mich auf Vektormodelle. Insbesondere sollen auch *räumliche Indizes* kurz vorgestellt werden.

Als Implementierungsgrundlage für vektorbasierte GIS wird meist eine *räumliche Datenbank (Spatial Database)* vorgeschlagen, während für rasterbasierte GIS u.a. *Bilddatenbanken (Image Databases)* verwendet werden könnten ([Güt94]).

3.4.1 Relationale Datenbanken und GIS

In der Vergangheit wurden diverse Versuche unternommen, GIS ausschließlich unter Verwendung relationaler Datenbanktechnologie zu implementieren (s. Abb. 3.7(a)). Die Standard-Datenbank-

sprache für relationale Datenbanken ist SQL (s. z.B. [Dat95, Chap. 8]):[7] in den Bereich der Sprache fallen nicht nur Datenbankabfrage (Database Query Language, DQL), sondern auch Datenbankmanipulation (Database Manipulation Language, DML) und -definition (Database Definition Language, DDL). Das konzeptionelle Datenmodell eines vektorbasierten GIS könnte im relationalen Datenmodell durch Tabellen implementiert werden – dabei könnte man wie folgt vorgehen: Eine Punkt-Tabelle hat die Spalten `(Id,x,y)` (`Id` steht für *Identifier*), eine Linien-Tabelle `(Id,p1-Id,p2-Id)`, eine Polygon-Tabelle `(Id,Segment-Id)`.

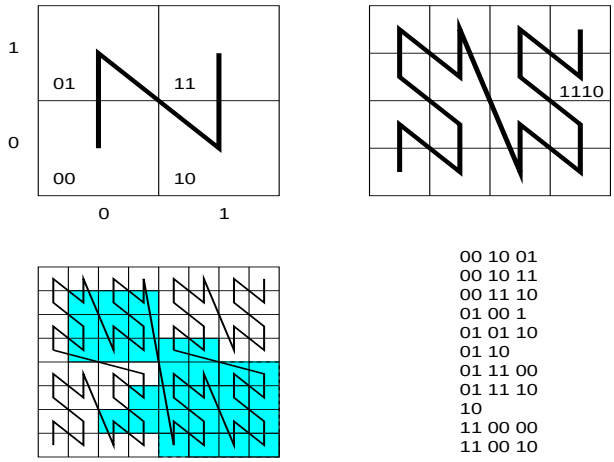


Abbildung 3.6: Eindimensionale Kodierung durch Z-Ordnung

Diese Vorgehen ist jedoch inzwischen aus einer Reihe von Gründen überholt:

- **Fehlende Abstraktionen:** Punkte, Linien und Polygone sollten auf *Typebene* vorhanden sein, wie z.B. Zahlen und Zeichenketten ([GS93, Sch96, Güt94]). Im relationalen Modell müssen diese räumlichen Datentypen jedoch durch Tabellen *implementiert* werden – Polygone mit *variabler Segmenteanzahl* müssen als Relation (wie oben) definiert werden, da keine Schemata bzw. Tabellen mit variabler (dynamischer) Spaltenanzahl definiert werden können. Komplexe Objekte können nicht mehr durch ein einziges Tupel in der Relation repräsentiert werden - diese Zersplitterung beeinträchtigt z.B. die Performanz (selbst ein einzelnes Polygon muß erst „zusammengesucht“ werden).
- **Das relationale Modell per se ist unzureichend** ([Ege92]).
- **Fehlende SQL-Unterstützung:** SQL kann ohne Erweiterungen nicht sinnvoll verwendet werden – entsp. Anfragen werden dann unnötig kompliziert oder können aufgrund der Beschränkungen des relationalen Modelles nicht formuliert werden. Zudem ist die *fehlende grafisch-interaktive Unterstützung* ein wesentliches Manko ([Ege92]).
- **Mangelhafte Performanz:** die Indizierungsverfahren relationaler Datenbanksysteme greifen nicht, da die räumlichen Attribute der Objekte nicht als Selektionskriterien (Schlüssel) verwendet werden. Die räumlichen Eigenschaften der Objekte werden über die ganze Datenbank verstreut und können somit nicht zur effizienten Indizierung benutzt werden ([Güt94]). Eingeschränkt können hier jedoch wieder spezielle (und künstliche eindimensionale) Kodierungen wie die Z-Ordnung (s. Abb. 3.6, auch N- oder Peano-Orndung genannt) helfen – durch diese Kodierung bleibt die Nachbarschaft von räumlichen Objekten teilweise erhalten und durch B-Bäume indizierbar.

Daher ging man relativ schnell dazu über, GIS durch *zwei separate Komponenten* zu implementieren (s. Abb. 3.7(b)): so verwendet z.B. das kommerzielle „ARC/INFO“-System eine relationale

[7] Daher gehe ich davon aus, daß der Leser mit SQL vertraut ist.

Datenbank („INFO“) für thematische Daten (s.u.), sowie eine spezialisierte Datenbank für die räumlichen Daten („ARC“). Diese Zweiteilung verwenden die meisten heute kommerziell erhältlichen GIS. Während diese Vorgehen den Vorteil bietet, nun den räumlichen Charakter der Objekte adäquat behandeln und verwalten zu können, ist jedoch die Anfragebearbeitung umso komplizierter, da verschiedene Teile der Anfrage von verschiedenen Komponenten zu bearbeiten sind. Hier gibt es z.B. Probleme mit der Optimierbarkeit ([Güt94]).

Schließlich wurde SQL für diese hybride Architektur entsp. erweitert, so daß nun räumliche und nicht-räumliche (thematische) Eigenschaften von Objekten einheitlich behandelt bzw. angefragt werden können. Unter anderem finden sich nun neben den bekannten numerischen Operationen etc. auch räumliche Operationen (wie geometrische Berechnungen) und Prädikate in der `WHERE`-Klausel.

So würde die Anfrage „Finde alle Städte mit einer Bevölkerung über 5000 innerhalb einer Entfernung von 200 km um die Zugspitze“ in „GEOQL“ (hierbei handelt es sich um einen erweiterten SQL-Dialekt) wie folgt formuliert werden:

```
SELECT  CITY.Name
FROM    CITY, MOUNTAIN
WHERE   MOUNTAIN.Name = "Zugspitze" and
        CITY.POPULATION >= 5000 and
        CITY within 200 km of MOUNTAIN.
```

Offensichlicht werden hier in der `WHERE`-Klausel sowohl räumliche als auch nicht-räumliche Bedingungen gestellt. „GEOQL“ stellt in der `WHERE`-Klausel folgende Prädikate für räumliche Relationen zur Verfügung: *intersects, adjacent, joins, ends-at, contains, situated-at, within, closest, furthest.* Diese Prädikate unterstützen die sog. *räumliche Verknüpfung (Spatial Join)* zwischen „Tabellen“: die Tabellen `CITY` und `MOUNTAIN` werden durch die räumliche Bedingung `within 200 km of` miteinander verknüpft. Die effiziente Unterstützung des *Spatial Join* ist ein wesentliches Charakteristikum *räumlicher Datenbanken.* Selbiges gilt für die *räumliche Selektion (Spatial Selection)*: hier gibt der Benutzer in der Regel eine räumliche Konstante als Bezugsobjekt vor. Diese Konstante kann z.B. in Form eines auf einer dargestellten Karte mit der Maus interaktiv erzeugten Rechteckes als auch durch das Objekt „Zugspitze“ in der `MOUNTAIN`-„Tabelle“ vorgegeben werden.

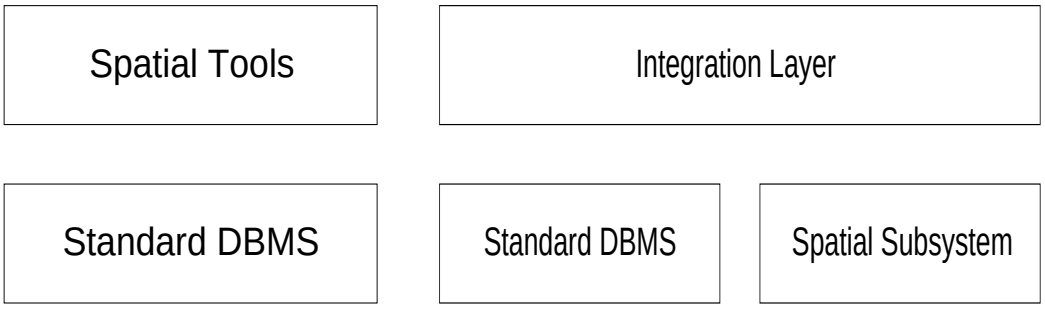
Die unterstützten räumlichen Objekte in „GEOQL“ sind Punkte, Linien und Gebiete: auch hier wird die Zweiteilung in Form von externen Dateien für räumliche Daten und relationaler Datenbank für thematische Daten aufrechterhalten. Verweise zwischen diesen werden durch gemeinsame Identifizierer verwaltet.

3.4.2 Räumliche Datenbanken und GIS

Als Basismaschine für ein GIS wird meist eine *räumliche Datenbank (Spatial Database)* vorgesehen, ebenso, wie eine Datenbank meist Grundlage konventioneller Informationssysteme ist (s. obige Definition). Laut Güting wird eine *räumliche Datenbank* folgendermaßen charakterisiert ([Güt94]):

1. A spatial database system is a database system.
2. It offers spatial data types (SDTs) in its data model and query language.
3. It supports spatial data types in its implementation, providing at least spatial indexing and efficent algorithms for spatial join.

Somit müssen also lt. Güting mindestens drei Anforderungen erfüllt sein: sowohl im Datenmodell als auch in der Anfragesprache einer räumlichen Datenbank müssen räumliche Datentypen (Spatial Data Types, SDTs) vorhanden sein – hierunter sind generische Typen wie Punkt, Linie, Polygon etc. zu verstehen. Der effiziente Umgang mit diesen SDTs muß durch die Implementation (bzw.



(a) Historische Architektur (b) Heutige duale Architektur

Abbildung 3.7: Architekturmodelle räumlicher Datenbanken (reproduziert nach [Güt94])

Architektur) unterstützt werden: Hierzu ist insbesondere ein *räumliches Indizierungsverfahren (Spatial Index)* vorzusehen, mit dessen Hilfe *räumliche Selektionen* auf dem Datenbestand sehr viel effizienter vorgenommen werden als durch inviduelles Betrachten jedes einzelnen gespeicherten Objektes. Die *räumliche Verknüpfung (Spatial Join)* zwischen Mengen („Tabellen“) räumlicher Datenobjekte soll ebenfalls in einer Weise unterstützt werden, die wesentlich effizienter als das Filtern des kartesischen Produktes dieser beiden Mengen ist.

3.4.3 Räumliche Indizes

Güting schreibt in [Güt94]:

> The main purpose of spatial indexing is to support spatial selection, that is, to retrieve from a large set of spatial objets (objects with an SDT *[Spatial Data Type]* attribute) those in some particular relationship with a query SDT. A spatial indexing method organizes space and the objects in it in some way so that only parts of the space and a subset of the objects need to be considered to answer such a query.

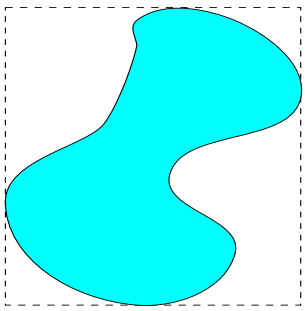


Abbildung 3.8: Kleinstes umschließendes Rechteck (MBR)

Räumliche Indizierung unterstützt also das Auffinden von räumlichen Objekten anhand bestimmter räumlicher Eigenschaften und Beziehungen zu anderen räumlichen Objekten. Das Verfahren soll wesentlich effizienter sein, als es durch die dedizierte Untersuchung jedes einzelnen Objektes im Datenbestand geschehen könnte. Durch einen räumlichen Index wird in der Regel auch unmittelbar die *räumliche Verknüpfung (Spatial Join)* unterstützt.

Insbesondere *Approximationen räumlicher Objekte* spielen eine große Rolle: so z.B. das *kleinste umschließende Rechteck (Minimum Bounding Rectangle, MBR)* (s. Abb. 3.8). Das MBR spielt im wesentlichen die Rolle eines *räumlichen Schlüssels (Spatial Keys)*, der zur Selektion verwendet

werden kann. Während MBRs Schlüssel für räumlich ausgedehnte Objekte sind, müssen jedoch auch Punkte effizient indiziert werden können: man unterscheidet daher zwischen räumlichen Indizes für Punkte und Rechtecke. Für Punkte ist natürlich die *Position* der wesentliche räumliche Schlüssel.

Anhand eines vorgegebenen Punktes sollen z.B. alle Punkte im Datenbestand zurückgeliefert werden, die

- die gleiche Position haben wie dieser, oder
- einen Abstand kleiner als r Meter zu diesem Punkt haben.

Zusätzlich ist die sog. *Bereichsanfrage (Range Query)* von großer Wichtigkeit:

- Liefere alle Punkte, die innerhalb eines vorgebenen Rechteckes liegen.

Für Rechtecke sollen oftmals alle Rechtecke zurückgegeben werden, die

- das Rechteck schneiden oder
- im vorgegebenen Rechteck enthalten sind.

Eine fundamentale Strategie zur Unterstützung räumlicher Selektionen anhand von Indexstrukturen ist die *Filtere und Verfeinere (Filter and Refine)*-Strategie: zunächst wird ein Filterschritt anhand von Approximationen (z.B. MBRs für räumliche Objekte) eine Reihe von *Kandidaten* liefern – hier wird der Datenbestand zunächst also *grob gefiltert* (aufgrund der Approximationen). Dieser Schritt muß sehr effizient durchgeführt werden können – Ziel der Indexstrukturen ist im wesentlichen die effiziente Unterstützung des Filterschrittes. Im zweiten Schritt (*Refine*) werden nun die so erhaltenen Kandidaten ein weiteres mal gefiltert, diesmal jedoch ohne Approximationen. Die in der Selektion gestellte räumliche Bedingung muß für jedes Objekt individuell anhand der Geometrie überprüft werden, und die übrigbleibenden Objekte werden als Ergebnis zurückgegeben.

Offensichtlich ist das Verfahren um so effizienter, je weniger Kandidaten der erste Schritt liefert: liefert der Filter stets alle (oder einen großen Prozentsatz aller) Datenbankobjekte als Kandidaten, so ist kein Effizienzgewinn möglich.

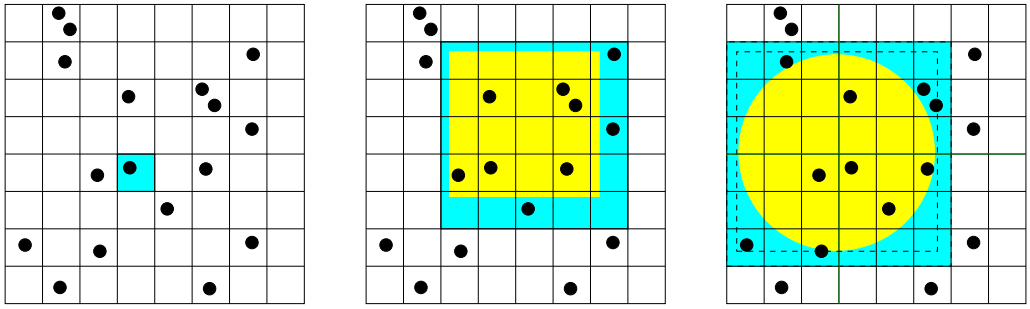
In der Literatur findet man eine überwältigende Fülle von Indexstrukturen (für Punkte: Point Quad Trees, KD-Trees, KDB-Trees, Gridfiles, LSD-Trees, hB-Trees etc.; für Rechtecke: R-Trees, R^+-Trees, R^*-Trees, Region Quad Trees, Cell-Trees, etc., s. z.B. [Bar95, Güt94]).

Indexstrukturen für Punkte

Indexstrukturen für Punkte werden im Bereich konventioneller Datenbankmanagementsysteme schon seit langem verwendet: schließlich läßt sich ein Tupel $(a_1, a_2, \cdots, a_n)$ einer Relation (über den entsp. Attributen) als n-dimensionaler Punkt im Produktraum der Attribute auffassen. Die *Gridfile*-Methode (s.u.) wurde ursprünglich zur Indizierung anhand von Attributen (Schlüsseln) in relationalen Datenbanken genutzt.

Gitter: Das *regelmäßige Gitter* ist die einfachste räumliche Indexstruktur für Punkte: die einzelnen Gitterzellen werden z.B. durch Seiten (Pages) eines Speichermediums implementiert (auch Buckets genannt). Ein Punkt wird nun einfach anhand seiner Position in die entsp. Seite eingetragen (Abb. 3.9).

Für eine Punktselektion („Finde alle Punkte, die die Position (x, y) haben“, s. Abb. 3.9(a)) kann nun (x, y) als Schlüssel verwendet werden: in einem ersten Schritt wird also die relevante Gitterzelle ermittelt (Filter), und dann kann innerhalb dieser lokal gesucht werden (Refine). Wenn



(a) Punktselektion (b) Bereichsselektion (c) Radiusselektion

Abbildung 3.9: Punktindizierung mit einem Gitter

das Gitter $n_x \times n_y$ Maschen hat und einen Bereich von $l_x \times l_y$ Metern repräsentiert, so ist der Index in das Gitter für den Punkt (x, y) natürlich $(\lfloor \frac{x \cdot n_x}{l_x} \rfloor, \lfloor \frac{y \cdot n_y}{l_y} \rfloor)$. Hierbei handelt es sich um eine zweidimensionale *Hashfunktion*.

Aber auch eine Bereichsanfrage („Finde alle Punkte, die innerhalb des Rechteckes $((x_1, y_1), (x_2, y_2))$ liegen“, s. Abb. 3.9(b)) kann einfach beantwortet werden: hierzu sind lediglich alle Kandidaten der Gitterzellen zu untersuchen, die mit dem Rechteck überlappen (also einen nichtleeren Schnitt haben).

Insbesondere kann ein Gitter (wenn man Mehrfachspeicherungen von Objekten in verschiedenen Zellen zuläßt) auch für räumlich ausgedehnte Objekte verwendet werden. Weitere Details werden in Kap. 6 vorgestellt, wo die Implementation eines primitiven räumlichen Indexes in Form eines Gitters dargestellt wird.

Die *Feinheit des Gitters* muß den jeweiligen Gegebenheiten individuell angepaßt werden – Baumstrukturen bieten hier größere Dynamik, s.u. Ein Gitter ist nur dann effizient verwendbar, wenn die Punkte *homogen verteilt* sind. Für stark inhomogene und/oder sehr dynamische Datenbestände sind Baumstrukturen wesentlich besser geeignet.

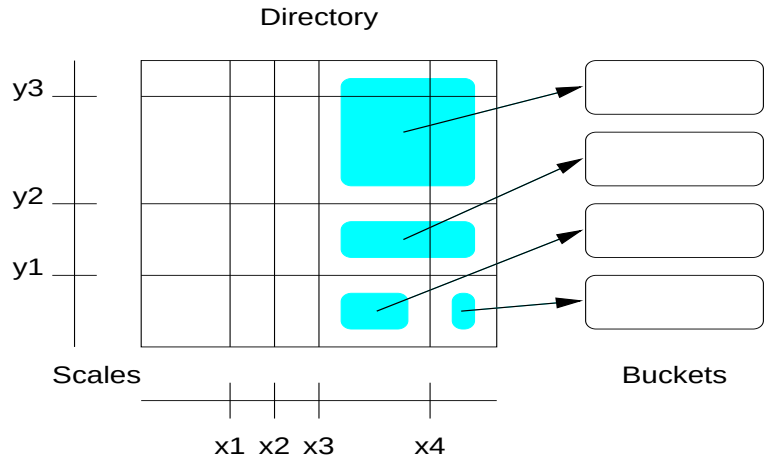


Abbildung 3.10: Gridfile (reproduziert nach [Güt94])

Gridfile: Eine Verallgemeinerung des regelmäßigen Gitters stellt das sog. *Gridfile* (Abb. 3.10) dar: hierbei handelt es sich um ein unregelmäßiges (in der Regel mehrdimensionales) Gitter. Man unterscheidet zwischen dem

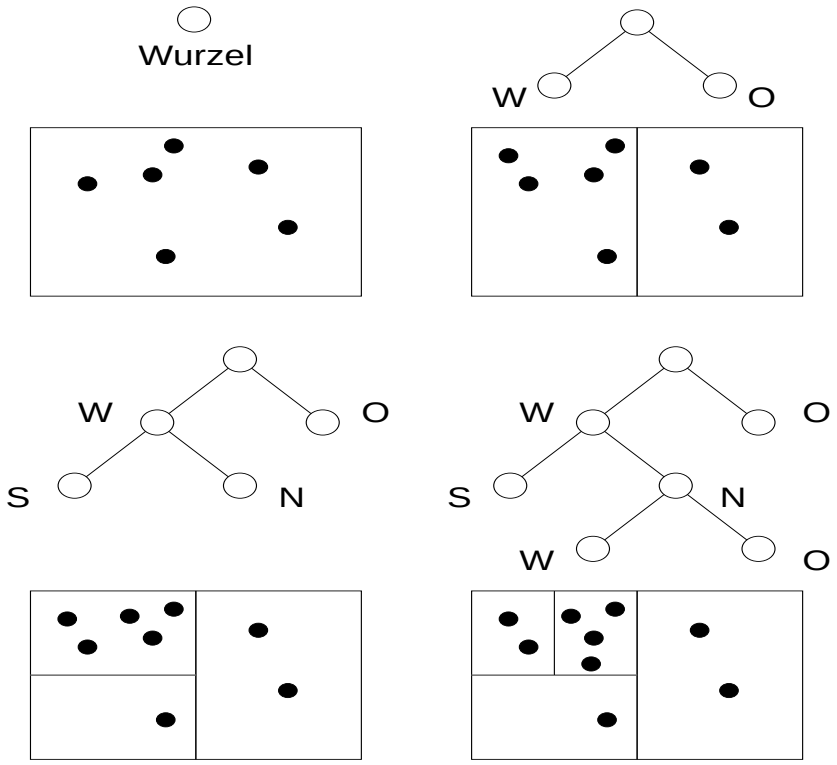


Abbildung 3.11: KD-Baum (reproduziert nach [Bar95])

- Verzeichnis (Directory), den
- Seiten (Buckets) und den
- Skalen (Scales).

Das Verzeichnis wird einfach als n-dimensionales Feld (Array) implementiert: es enthält Zeiger (Pointer) auf die entsp. Seiten des Speichermediums (z.B. Plattenblöcke). Durch Vergleiche mit den Skalenintervallen (hier bietet sich *binäres Suchen* an) können nun über das Verzeichnis die entsp. Seiten gefunden werden. Wiederum muß dann lokal innerhalb dieser gesucht werden.

Ähnlich würde man natürlich auch obiges Gitter implementieren – lediglich die Skalen wären dann überflüssig, da das einfache Gitter ja homogene Auflösung hat. Im Unterschied zum Gitter kann der Index in das Verzeichnis eines Gridfiles nicht mehr einfach errechnet werden – hierfür ist nun z.B. binäres Suchen in den Skalen notwendig.

Die *Skalen bzw. Achsen* werden in relationalen Datenbanksystemen als Wertebereiche für die entsp. Attribute angesehen. Pro Attribut gibt es eine Skala. Hier spricht man auch vom n-dimensionalen *Datenraum*, der durch die Attribute aufgespannt wird.

Baumstrukturen und „EXCELL"-Adressierung: Der raumbezogene Zugriff in *Baumstrukturen* ist in der Regel aufwendiger als in Gitterstrukturen, da die direkte Analogie zur Ebene verloren geht: dafür werden nun aber die einzelnen Seiten (Pages oder Buckets) des Speichermediums maximal effizient genutzt. Wenig oder nicht gefüllte Seiten kommen nicht mehr vor (Ausnahme: Blätter des Baumes). Die Seiten entsprechen nun einzelnen Knoten (und Blättern) des Baumes.

Kriterium für die Aufspaltung des Baumes in Nachfolgezweige ist in der Regel die *Kapazität des Knotens:* so wird die Aufspaltung eines *KD-Baumes* dann gestoppt, wenn jeder Knoten höchstens

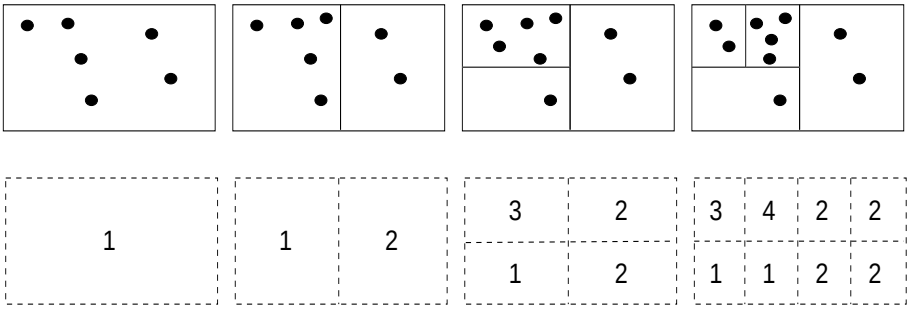


Abbildung 3.12: EXCELL-Adressierung (reproduziert nach [Bar95])

n Punkte enthält (Abb. 3.11). Diese Aufspaltung entspricht einer Partitionierung der Ebene: die Aufspaltung der Ebene geschieht jeweils abwechselnd horizontal und vertikal.

Die *„EXCELL (Extendible Cell Structure)"-Adressierung* erlaubt es nun, den Vorteil der einfachen Adressierbarkeit des Gitters mit der effizienten und dynamischen Speicherung in Baumstrukturen zu verbinden. Hierzu versieht man die beispielsweise durch einen KD-Baum partitionierte Ebene erneut mit einem in vertikaler und horizontaler Richtung jeweils einheitlichem Gitter: dieses Gitter speichert nun wiederum Verweise auf die Knoten des KD-Baumes. Die Maschenweite des Gitters ist nicht fest, denn das Gitter wächst dynamisch mit dem KD-Baum (s. Abb. 3.12). Die Einführung einer zusätzlichen Indirektion (in Form des überlagerten Gitters) erlaubt es also, die Daten zum einen effizient und dynamisch in Baumstrukturen zu verwalten, zum anderen aber auch effizient raumbezogen zu selektieren (aufgrund der Analogie des Gitters zur Ebene).

Indexstrukturen für Rechtecke

Rechtecke sind deshalb wichtig, weil die MBRs ausgedehnter Objekte als Schlüssel für die Indizes verwendet werden, die ihrerseits anhand der MBRs der so indizierten Objekte aufgebaut werden.

Denkt man wieder an ein Gitter, so stellt sich die Frage, in welcher Zelle ein ausgedehntes (mehrere Zellen überlappendes) Objekt gespeichert werden soll. In der Literatur findet man u.a. folgende Vorschläge:

- Mehrfachspeicherung (ein Objekt wird in jeder überlappenden Zelle gespeichert)
- Reduktion auf Zentroide (der Schwerpunkt des Objektes bestimmt die speichernde Zelle)
- Transformation in höhere Dimensionen (s.u.).

Prinzipiell läßt sich ein *zweidimensionales Rechteck* als *vierdimensionaler Punkt* darstellten, so daß obige Indexstrukturen für Punkte auch zur Indizierung von Rechtecken verwendet werden könnten. So kann z.B. die Bedingung, daß sich zwei Rechtecke überlappen sollen, auch als entsp. Bedingung zwischen den vierdimensionalen Punkten dieser Rechtecke formuliert werden. In der Regel führt diese Vorgehen jedoch zu einer stark inhomogenen Verteilung der vierdimensionalen Punkte und zu einer erheblichen Verkomplizierung aller Operationen.

Hier sollen nur die *R-Bäume* kurze Erwähnung finden: bei ihnen handelt es sich um eine mehrdimensionale Erweiterung der klassischen *B-Bäume.* Ein R-Baum nutzt die MBRs der durch ihn verwalteten Objekte zur Bildung einer Hierarchie von Rechtecken („R" steht für Rechteck, s. Abb. 3.13).

Die *Blätter* des Baumes entsprechen dabei direkt den MBRs der durch den R-Baum indizierten Objekte, und da es sich um einen *höhenbalancierten* Baum handelt, liegen alle Blätter auf gleicher Ebene. Ein *Zwischenknoten* (der beliebig viele Nachfolger haben darf) repräsentiert ebenfalls ein Rechteck, nämlich das MBR aller Rechtecke der Nachfolgeknoten dieses Knotens. Der R-Baum

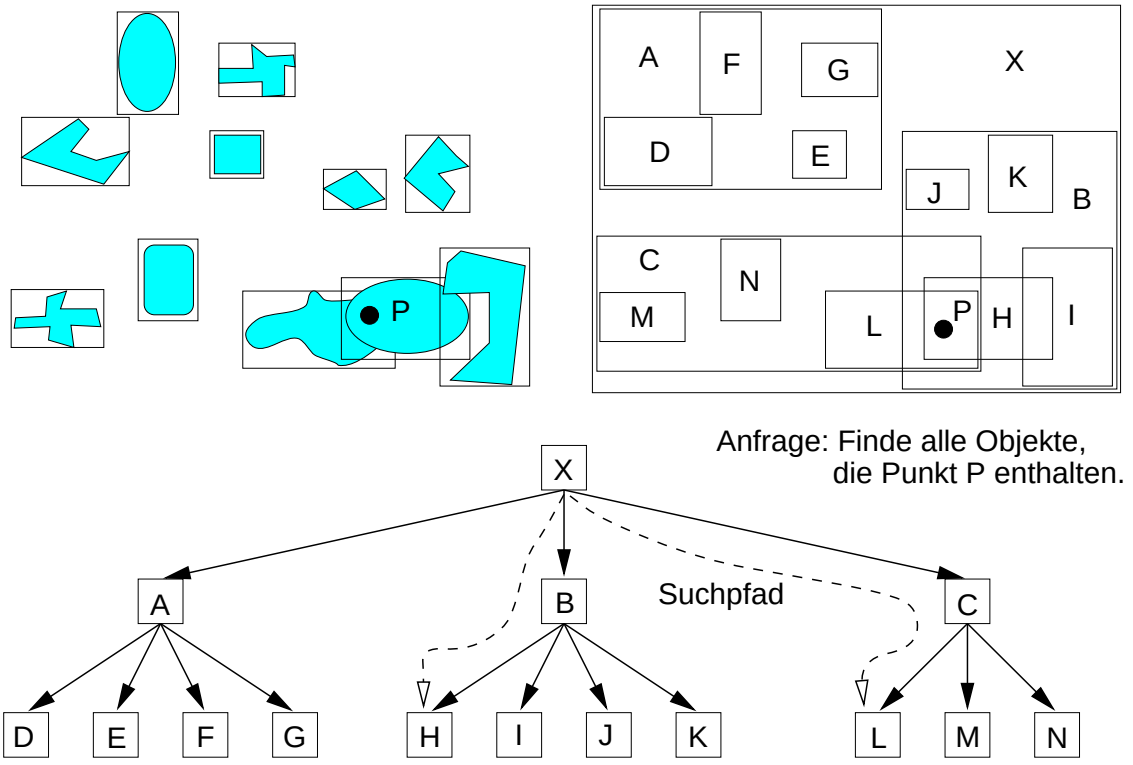


Abbildung 3.13: Räumliche Indizierung mit einem R-Baum

wird nun so aufgebaut und verwaltet (hier sind die Operationen „Einfügen“ und „Löschen“ zu erwähnen), daß möglichst zwei Bedingungen erfüllt werden:

1. Minimale Überlappungen der Rechtecke
2. Minimale Tiefe des Baumes

Räumliche Selektionen werden nun durch *Travesierung des R-Baumes* unterstützt, wobei anhand der Rechtecke der Knoten und der entsp. Selektionsbedingung entschieden wird, ob ein in einem Zwischenknoten beginnender Ast des Baumes weiter abgestiegen werden soll oder nicht (Filter). Hat man schließlich einen Blattknoten vorliegen, so kann das entsp. Objekt dem Verfeinerungsschritt (Refine) unterzogen werden.

Soll z.B. die Anfrage in Abb. 3.13 „Finde alle Objekte, in denen der vorgegebene Punkt P enthalten ist“ bearbeitet werden, so wird der R-Baum von der Wurzel X aus traversiert, wobei von einem Zwischenknoten aus dann weiter abgestiegen wird, wenn der entsp. Punkt im Rechteck des Zwischenknotens enthalten ist. Da die Rechtecke sich überlappen können, müssen evtl. mehrere Äste des Baumes traversiert werden, im Bsp. B und C. Schließlich landet man bei den Blättern: alsdann werden all die Objekte zurückgegeben, die den Feintest bestehen (Objekte H und L). Hier muß die konkrete Geometrie der Objekte vollständig berücksichtigt werden.

Dieses Verfahren ist natürlich um so effizienter, je besser die beiden obigen Bedingungen eingehalten werden. Es sollte klar sein, daß mit R-Bäumen nicht nur Punktabfragen, sondern u.a. ebenso Bereichs- und Überlappungsselektionen durchgeführt werden können. In R^+*-Bäumen* werden *überlappende Rechtecke* ausgeschlossen ([RFS88]): wenn notwendig werden die indizierten Objekte an den Grenzen der Rechtecke entsprechend abgeschnitten (Clipping). Dann muß bei der Suche nicht mehr verzweigt werden.

Kapitel 4

Visuelle räumliche Anfragesprachen (Visual Spatial Query Languages)

Ceci n'est pas une pipe
frei nach René Magritte,
„Die Melodie und das Lied", 1964

In diesem Kapitel möchte ich zunächst die Relevanz für die Beschäftigung mit *visuellen räumlichen Anfragesprachen (Visual Spatial Query Languages, VSQLs)* darstellen. Dazu wird der Begriff der *visuellen Anfragesprache (Visual Query Language, VQL)* beleuchtet; dann beschränke ich mich jedoch auf eine spezielle Teilklasse, die visuellen *räumlichen* Anfragesprachen. Eine Diskussion von Vor- und Nachteilen derartiger Sprachen bildet den Hauptteil dieses Kapitels. Schließlich folgt eine Vorstellung und Vergleich einiger in der Literatur zu findender VSQLs.

4.1 Definitionen

Die *visuellen Anfragesprachen (Visual Query Languages, VQLs)* bilden eine spezielle Teilklasse der visuellen Sprachen (Visual Languages, VLs, Def. s. Kap. 2). Der Begriff läßt vermuten, daß es sich bei visuellen Anfragesprachen um visuelle Sprachen handelt, die zur Informationswiedergewinnung (Information Retrieval) aus Informationssystemen (Def. s. Kap. 3) oder Befragung von Datenbanken geeignet sind.

Eine Definition für den Begriff *visuelle Anfragesprache* ist Catarci et al. ([CCLB97]) entnommen, wobei hier der Fokus zunächst auf *visuellen Anfragesystemen* liegt:

> Visual Query Systems (VQS) are defined as systems for querying databases that use a visual representation to depict the domain of interest and express related requests. ... VQS provide both a language to express the queries in a visual format and a variaty of functionalities to faciliate user-system interaction. ... VQLs are the languages implemented by VQSs. VQLs can also be seen as a particular subclass (aimed at extracting information from databases) of the more general class of visual languages ... visual query languages are a subclass of visual programming languages ...

Eine visuelle Anfragesprache dient somit zur Kommunikation mit einem visuellen Anfragesystem. Ein visuelles Anfragesystem dient der Befragung einer Datenbank, wobei eine visuelle Repräsen-

tation zur Darstellung des Interessenbereiches und diesbezüglicher Anfragen verwendet werden soll.

Während Catarci et al. VQLs als spezielle visuelle Programmiersprachen (VPLs) ansehen, vertrete ich diese Sichtweise nicht, da VQLs per se nicht zur Formulierung von Algorithmen dienen, sondern zur Befragung einer Datenbank bzw. zur Informationswiedergewinnung aus einem Informationssystem.[1]

Während in obiger Definition noch nicht ausgesagt wird, daß der Datenbestand der Datenbank inhärent räumlichen Charakter haben muß, wird genau diese Einschränkung für eine spezielle Teilklasse visueller Anfragesprachen gemacht, nämlich die *visuellen räumlichen Anfragesprachen (VSQLs)*. Das Attribut „räumlich" charakterisiert also den Datenbestand der Datenbank bzw. des Informationssystemes, während „visuell" sich auf die grafischen Erscheinungen der Sprachelemente bezieht (im Sinne einer visuellen Sprache bzw. eines visuellen Systemes).

Räumliche Anfragesprachen (Spatial Query Languages) für GIS wurden in Kap. 3 kurz angesprochen – dort wurde u.a. der erweiterte SQL-Dialekt „GEOQL" vorgestellt. Nahezu alle heutzutage in der Praxis benutzen Anfragesprachen sind konventionelle textuelle Sprachen (meist erweiterte SQL-Dialekte) und somit keine visuellen Sprachen.

Querbezüge lassen sich auch zu *Bilddatenbanken* (Image Databases) herstellen. Hier ist man meist an einem *inhaltsorientierten Bildzugriff (Content Based Image Retrieval)* interessiert (s. [BPS94, HK92, GR95]). Auch für *Multimediadatenbanken* existieren schon Vorschläge für VQLs.

4.2 Erwartungen und Probleme

Als Rechtfertigung für die Beschäftigung mit visuellen Anfragesprachen lassen sich alle bereits für die visuellen Sprachen dargestellten Gründe anführen (s. Kap. 2). Besondere Relevanz ergibt sich aber für Datenbanken und Informationssysteme: Sicherlich spielen Informationssysteme eine zunehmend wichtigere Rolle in unserer Gesellschaft, so daß die Gruppe der Benutzer zwangsweise wachsen wird und muß. Insbesondere ist auch an die „Wucherung" des World Wide Web (WWW) mit seinen riesigen Informationsbeständen zu denken. Informationsauffindung im WWW ist zur Zeit wohl eines der aktuellsten Themen in der Informatik. Der Prozentsatz von EDV-Spezialisten und Informatikern in der Gruppe der Benutzer derartiger Informationssysteme wird zunehmend geringer, so daß entsprechende Maßnahmen ergriffen werden müssen, um eine adäquate Nutzung dieser Informationsmengen für die „Informationsgesellschaft" zu ermöglichen ([Pos95]).

Da ein Bild in der Regel (bei geeigneter Interpretation) sehr viel Information enthält und das menschliche visuelle System von der Evolution für die Verarbeitung mehrdimensionaler Zusammenhänge optimiert wurde, besteht hier Hoffnung, auf der einen Seite sehr komplexe Anfragen auf eine dem Menschen entgegenkommende kompakte Art und Weise darstellen zu können, und auf der anderen Seite die hohe Bandbreite des menschlichen visuellen Systemes zur *Orientierung* in diesen umfangreichen Datenbeständen nutzen zu können. Daten müssen in zunehmend komplexeren Anwendungen verknüpft und ausgewertet werden, so daß auch zunehmend komplexere Anfragen notwendig werden.

Die im folgenden diskutierten Plus- und Minuspunkte sind nicht disjunkt, sondern bedingen sich wechselseitig auf komplexe Art und Weise. Von vielen Forschern wird die Auffassung vertreten, daß in der Klasse der Anfragesprachen die geeignetsten Kandidaten für visuelle Sprachen zu finden sind.

[1] Letztendlich wird hier natürlich ein die Anfrage bearbeitender Algorithmus visuell und möglichst deklarativ formuliert – pragmatisch ist aber nicht die Algorithmenformulierung, sondern das Anfrageergebnis von Interesse. Natürlich kann auch eine allgemeine Allzweck-Programmiersprache (etc. „C") zur operationalen Datenbankabfrage verwendet werden.

4.2.1 Motivation und Erwartungen

Kritik am bisherigen Vorgehen

Das in der Praxis gängige Vorgehen, SQL um entsp. räumliche Konstrukte und Konzepte zu erweitern, um so zu räumlichen Anfragesprachen zu kommen, wurde insbesondere von Egenhofer stark kritisiert ([Ege92]). Einige seiner Argumente sind (s. auch Kap. 3):

- **Fehlende Natürlichkeit:** Bei der Anfrageformulierung muß der Benutzer sein „mentales Bild“ von der Anfrage auf die als semantisch zu niedrig und somit nicht adäquat empfundenen SQL-Operationen (auf Tabellen) abbilden. Dieser Aufwand wird als unnötig hoch eingestuft; die Anfragen werden unnötigerweise übermäßig komplex, da alle räumlichen Aspekte textuell explizit gemacht bzw. versprachlicht werden müssen (s. Kap. 2, „propositional vs. analogisch“).
- **Heterogenität verschiedenster SQL-Erweiterungen:** Jedes erweiterte SQL bietet zwar SDTs (Spatial Data Types, s. Kap. 3), aber sowohl die Typen selbst als auch die entsp. Operationen auf ihnen variieren stark von Erweiterung zu Erweiterung. Sowohl Syntax als auch Semantik sind breit gestreut, und zudem gibt es für die Semantik der räumlichen Relationen meist keine formalen Definitionen. Einigkeit besteht lediglich darüber, daß das relationale Modell um SDTs erweitert werden muß: wenn es auch möglich ist, räumliche Objekte und Relationen in einem relationalen Datenmodell (in Tabellen) zu repräsentieren, so führt dieses Vorgehen doch zu erheblichen Problemen (u.a. zu mangelhafter Performanz, s. Kap. 3). Inzwischen wird im Rahmen der SQL-Standardisierungsverfahrens (SQL 3 und 4) jedoch ein Standard für ein erweiteres SQL mit SDTs erarbeitet.
- **Fehlende Grafikmöglichkeiten:** SQL sieht weder grafische Darstellungen noch grafische Interaktionsmöglichkeiten vor, was insbesondere in GIS-Anwendungen extrem nachteilig ist, da hier Ergebnisresultate und Datenbestände „kartenartig“ inspiziert und manipuliert werden müssen. Auch hier wurden wieder die unterschiedlichsten Erweiterungen vorgenommen.
- **Unzureichende Mächtigkeit des relationalen Modelles:** Das relationale Datenmodell an sich ist unzureichend, denn best. Anfragen können nicht formuliert werden. So können z.B. Typanfragen per Selektion von präsentierten Objekten nicht beantwortet werden (der Benutzer selektiert z.B. mit dem Mauszeiger ein präsentiertes grafisches Objekt und fragt „Was ist das?“), da solche Anfragen das *Datenbankschema* und nicht die Extension der Datenbank betreffen (es handelt sich um „Metadata-Queries“).

Als Fazit seiner Untersuchung kommt Egenhofer zu folgendem Schluß ([Ege92]):

> Instead, new high level interface languages for GIS are necessary that support the retrieval of data, the appropriate representation of query results and the interaction between the user and the system in an integrated fashion. The design of such a language must start at the user level by investigating what kinds of operations users want to perform on spatial databases and how they do it ... In a complex environment, such as a GIS, „queries“ are part of a dynamic process during which users request information with respect to the currently visible information and make modifications in the information displayed. To support such a working behaviour, interface languages for GISs are necessarily based on cognitive and mental models, such as image schemata and metaphors ..., applied to spatial data ...

Gesteigerte Benutzungsfreundlichkeit

Nur wenige (potentielle) Nutzer von Informationssystemen und Datenbanken verfügen über ausreichende Detailkenntnisse z.B. der Datenmodelle oder Anfragesprachen, da sie in der Regel keine

EDV-Spezialisten oder Informatiker sind. Visuelle Anfragesprachen bieten evtl. die Möglichkeit, diese Systeme und deren Datenbestände einem breiteren Personenkreis zugänglich zu machen (man spricht auch von der Gruppe der *gelegentlichen Benutzer [Casual Users]* und *Endbenutzer [End Users]*). Sicherlich spielen Informationssysteme eine zunehmend wichtigere Rolle in unserer Gesellschaft (s. obige Bemerkungen zum WWW etc.).

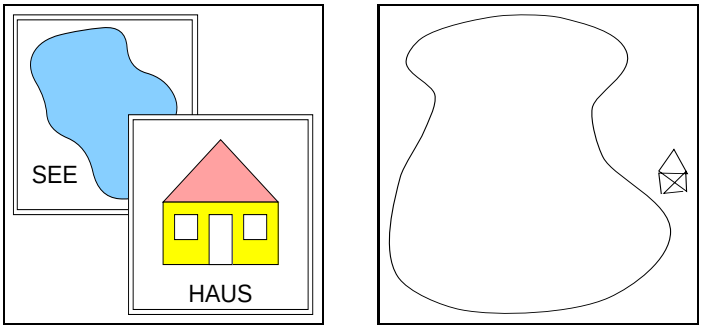
Da in heutigen Informationssystemen bzw. Datenbanken längst nicht mehr nur alphanumerische Daten (wie z.B. in einem klassischen Personalinformationssystem) gehalten werden, werden diese auch zusehends komplexer und komplizierter in der Handhabung. Eventuell können visuelle Sprachen diesem Trend entgegenwirken.

Informationsquantität durch visuelle Darstellungen

Die *hohe Bandbreite des menschlichen visuellen Systems* kann auch zur *Orientierung* in diesen umfangreichen Datenbeständen genutzt werden: so könnte z.B. eine virtuelle Realität (Virtual Reality) zur Visualisierung umfangreicher Datenbestände genutzt werden. Der Benutzer könnte die Daten durchwandern, überfliegen und direkt-manipulativ mit große Datenmengen hantieren. Durch sogenannte perzeptuelle Inferenzen könnten er Trends und bisher versteckte Informationen entdecken (Data Mining). Visualisierung ist jedoch nicht Thema dieser Arbeit.

Natürlichkeit

Während die meisten bisher realisierten VQLs für konventionelle alphanumerische Informationssysteme und Datenbanken gedacht sind (s. [CCLB97]), werden im Kontext dieser Arbeit VSQLs betrachtet. Sie sind für Informationssysteme gedacht, deren Daten inhärent räumlichen Charakter haben: in einem GIS sollen z.B. Konstellationen wie „Ein Haus am See“ angefragt werden, und zwar auf visuelle Weise. Daher stellt sich die Frage, welche Möglichkeiten der visuellen Repräsentation der Objekte „Haus“ und „See“ sowie der Präposition „am“ – welche ja eine Art räumliches „in der Nähe von“ bedeutet – es geben könnte. Man benötigt also einen geeigneten *visuellen Formalismus* zur Repräsentation derartiger Anfragen: in Kap. 2 wurde die in der Literatur zu findende Einteilung in ikonische, formular- und diagrammbasierte visuelle Formalismen vorgestellt.



(a) Ikonische Anfrage (b) Skizzenbasierte Anfrage

Abbildung 4.1: Visuelle Anfrage „Haus am See“

Man könnte z.B. ikonische abstrakte Stellvertreter für Haus und See festlegen (s. Abb. 4.1(a)) und diese in der Ebene in eine bestimmte räumliche Relationen zueinander setzen. Diese Relationen müßte nicht explizit (wie in einer textuellen Anfragesprache) notiert werden, sondern wäre intrinsisch repräsentiert. In den meisten ikonischen visuellen Sprachen ist die „Überlappt“-Relation zwischen Ikonen die wichtigste Relation.

Eine andere Möglichkeit wäre, den Benutzer eine Skizze der aufzufindenden interessierenden Konstellation malen zu lassen (s. Abb. 4.1(b)): wiederum könnten topologische (und evt. auch geome-

trische) Aspekte der Skizze als notwendige Bedingungen für die aufzufindende Konstellation im Datenbestand interpretiert werden. Diese Möglichkeit erscheint besonders attraktiv, da sie für den Benutzer die Möglichkeit bietet, sein mentales Bild nahezu unmittelbar (und mit minimalen kognitiven Aufwand) in die Anfrage zu projizieren. Der Aufwand wird umso geringer, je „ähnlicher" der verwendete visuelle Formalismus und die mentalen Bilder sind.

In beiden Fällen ergeben sich positive Auswirkungen aufgrund der intrinsischen Selbstkonsistenz (s. Kap. 2): bestenfalls können inkonsistente Anfragen nicht mehr formuliert werden. Werden inkonsistente Anfragen nicht im vornherein von der Anfragebearbeitungsmaschine erkannt, so führt ihre Ausführung zu einem maximalen Zeitaufwand, nur um dann das leere Ergebnis zu liefern. Während in einem erweiterten SQL-Dialekt ohne Probleme eine Bedingung der Form `overlaps(a,b) AND disjoint(b,a)` formuliert werden könnte, ist es eben nicht möglich, eine Skizze dieser Konstellation anzufertigen. In Kap. 2 wurde ja dargestellt, daß die Entdeckung derartiger Inkonsistenzen einen umfangreichen qualitativen räumlichen Kalkül erfordert und z.B. für die Egenhofer-Relationen NP-hart ist ([GPP95]) – eine bildhafte analogische Repräsentation der wesentlichen räumlichen Aspekte hingegen macht diese Inferenzen ohne zusätzlichen Aufwand (s. auch [Lin95]).[2] Diese Selbstkonsistenz gilt natürlich nicht automatisch für jede bildhafte Darstellung und visuelle Sprache.

Die visuelle Repräsentation räumlicher Konzepte scheint sowohl problem- als auch kognitiv adäquater als eine propositionale bzw. textuelle Repräsentation zu sein. Diese Angemessenheit unterscheidet eine „gute" von einer „schlechten" Repräsentation für ein gegebenes Problem.

4.2.2 Probleme

Transparenz

Ist vom Benutzer erst einmal eine visuelle Anfrage erstellt, so muß vom System eine Interpretation geleistet werden, um die Bedeutung zu extrahieren (s. Kap. 2). Dies ist notwendig, da ein „Bild" zunächst stets ein syntaktisches Gebilde ist und somit an sich keine Bedeutung (Semantik) hat. Insbesondere muß nun auf potentielle Diskrepanzen zwischen der vom Benutzer intendierten Bedeutung der Anfrage und der vom System (anhand der Interpretationsfunktion) inferierten Semantik geachtet werden. Im Rahmen des visuellen Parsings (s. Kap. 2) wird das System anhand der konkreten Syntax des Bildes (also anhand der intrinsisch repräsentierten Aspekte) eine abstrakte syntaktische, explizite propositionale Repräsentation ermitteln (z.B. in Form eines Anfragegraphen).

Wie auch bei der Interpretation einer Landkarte kommt es hier auf die exakte Kenntnis der relevanten Aspekte an: so werden z.B. in einer politischen Europakarte die Farben der dargestellten Länder nicht als Farben des Erdbodens dieser Regionen interpretiert. Dies ist umso wichtiger bei analogischen Repräsentationen, da der große Vorteil der intrinsischen Repräsentation gewisser Aspekte auch gleichzeitig der größte Nachteil ist, denn es gibt keine expliziten visuellen Anzeichen in Form von Objekten für z.B. das Vorhandensein einer best. räumlichen Relation. Während in einem erweiterten SQL-Dialekt die entsp. räumliche Relation explizit repräsentiert werden muß und das Vorhandensein der Relation trivialerweise durch „einfaches Lesen" ermittelt werden kann, muß hier der Benutzer zusätzlich die vom System erkannten Relationen *kennen* und *selbst* anhand einer perzeptuellen Inferenz auf das Vorhandensein dieser Relation schließen. So macht es z.B. wenig Sinn, die Bedeutung eines Kreises mit einem Radius von 1 cm als Konzept A, die eines Kreises mit einem Radius von 1,01 cm jedoch als Konzept B festzulegen.

[2] Der Vergleich hinkt etwas, da in einem Constraintnetzwerk ja auch ganze Disjunktionen von möglichen Relationen – also un- bzw. unterbestimmte Informationen – dargestellt werden können, was in einer bildhaften Darstellung nicht ohne weiteres möglich ist.

Überspezifiziertheit

Ein generelles Problem ist die Überspezifiziertheit: ein Bild (z.B. ein Diagramm) repräsentiert nicht nur die als relevant betrachteten Aspekte perfekt, sondern durch die konkrete Geometrie der Darstellung erzwungenermaßen auch Aspekte, die keine Bedeutung haben ([Hab87, Ege96b, BF97]). Während es in einer expliziten Modellierung keine Anzeichen für nicht-relevante Relationen und Eigenschaften gibt, ist dies hier nicht der Fall: ein Bild ist ohne Festlegung aller räumlicher Aspekte nicht möglich. Selbst wenn die Form eines dargestellten Objektes vollständig irrelevant ist, muß man für die bildhafte Darstellung doch eine Form auswählen (dies ist in einer textuellen Sprache strenggenommen auch der Fall: nennt man einen Bezeichner z.B. `HAUS`, so hat dieser auch eine Form, wenn man einen Text als „Bild“ betrachtet – wir sind jedoch daran gewöhnt, hiervon zu abstrahieren). Für den Benutzer ist somit wieder die korrekte Kenntnis der Interpretationsfunktion notwendig.

Knapp werdende Visualisierungsdimensionen

In Kap. 2 wurde die Palmersche Repräsentationstheorie anhand der Welten $\mathcal{A}$ bis $\mathcal{H}$ diskutiert: hier wird jeweils eine andere „Visualisierungsdimension“ als bedeutungstragend erklärt – von den anderen kann abstrahiert werden. Die einzelnen Welten benutzen verschiedene Visualisierungsdimensionen wie Höhe, Länge, Breite, Position, Form, Größe, Orientierung (Lage), Farbe, Textur, etc. Die von der Repräsentationsfunktion (oder auch Visualisierungs, Präsentations- oder Materialisierungsfunktion) nicht genutzten visuellen Dimensionen sind frei für eine günstige Wahl, z.B. für ein geschicktes Layout nach ästhetischen Gesichtspunkten. Ein bekannter Effekt ist, daß bei ungünstiger Wahl der freien visuellen Variablen (und nur partieller Kenntnis der Interpretationsfunktion bei den Interpreten) ungewollte Bedeutungen induziert werden. Dieses Problem ist z.B. in der Kartographie bekannt. Das Design von Karten wurde ausgiebig von J. Bertin untersucht – von ihm stammt auch der Begriff der „visuellen Variablen“ (s. die Übersicht in [McC83]).

Ein offensichtliches Problem besteht nun darin, daß die freien Visualisierungsdimensionen knapp werden, wenn man den Anspruch erhebt, *möglichst viele Aspekte der Weltobjekte direkt zu repräsentieren.* Im Extremfall nähert man sich einem Isomorphismus, der die Identitätsfunktion ist. Ein Photo einer zweidimensionalen Welt kommt diesem sehr nahe – hier gibt es keine Abstraktionsmöglichkeiten mehr. Sollen z.B. Rechtecke visualisiert werden, so ist es sicherlich kontraintuitiv, diese durch Kreise darzustellen. Form sollte identisch durch Form repräsentiert werden, denn dies ist die natürlichste Repräsentation, da sie der Wirklichkeit am nähesten kommt. Soll die Interpretierbarkeit durch den Menschen sichergestellt werden und ist jede Form prinzipiell möglich, so gibt es praktisch keine andere Möglichkeit, als Form *identisch* (und somit natürlich auch isomorph) durch Form zu repräsentieren.[3]

Spielt hingegen die Form von Objekten keine Rolle (bzw. handelt es sich um formlose abstrakte Objekte) und muß diese Eigenschaft somit nicht repräsentiert werden, so spricht nichts dagegen, alle Objekte z.B. als Kreise darzustellen. In „Entity Relationship (ER)“-Diagrammen werden z.B. lediglich die Formen Kreis, Rechteck und Raute verwendet – sie repräsentieren jedoch nicht die Form der Objekte, sondern dienen lediglich zur Unterscheidung von Attributen, Entitäten und Relationen (daher handelt es sich nicht um einen analogische Repräsentationsformalismus). Offensichtlich hat man bei der Visualisierung abstrakter Strukturen sehr viel mehr Freiheiten als bei der visuellen Repräsentation von Objekten der wirklichen Welt, die ja ein intrinsisches visuelles Erscheinungsbild haben und somit – wenn keine Abstraktionen vorgenommen werden – diesbezüglich möglichst ähnlich dargestellt werden sollten.

[3] Natürlich könnte man sich eine komplizierte bijektive Deformationsfunktion vorstellen – für einen Menschen besteht dann jedoch keine Möglichkeit mehr, die ursprüngliche Form und somit Bedeutung der Repräsentation zu rekonstruieren (Rotation und Skalierungen gelten nicht als Deformationen).

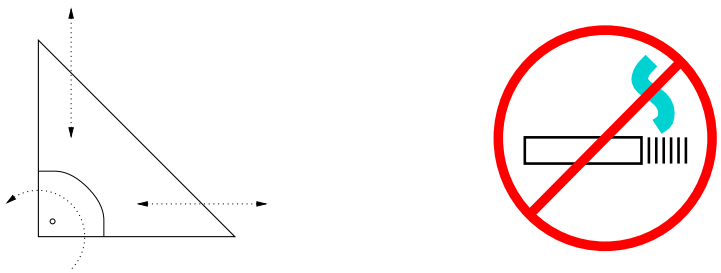
Vagheit

Ähnliche Probleme ergeben sich bei der Darstellung vager Information: will man Form identisch durch Form repräsentieren (und jede Form in der Welt sei möglich), so gibt es zunächst wieder keine Möglichkeit, eine abstrakte oder vage Form direkt durch Form zu repräsentieren. Man könnte sich eine spezielle Form als Repräsentanten aller abstrakten Formen ausdenken, müßte aber gleichzeitig sicherstellen, daß es keine konkrete Form in der Welt gibt, die dieselbe Form wie dieser Repräsentant hat, da anderseits die eindeutige Interpretierbarkeit der abstrakten Form des Repräsentanten nicht mehr gegeben wäre. Abstraktion von der dargestellten Form ist nur dann möglich, wenn die Eigenschaft „Form“ nicht zur direkten Repräsentation von Form selbst genutzt wird (sondern z.B. zur Darstellung von Klassenzugehörigkeiten wie in ER-Diagrammen). Im Extremfall der Topologie spielt die Form keine Rolle mehr: topologisch läßt sich ein Kreis nicht von einem Viereck unterscheiden, da beide homöomorph sind.

Abstraktheit

Bei der bildhaften Darstellung abstrakterer Konzepte gibt es zunehmende Schwierigkeiten: Wie visualisiert man z.B. die Klasse aller rechtwinkligen Dreiecke? Hier wird es offensichtlich schwierig, da ein einziges rechtwinkliges Dreieck zunächst nur für sich selbst steht. Gilt hingegen die Konvention, daß die Form keine Rolle spielt, so wird nicht nur die Klasse der rechtwinkligen Dreiecke repräsentiert, sondern eine sehr viel größere Klasse.

Die Interpretationsvorschrift, daß es nur auf den rechten Winkel ankommt und die Dreiecksform ankommt, ist ohne weiteres nicht direkt ersichtlich: hier könnten jedoch explizite Anzeichen in Form zusätzlicher Objekte helfen, die im weiteren Verlauf der Arbeit *Metaobjekte* genannt werden. Metaobjekte dienen dazu, dem Betrachter Hinweise für die richtige Interpretation zu geben. In technischen Zeichnungen wird ein rechter Winkel durch einen Viertelkreis mit einem Punkt annotiert und somit in gewisser Weise „explizit“ gemacht. Zudem werden unterschiedliche Pfeile verwendet. Eine gute Visualisierung der Klasse der rechtwinkligen Dreiecke ist in Abb. 4.2(a) dargestellt. Ein anderes Metaobjekt wird im „Nicht Rauchen!“-Ikon verwendet (s. Abb. 4.2(b)): wir haben gelernt, daß hier nicht eine qualmende Zigarette hinter einer Schranke dargestellt ist. Wieder könnte man den Querbalken als Metaobjekt bezeichnen – er repräsentiert kein Objekt in der Welt per Ähnlichkeit, sondern dient dazu, die Botschaft „Rauchen“ zu negieren. Im weiteren Verlauf sind jedoch vor allem Ikonen interessant, die in Kap. 2 als *repräsentationale Ikonen* bezeichnet wurden: das Ikon in Abb. 4.2(a) ist als solches einzustufen, im Gegensatz zum „Nicht Rauchen!“-Ikon (Abb. 4.2(b)), welches zur Klasse der *abstrakten Ikonen* (s. Kap. 2) gehört. Das „Nicht Rauchen!“-Ikon könnte als Kombination eines repräsentationalen Ikones und eines *Meta-Ikones* betrachtet werden: das Meta-Ikon dient zur Negierung des durch das repräsentationale Ikon formulierten Sachverhaltes. Offensichtlich handelt es sich dann bereits um einen *ikonischen bzw. visuellen Satz.*



(a) Die Klasse der rechtwinkligen Dreiecke (b) „Nicht Rauchen!“-Ikon

Abbildung 4.2: Ikonen

Die Schwierigkeiten bei der Darstellung abstrakter Konzepte wurden u.a. von Sloman beobachtet ([Slo95]):

> Analogical representations (e.g. pictures and diagrams) can be very powerful in some cases ... but in other cases logic wins, especially where disjunctive, negative, conditional, or quantified information is involved.

Diese Problematik mag in unserer Wahrnehmung begründet sein: in der Regel werden nur existente Objekte wahrgenommen, und wenn mehrere Objekte wahrgenommen werden, so existieren sie alle; es liegt also eine Konjunktion vor. Nichtexistente Objekte werden auch nicht wahrgenommen (man mag sich darüber streiten, ob eine „Nicht-Wahrnehmung“ nicht auch eine Wahrnehmung ist).

Daß es verschiedenste Abstraktheitsgrade visueller Darstellungen gibt, wurde bei der Diskussion der Ikonen deutlich – Klassifizierungen sind hier schwierig und kaum einzuhalten (u.a. hierin zeigt sich wieder die Breite möglicher bildhafter Darstellungen – hier existieren noch mehr Freiheiten als in der natürlichen Sprache). Teilweise werden also visuelle Repräsentationen von Botschaften, Prozessen oder Konzepten benötigt, für die es kein offensichtliches visuelles Erscheinungsbild gibt. Von Chang stammt der Begriff des *Operator-* oder *Prozeßikones (Process Icons)* ([Cha90, Kap. 1]): ein solches Ikon steht für eine Berechnung, eine Operatoranwendung oder einen Prozeß. Von analogischen Repräsentationen kann hier jedoch nicht mehr die Rede sein. Mit zunehmender Abstraktheit verschwindet die Ähnlichkeit bzw. strukturelle Äquivalenz zusehends, so daß die richtige Interpretation zusehends von Kontext, Metaphorik und Konventionen abhängt.

4.3 Vorstellung einiger visueller räumlicher Anfragesprachen

Im folgenden sollen die wesentlichsten Charateristika von vier visuellen räumlichen Anfragesprachen vorgestellt und gleichzeitig miteinander verglichen werden. Auffallend ist die geringe Anzahl bisher implementierter Systeme und entwickelter VSQLs. Die im folgenden diskutierten VSQLs wurden im historischen Werdegang dieser Arbeit berücksichtigt, woraus sich die Relevanz der folgenden Darstellung ergibt. Zwei der vorgestellten Sprachen sind ikonisch, die anderen beiden benutzen Diagramme bzw. Skizzen.

Die Ikonen bzw. skizzierten Objekte repräsentieren stets die aufzufindenden Objekte im Datenbestand (ich nenne sie „Anfrageobjekte“). Alle vier Sprachen beschränken sich auf zweidimensionale räumliche Konzepte. Die erzwungenen Relationen zwischen den Anfrageobjekten sind stets topologischer Art – geometrische und metrische Bedingungen können somit nicht oder nur durch die Anwendung spezieller Operatoren formuliert werden. Die einzelnen Sprachen unterscheiden sich bzgl. der Ausnutzung intrinsisischer Aspekte der Ebene für die Anfrageformulierung beträchtlich voneinander: eine Sprache verwendet die Ebene ausschließlich zur Visualisierung („CIGALES“), eine Sprache unterscheidet zwischen extrinsischen und intrinsischen räumlichen Relationen (Lee und Chin), die anderen beiden Sprachen verwenden ausschließlich intrinsisisch repräsentierte Relationen.

Allen Sprachen ist gemein, daß mit ihnen sowohl räumliche als auch nichträumliche Aspekte angefragt bzw. festgelegt werden können. Natürlich müssen Typangaben bzw. thematische Bedingungen wie z.B. „Haus“ oder „See“ formuliert werden können. Ich konzentriere mich in der folgenden Darstellung jedoch primär auf die räumlichen Aspekte der einzelnen Sprachen. Alle Sprachen benutzen eine Art „Grafikeditor“ zur Konstruktion der visuellen Anfrage. Diese wird dann übersetzt und z.B. von einem DBMS bearbeitet.

Man kann alle vier Systeme als „Query-by-Pictorial-Example“-Systeme ansehen, da ja vom Benutzer eine Art Beispiel der aufzufindenden Objektkonstellation (bzw. gewisse Aspekte dieser) vorgegeben wird (der Begriff „Query-by-Pictorial-Example“ hat hier nichts mit der visuellen Formularsprache „Query-by-Example“ zu tun!). Im folgenden bezeichne ich die aufzufindenden Objekte im

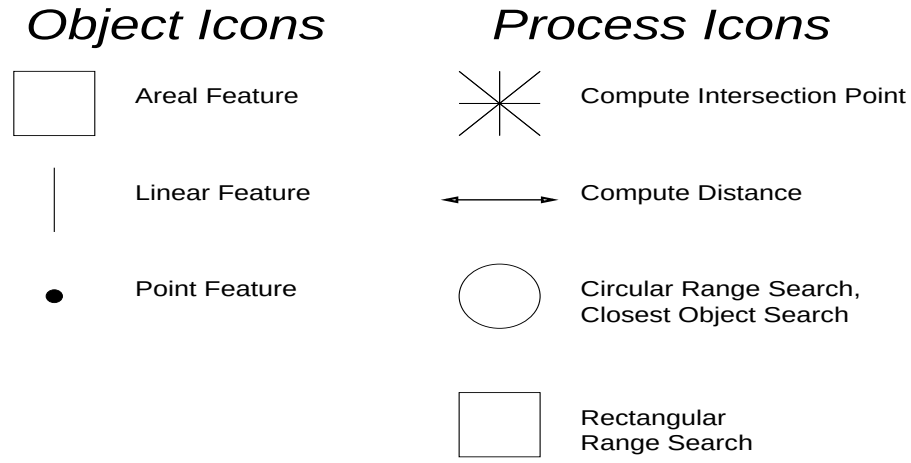


Abbildung 4.3: Ikonen in der Sprache von Lee und Chin (reproduziert nach [LC95])

Datenbestand als *DB-Objekte* und ihrer Stellvertreter in der visuellen Anfrage als *Anfrageobjekte.* Eine andere interessante Sprache ist [PS95], die nicht in den Vergleich mit aufgenommen wurde, da sie in gewisser Weise zu abstrakt erscheint.

4.3.1 Eine ikonische Anfragesprache für GIS

Hierbei handelt es sich um eine ikonische Anfragesprache für GIS ([LC95]). Die Ikonen repräsentieren DB-Objekte im GIS wie Seen, Flüsse, Länder, Städte etc. Es handelt sich also um repräsentationale Ikonen (lt. Chang „Object Icons“). Die Ikonen sind recht abstrakt, denn sie repräsentieren lediglich die Dimensionalität der Weltobjekte analogisch. Vorgesehen sind Ikonen für Rechtecke, Strecken und Punkte (s. Bild 4.3). Im Gegensatz zu konventionellen Ikonen läßt sich nicht nur die Position, sondern auch die Ausdehnung der Ikonen beliebig wählen. Anhand der Position und Ausdehnung der Ikonen werden nun topologische Relationen zwischen den Ikonen errechnet, die als *notwendige Bedingungen* für die aufzufindenden DB-Objekte gelten müssen. Lee und Chin sprechen unpassenderweise von „ikonischen Operatoren“ - diese Operatoren werden jedoch automatisch angewendet und müssen nicht etwa vom Benutzer explizit angewendet werden. Zwei Anfragen in dieser Sprache sind in Abb. 4.4 dargestellt: die textuelle Umschreibungen dieser Anfragen wären

1. „Finde alle schiffbaren Flüsse, die in Land-A hineinfließen“ („Find all navigable rivers entering Country-A“)
2. „Finde eine Landparzelle, die einen Brunnen enthält und an einem Teich mit Freizeitwert und Inseln liegt. Das Landstück soll teilweise bewaldet und von der Stadt-A über eine Straße erreichbar, aber mindestens einen Kilometer von der nächsten Tankstelle entfernt sein („Find a land parcel containing a well and facing a recreational lake with islands on it. The parcels are partially wooded, accessible from City-A by road but are at least 1 km from any gas station.“).

Betrachtet man die visuellen Anfragen, so erscheint die Klassifizierung als ikonische Sprache zumindest fraglich, da es sich eher um *Diagramme* (s. Kap. 2) als um ikonische Konstellationen im klassischen Sinne handelt.

Die vom System erkannten topologischen Relationen sind in Abb. 4.5 dargestellt und können mit qualitativen Orientierungen (s. Kap. 2) verfeinert werden, wenn sich das Programm in einem speziellen Modus befindet. Zudem sind Operatorikonen („Process Icons“ lt. Chang) vorgesehen: so wird z.B. der Operator „Selektion aller Objekte im Radius von r Metern um einen Punkt“ durch einen Kreis dargestellt, während der Operator zur Berechnung der Distanz zwischen zwei Objekten

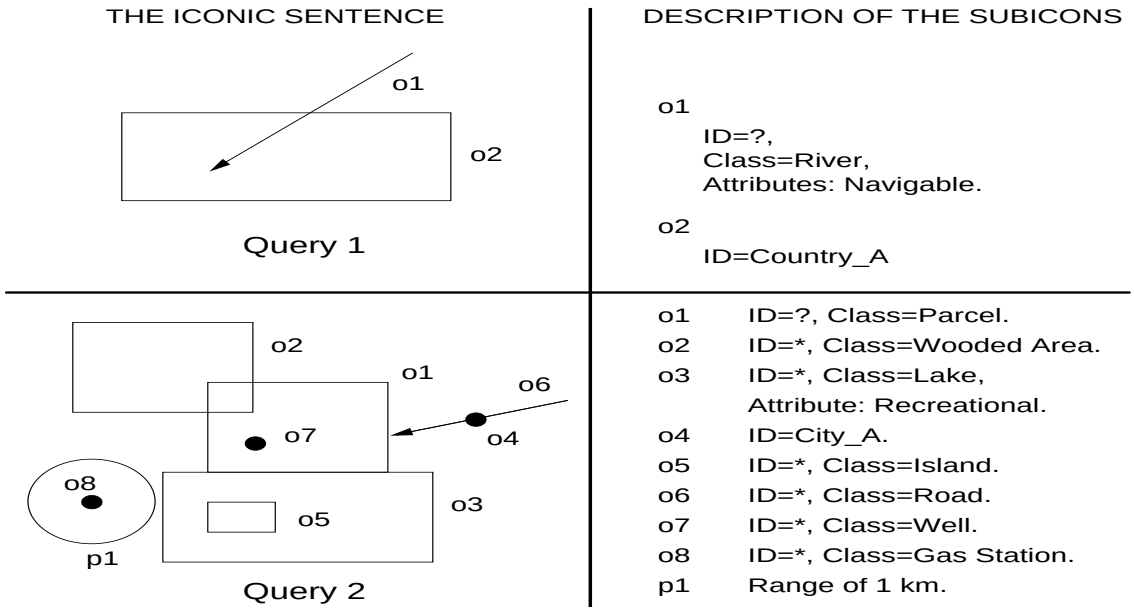
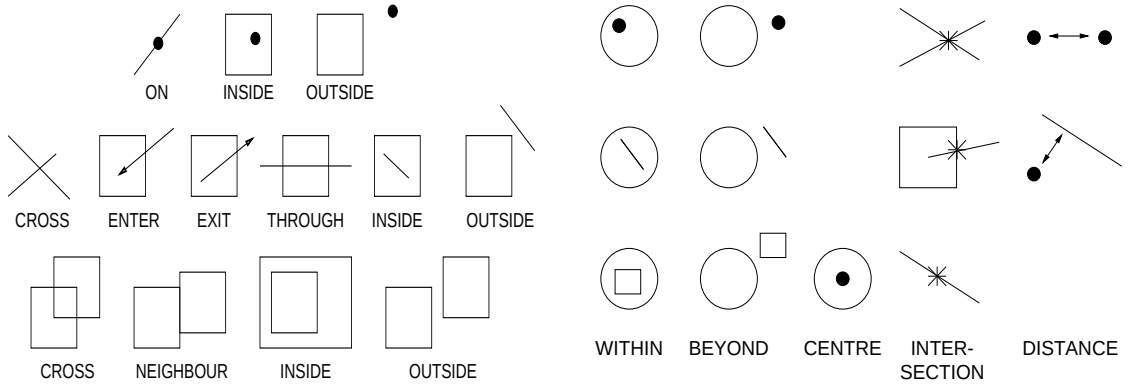


Abbildung 4.4: Zwei ikonische Anfragen: das primäre aufzufindende Objekt ist in den rechten Seiten der Anfragen mit „ID=?“ gekennzeichnet. „ID=*“ steht hingegen für ein beliebiges Objekt (reproduziert nach [LC95])

durch einen Doppelpfeil dargestellt wird. Zwischen Ikonen und Prozeßikonen werden die in Bild 4.5(b) dargestellten topologischen Relationen erkannt. Soll z.B. der Schnittpunkt zweier Objekte berechnet werden, so wird in der Konstellation an der Stelle, wo sich die beiden Ikonen schneiden, vom System der Schnittpunkbestimmungsoperator eingezeichnet. Durch die Operatoren und die so entstehenden Ikonen werden also spezielle räumliche Relationen vom Benutzer explizit gemacht. Da eine Konstellation lediglich topologisch interpretiert wird, stellt sich wieder die Frage, ob nicht in Verbindung mit den metrischen Operatoren inkonsistente Konstellation konstruiert werden können.

Interessant ist auch das *Vordergrundkonzept (Foreground Concept):* nicht alle in einer vom Benutzer angelegten ikonischen Konstellation ersichtlichen topologischen Relationen wurden von ihm tatsächlich beabsichtigt. Doch auch unbeabsichtigte Relationen sind perfekt intrinsisch repräsentiert und erzeugen eine nichtbeabsichtigte Bedingung bzw. Einschränkung (Constraint) in der Anfrage. Es wurde ja bereits diskutiert, daß hier ein Abstraktionsmechanismus wünschenswert ist. Beim Erzeugen eines neuen Ikones können daher vom Benutzer zuvor genau die Objekte in den *Vordergrund* geholt werden, zu denen Relationen berechnet werden sollen. Dieses Vorgehen scheint auf die Dauer etwas lästig zu sein – während zwar der Typ der räumlichen Relation automatisch anhand der Geometrie errechnet wird, muß hier jedoch jeweils das zweite Argument des entsp. Relationstupels vom Benutzer explizit durch Selektion bestimmt werden, indem er das entsp. Objekt zuvor in den Vordergrund holt. Schließlich ist noch zu kritisieren, daß in der Konstellation erzwungene und nicht-erzwungene Relationen später nicht mehr unterscheidbar sind: die Sprache ist daher nicht visuell vollständig (s. Kap. 2). Nachdem der Benutzer die Konstruktionshistorie vergessen hat, ist die Bedeutung der visuellen Anfrage nicht mehr rekonstruierbar: Ein spezielles Werkzeug soll daher (in Verbindung mit einer textuellen Repräsentation) zur Konstruktion und zum Verstehen der so erzeugten ikonischen Sätze verwendet werden. Im Prototypen wird jeweils eine textuelle Repräsentation mitgeführt, da ohne diese die dargestellten ikonischen Sätze nicht mehr eindeutig interpretierbar wären.

Anhand der kompletten visuellen Anfrage wird schließlich eine textuelle „EQUEL“-Anfrage erzeugt, denn das zugrundeliegende GIS ist mit Hilfe eines konventionellen RDBMS („INGRES“)



(a) Räumliche Relationen zwischen „Object Icons“

(b) Räumlicher Relationen zwischen „Object Icons“ und „Operator Icons“

Abbildung 4.5: Erkannte räumliche Relationen (reproduziert nach [LC95])

realisiert. Zum Zeitpunkt der Veröffentlichung des Artikels wurden nur Punkte unterstützt („limitations of a conventional DBMS“).

4.3.2 „Sketch!“

Zentrale Metapher für „Sketch!“ ist eine Tafel, an die Skizzen gezeichnet werden ([Mey94a, Mey94b]). Die so skizzierten Objekte werden von Meyer als *Acons (Analogical Icons)* bezeichnet. Laut Meyer sind Acons Objekte, die sowohl die Eigenschaften von Skizzen als auch Ikonen haben. Im Gegensatz zur VSQL von Lee und Chin können hier also *beliebige Formen* gezeichnet werden, was als deutlicher Vorteil zu sehen ist, da so die zu findenden DB-Objekte den Ikonen *ähnlich* sehen. Die Acons werden mit einem Grafikeditor erzeugt, so daß bestimmte Probleme beim Interpretieren der Skizze gar nicht erst auftreten. Aufgrund der Freiformgestaltung kann sich der Benutzer in seiner Anfrage wesentlich besser zurechtfinden und orientieren, da die Oberfläche weniger abstrakt erscheint. Die Gestaltung einer konkreten Anfrage fällt daher leichter; dazu trägt auch die Verwendung der Tafelmetapher bei. Die Wichtigkeit von Metaphern wurde ja bereits in Kap. 2 dargestellt.

Eine Anfrage in „Sketch!“ ist nun in Abb. 4.6 dargestellt: ein Fluß fließe von oder in einen überfüllten See. In der Anfrage wird nun nach den Namen aller Personen gefragt, die eventuell durch den überfüllten See auf ihrem Weg zur Arbeit behindert werden, was der Fall ist, wenn diese Personen den besagten Fluß überqueren müssen, um zur Arbeitstelle zu gelangen. Da der See überfüllt ist und somit der Fluß viel Wasser führt, könnte der Weg zur Arbeit bzw. die zu überquerende Brücke überschwemmt sein. Zusätzlich zur Skizze sind in Abb. 4.6 noch *propositionale Bedingungen* in einer ER-Diagramm-ähnlichen grafischen Notation formuliert. Die Semantik der Anfrage ist nun folgende:

$$
\begin{aligned}
&P_1 : person \wedge T_1 : town \wedge Rd_1, Rd_2 : road \wedge F_1 : factory \wedge \\
&R_1 : river \wedge L_1 : lake(level = \text{„}high\text{“}) \wedge B_1 : bridge \wedge \\
&W : works\_at(employee = P_1, place = F_1) \wedge \\
&L : lives\_in(resident = P_1, location = T_1) \wedge \\
&R_1 \ intersects \ L_1 \wedge R_1 \ intersects \ B_1 \wedge Rd_1 \ touches \ F_1 \wedge \\
&Rd_1 \ touches \ B_1 \wedge B_1 \ touches \ Rd_2 \wedge Rd_2 \ touches \ T_1.
\end{aligned}
$$

Dennoch sind die Acons deutlich ausdrucksschwächer als Skizzen (bzw. die Bedeutung, die das System ermittelt), da die skizzierte Form der Acons hier völlig irrelevant ist und nicht interpretiert wird. Kein Mathematiklehrer würde seinen Schülern das Konzept des Kreises mit einer Skizze eines

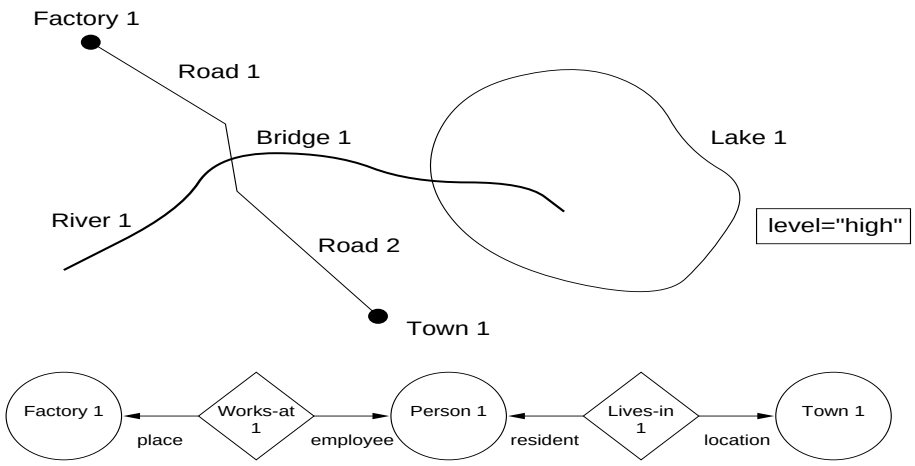


Abbildung 4.6: Eine skizzenartige Anfrage in „Sketch!“ (reproduziert nach [Mey94a])

Dreiecks an der Tafel erklären, so daß Meyers Metapher hier nicht mehr greift. Meyer beansprucht jedoch breite Einsetzbarkeit seiner Sprache, auch außerhalb des GIS-Kontextes. In vielen Anwendungen ist es jedoch notwendig, Objekte anhand ihrer geometrischen Attribute und Relationen klassifizieren zu können (s. z.B. „ADIK“ in Kap. 2): diese Ausdrucksmächtigkeit fehlt allen hier vorgestellten VSQLs.

Ähnlich der VSQL von Chin und Lee werden auch hier topologische Relationen zwischen den Acons berechnet: Meyer sieht die Relationen „closest“, „intersects“, „neighbouring“, „inside“ und „touches“ vor (die Menge räumlicher Relationen kann bei Bedarf vom Anwender erweitert werden). Zu einigen Prädikaten gibt es korrespondierende Funktionen: z.B. kann für zwei Objekte, die in einer „intersects“-Relation zueinander stehen, das Schnittobjekt durch Anwendung der „intersection“-Funktion berechnet werden. So können u.a. *auftauchende Objekte* (von Meyer *Subobjekte* genannt) – z.B. das Schnittrechteck zweier sich schneidender Rechtecke – explizit gemacht werden. Per Operatoranwendung können auch die Kanten oder Punkte eines Polygon-Acons als eigenständige Objekte eingeführt, das Schnitt- oder Vereinigungspolygon zweier Polygonen berechnet oder der Mittelpunkt eines Objektes expliziert werden, etc.

Die Interpretation der vom Benutzer erzeugten Skizze erfolgt nach folgenden Regeln:

1. Topologische Relationen werden für alle Objektpaare getestet.
2. Alle Relationen in der Menge $P = \{closest, intersects, neighbouring, inside, touches\}$ werden getestet.
3. Wenn ein Prädikat $P(x, y)$ wahr wird und eine korrespondierende Funktion $f \in F$ vorhanden ist, dann werden die Subobjekte $f(x, y)$ expliziert und wie andere Objekte auch behandelt.
4. Topologische Relationen werden immer nur für ganze Objekte getestet; z.B. wird niemals abgeleitet, daß ein Teil eines Objektes in einem anderen enthalten ist.
5. Es werden keine negativen Bedingungen abgeleitet (z.B. $\neg inside(x, y)$).
6. Es werden keine Relationen zwischen einem Objekt und seinen Subobjekten abgeleitet.
7. Subsumierte (und damit redundante) Teilformeln werden erkannt und entfernt (z.B. kann $inside(a, c)$ aus der Beschreibung $inside(a, b) \wedge inside(b, c) \wedge inside(a, c)$ entfernt werden).

Auch Meyer sah die Notwendigkeit, topologische Relationen zwischen einigen Objekten definieren zu können und andere topologische Relationen dabei undefiniert zu lassen: Meyer spricht von „don’t care“-Relationen. Da dieses Problem in einer einzigen Visualisierungsebene nicht lösbar

ist, gibt es hier die Möglichkeit, *verschiedene Skizzen bzw. Fenster* zu erstellen. Dabei werden dann topologische Relationen nur zwischen den Acons *einer* Skizze berechnet, und dasselbe Acon darf in mehreren Skizzen auftauchen. Es stellt sich die Frage, ob dabei Inkonsistenzen auftreten können – dies wäre offensichtlich möglich, wenn die Skizzen unabhängig voneinander gemalt werden könnten (was wahrscheinlich nicht der Fall ist). Tatsächlich muß man sich wohl eher eine Art Projektion bereits erstellter Acons einer Skizze auf andere Visualisierungsebenen vorstellen (z.B. durch Selektion der entsp. Acons).

Eine Menge solcher Skizzen bildet nun eine *Ebene (Layer);* die *Bedeutung einer Ebene* ist die *Konjunktion der Bedeutungen der Skizzen (Fenster) der Ebene. Disjunktionen* sind ebenfalls formulierbar, indem *mehrere Ebenen* definiert werden. Die *Bedeutung der Anfrage* ist die *Disjunktion aller Ebenen. Negationen* sind durch *Durchstreichen einzelner Skizzen oder auch ganzer Ebenen* möglich.

Jede Anfrage in „Sketch!“ hat zusätzlich zur analogischen (skizzierten) Ebene („S-Windows“) noch eine propositionale Ebene („P-Windows“). Hier werden thematisch-semantische Bedingungen formuliert. Die fertige Anfrage wird in eine objektorientierte Logik namens „DeLOM“ translatiert. Dabei besteht gewisse Ähnlichkeit mit „DATALOG“.

„Sketch!“ hebt sich von den anderen drei hier diskutierten Sprachen vor allem durch die große Ausdrucksmächtigkeit und die formale Semantik ab: „Sketch!“ ist die einzige der hier betrachteten VSQLs, in der Negationen, Disjunktionen, Sichten sowie *rekursive Definitionen* formuliert werden können. „Sketch!“ ist mehr als *relational vollständig* – Meyer zeigte, daß „Sketch!“ mindestens die *Ausdrucksmächtigkeit linear stratifizierter Datalog-Anfragen* besitzt.

4.3.3 „CIGALES“

„CIGALES“ ist ebenfalls eine ikonische visuelle Anfragesprache ([CM94, AP95]). In der Sprache sind Ikonen für eindimensionale und zweidimensionale Objekte vorgesehen; diese haben die konkrete Form von Strecke und Kreis. Die Autoren reden zwar von einem „Grafikeditor“ zum Formulieren der Anfragen, tatsächlich wird die Maus aber lediglich zum Betätigen von Tastern (Push Buttons) verwendet. Weder Form noch Größe noch Position der Ikonen kann vom Benutzer beeinflußt werden – das Layout der erzeugten ikonischen Konstellation wird vollständig vom System bestimmt. Während automatische Layoutgenerierung in der Regel als Pluspunkt anzusehen ist, vertrete ich in diesem Kontext die Sichtweise, daß automatische Layoutgenerierung ungeeignet ist, da das erzeugte Layout in der Regel nicht mit dem mentalen Bild des Benutzers übereinstimmen wird. Die erzeugten Visualisierungen sind extrem abstrakt und verändern sich zudem dynamisch im Verlauf der Anfrageformulierung (s.u.), was nicht zur Orientierung des Benutzers beiträgt. Da das System also das Layout erzeugt, muß vom Benutzer jede erwünschte räumliche Relation bzw. Bedingung explizit gemacht werden.

Bei der Anfrageformulierung mit „CIGALES“ müssen nun schrittweise vom Benutzer im sogenannten *Arbeitsfenster (Working Area)* die relevanten Objekte und räumlichen Relationen explizit gemacht werden. Das Arbeitsfenster dient zur schrittweisen Konstruktion der Anfrage; die (teilweise) komplette Anfrage wird hingegen im *Anfragefenster (Query Area)* dargestellt (s. Abb. 4.7).

Im Arbeitsfenster können nun entweder neue Ikonen durch Betätigung eines Tasters erzeugt, oder aber bereits erzeugte Ikonen aus dem Anfragefenster per Selektion mit der Maus kopiert werden. Sind genau zwei Ikonen im Arbeitsfenster vorhanden, so kann eine räumliche Relation zwischen diesen beiden durch eine entsp. Taster-Betätigung verlangt werden. Das Layout der beiden Ikonen wird dann so verändert, daß die geforderte Relation in der Visualisierung sichtbar wird. Zum Beispiel kann zwischen zwei im Arbeitsfenster gezeigten Ikonen ein Schnitt verlangt werden. Der Schnitt wird dann eingezeichnet. Folgende Relationen können vom Benutzer explizit gemacht werden: „inside“, „intersects“, „touches (borders)“, „linked-with“, „distance-between“ (s. Abb. 4.7). Nachdem eine Relation erzeugt wurde, kann der Benutzer durch Betätigung eines anderen

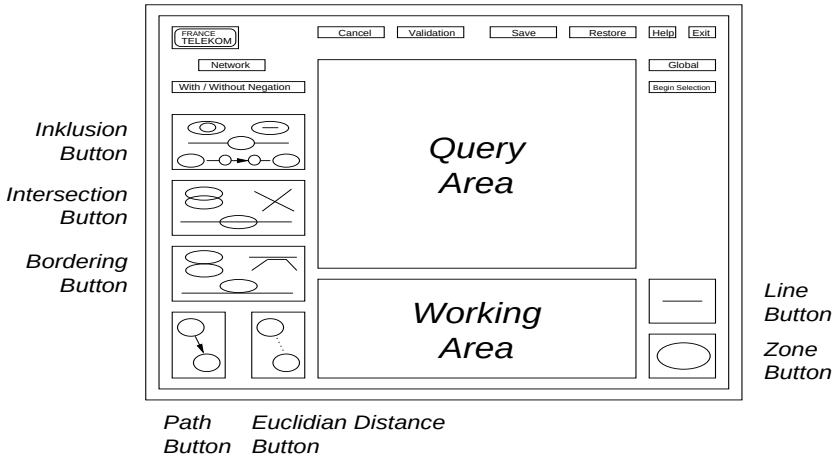


Abbildung 4.7: Der CIGALES-Editor (reproduziert nach [CM94])

Knopfes den Inhalt des Arbeitsfensters „validieren“ und somit *in das Anfragefenster integrieren.* Durch die Integration einer neuen räumlichen Bedingung oder eines neuen Objektes kann sich das Layout der ikonischen Konstellation im Anfragefenster dramatisch ändern, was nicht zur „visuellen Stabilität“ der Oberfläche beiträgt.

Tatsächlich ist das Layoutgenerierungsproblem so komplex, daß die Autoren keine allgemeine Lösung hierfür gefunden haben. Stattdessen limitieren sie maximale Anzahl, Art und Kombinationen der von einem Objekt eingehbaren räumlichen Relationen, wodurch die Anzahl der möglichen Konstellationen im vornherein stark limitiert wird. Zudem stellt sich die Frage, wie das System reagiert, wenn man ein nicht realisierbares Layout verlangt. Leider äußern sich die Autoren hierzu nicht. In den erzeugten Visualisierungen des Systemes sind zudem natürlich nicht nur die vom Benutzer verlangten explizit gemachten räumlichen Relationen ersichtlich, sondern ununterscheidbar hiervon auch zufällige, nicht-intendierte räumliche Relationen, die keine Bedeutung tragen.

Negierte räumliche Bedingungen werden *durchgestrichen* dargestellt. Zusätzliche nicht-räumliche Bedingungen für Objekte werden durch in der Nähe der entsp. Objekte plazierte Ikonen dargestellt. Hier sind u.a. Ikonen für Thematik (z.B. „Stadt“, „Wald“) und zusätzliche Einschränkungen über entsp. Attributen (z.B. „mehr als 100.000 Einwohner“) vorgesehen. Variablen (bzw. Attribute) können ebenfalls duch Ikonen visualisiert werden und durch einen „θ-Join“ miteinander verknüpft werden (z.B. ist die Anfrage „Welche beiden Städte haben die gleiche Anzahl Einwohner?“ mit einem θ =„=“-Join über dem Attribut „Einwohneranzahl“ darstellbar – in die Nähe der beiden repräsentationalen Stadt-Ikonen wird dann jeweils ein Ikon zur Darstellung des Attributes „Einwohneranzahl“ plaziert, und beide Einwohneranzahl-Ikonen werden zusätzlich mit der gleichen Ziffer beschriftet, wodurch die Gleichheit des Attributes visualisiert wird).

Einige Anfragen sind nun in Abb. 4.8 ersichtlich:

1. Welche Städte haben mehr als 100.000 Einwohner?
2. Welche beiden Städte haben die selbe ökonomische Aktivität?
3. Welche Städte grenzen an einen Wald?
4. Welche Routen von Paris nach Nizza passieren zunächst eine See und verlaufen dann durch einen Wald?
5. Welche Straßen verlaufen nicht durch Städte mit mehr als 100.000 Einwohnern?

Die erzeugten visuellen Anfragen können in verschiedene textuelle Sprachen übersetzt werden, u.a. „GeoSQL“. Schließlich bewerten die Autoren ihren Ansatz selbstkritisch ([CM94]):

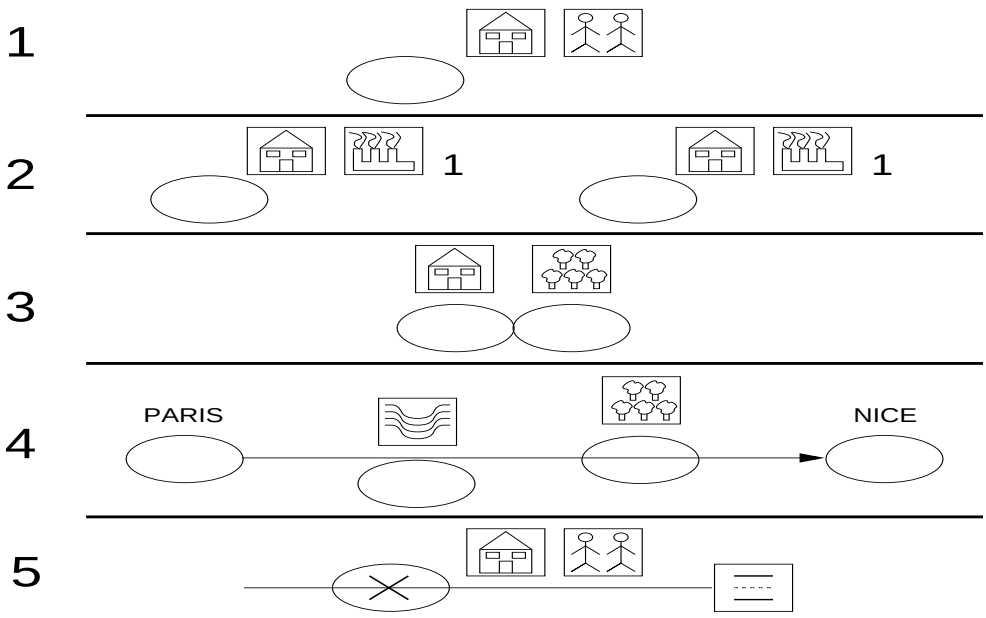


Abbildung 4.8: Ikonische Anfragen mit CIGALES (reproduziert nach [CM94])

> In spite of the user-friendliness of the visual approach, the representation of a query may be confusing. ... On the other hand, the query area present various difficulties due to the dynamics of the drawing in this area. The problem results either from the management of the several subqueries or from the interactions between the subqueries. ... Nevertheless, the graphical representation of a query may look ambiguous for the user. Providing an explanation facility might be a possible solution to avoid this problem.

Wird jedoch eine Erklärungskomponente benötigt um die Semantik einer visuellen Anfrage zu verstehen, so erscheint es fraglich, ob dann große Vorteile durch die visuelle Repräsentation derartiger Anfragen entstehen.

4.3.4 „Spatial-Query-by-Sketch“

Bei „Spatial-Query-by-Sketch“ ([Ege96b]) handelt es sich ebenso wie „Sketch!“ um eine VSQL, die auf der Skizzen-Metapher basiert (s. Abb. 4.9). Im Gegensatz zu den anderen drei Sprachen ist diese Sprache noch nicht implementiert.[4] Egenhofer sieht vor, die Vorteile und Möglichkeiten eines „Pen Pads“ (ähnlich einem Apple Newton) bei der Anfragegestaltung zu nutzen: im Gegensatz zu „Sketch!“ von Meyer ist hier tatsächliches Freihandzeichnen als Eingabeform vorgesehen, so daß das System potentiell viele Mehrdeutigkeiten handhaben muß (s. Kap. 2). Das Problem wird von Egenhofer jedoch nicht weiter analysiert; stattdessen sollen Mehrdeutigkeiten vom System erkannt werden und durch einen vom System geführten *Interview-Prozeß* vom Benutzer disambiguiert werden.

Egenhofer stellt insbesondere die Natürlichkeit der Skizzen-Metapher zur räumlichen Anfrageformulierung heraus. Er argumentiert, daß der Benutzer einer räumlichen Anfragesprache zunächst ein mentales Bild der aufzufindenden Konstellation im Kopf habe, bevor er mit der Formulierung der Anfrage beginne. Der Prozeß der Transformation des mentalen Bildes auf die Konstrukte der räumlichen Anfragesprache wird umso kürzer, je ähnlicher die in der Anfragesprache zur Verfügung

[4] Da es sich um ein *sehr* ambitioniertes Vorhaben handelt, ist es fraglich, ob dies beim heutigen Stand der Technik überhaupt gelingt.

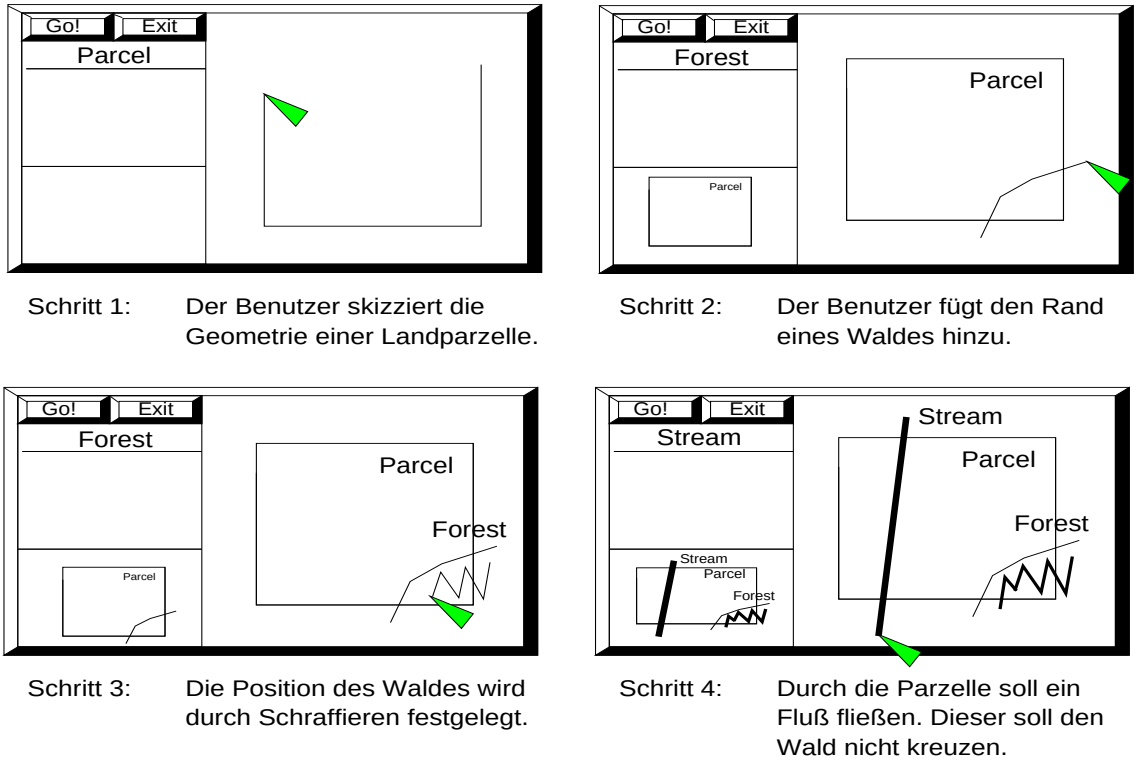


Abbildung 4.9: Erstellung einer freihandgezeichneten Skizze (reproduziert nach [Ege96b])

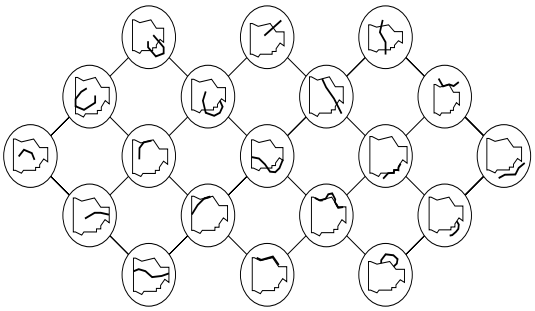
gestellten räumlichen Konzepte den mentalen Bildern sind. Die unmittelbare Rückmeldung (Feedback) durch die inkrementell enstehende Skizze wird dabei positiv hervorgehoben ([Ege96b]):

> Sketching provides immediate graphical feedback and, therefore, is an inherently more natural process to formulate many spatial constraints than a textual language.

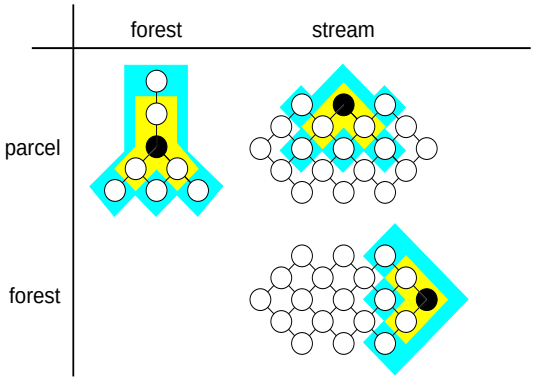
Die Interaktion mit dem System geschieht in fünf „logischen“ Phasen (die aber nicht notwendigerweise linear durchlaufen werden müssen):

1. Skizzieren der Anfrage.
2. Zusätzliches annotieren der Skizze, um weitere thematische oder metrische Bedingungen festzulegen (in [Ege96a] wird sogar Spracheingabe vorgesehen).
3. Vektorisieren und Interpretieren (Visual Parsing) der Skizze. Hier werden die implizit in der Skizze repräsentierten topologischen Relationen expliziert. Zwischen zweidimensionalen Objekten werden die in Kap. 2 diskutierten acht Egenhofer-Relationen sowie 19 topologische Relationen zwischen eindimensionalen Kurven (Linien) und zweidimensionalen Gebieten erkannt (s. Abb. 4.10(a)).
4. Erstellung eines Anfragebearbeitungsplanes. Mehrdeutigkeiten werden durch die Interview-Komponente im Dialog mit dem Benutzer geklärt.
5. Ausführen des Planes. Die erhaltenen Anfrageergebnisse werden anhand eines *Ähnlichkeitsmaßes* sortiert (s.u.).

Egenhofers Vorschlag ist der einzige, der eine formale Definition der verwendeten topologischen Relationen enthält (s. Kap. 2). Zusätzlich zu den acht Gebiets-Relationen werden 19 mögliche



(a) Konzeptuelle Nachbarschaften
für topologische Linie-Gebiet-Relationen



(b) Relaxierung der Anfrage

Abbildung 4.10: Topologische Relationen zwischen Linien und Gebieten und Relaxierung von Anfragen (reproduziert nach [Ege96b])

topologische Relationen zwischen Linien (Kurven) und Gebieten erkannt. Diese können ebenfalls durch die „9-Intersection"-Matrix formal definiert werden. Ebenso wie die Gebiets-Relationen sind diese in einer konzeptuellen Nachbarschaftsstruktur angeordnet, s. Abb. 4.10(a).

Desweiteren wird das Problem der *Aufweichung* oder *Relaxierung* topologischer Einschränkungen adressiert: selbst die topologische Beschreibung der der Skizze am ähnlichsten sehenden Konstellation im Datenbestand kann sich noch von der topologischen Beschreibung der Skizze selbst unterscheiden. Da die Beschreibungen nicht identisch sind, würde in diesem Fall auch kein Anfrageergebnis geliefert. Selbst die relativ schwach diskriminierenden topologischen Bedingungen sind in diesem Falle also noch „zu stark" – hier hilft dann eine Aufweichung der „Stärke" der vorhandenen Bedingungen (Relaxierung). Die Relaxierung geschieht nun so, daß im Falle eines leeren Anfrageergebnisses für jede Relation auch die konzeptuellen Nachbarn der Relation betrachtet werden; die Anfrage wird also (intern) modifiziert und in dieser generalisierten Form erneut gestellt (s. Abb. 4.10(b)). Wird wieder kein Ergebnis gefunden, so kann die Relaxierung nach selbigem Prinzip fortschreiten. In Abb. 4.10(b) ist eine zweistufige Relaxierung eingezeichnet.

Schließlich werden die gefundenen Anfrageergebnisse unter Verwendung eines *Ähnlichkeitsmaßes* sortiert: diese Funktion bewertet die Ähnlichkeit einer aufgefundenen Konstellation zur erstellten Skizze. In die Bewertungsfunktion werden u.a. auch diverse metrische Aspekte eingebracht. Das zugrundeliegende Verarbeitungsmodell für die Auswertung der skizzierten Anfrage ist lt. Egenhofer somit „Topologie zählt, Metrik verfeinert" („Topology matters, metric refines"). Leider ist der Aufbau des Ähnlichkeitsmaßes nicht sonderlich transparent und erscheint relativ willkürlich gewählt. Die unterschiedlichsten metrischen Attribute werden hierbei miteinander verrechnet.

Festzuhalten ist, daß „Spatial-Query-by-Sketch" multimodale Interaktionsformen (u.a. sogar Spracheingabe) vorsieht und die einzige (zumindest auf dem Papier existierende) VSQL ist, die auch von der Anfrage abweichende Anfrageergebnisse durch Relaxierung der Anfrageeinschränkungen bzw. Bedingungen zuläßt. Sowohl die verwendeten topologischen Relationen als auch der Begriff der Relaxierbarkeit über konzeptuelle Nachbarschaften sind mathematisch-präzise definiert. Der Interpretationsprozeß der Skizze seitens des Systemes scheint jedoch nicht sonderlich transparent: dies gilt insbesondere für die 19 vom System erkannten Linien-Gebiets-Relationen (der Benutzer muß diese auswendiglernen und kennen, wenn es keine Diskrepanzen zwischen benutzerintendierter und systeminferierter Bedeutung geben soll) und das Ähnlichkeitsmaß. Da das Ähnlichkeitsmaß lediglich zur Priorisierung der Anfrageergebnisse verwendet wird, ist dies weniger schwerwiegend. Egenhofer betont, daß weder Formähnlichkeit von Objekten noch relative Orientierungen zwischen Objekten in die Ähnlichkeitsfunktion eingehen sollten, was schwer nachvollziehbar erscheint.

Kapitel 5

Eine visuelle Sprache zur Definition räumlicher Konstellationen

In diesem Kapitel möchte ich die im Rahmen der Arbeit entwickelte (und teilweise implementierte, s. Kap. 6) visuelle Sprache VISCO vorstellen. VISCO ist eine Sprache zur Definition räumlicher Konstellationen. In diesem Kapitel wird die Sprache informell vorgestellt; im Anhang dieser Arbeit ist dann ein formaler Definitionsversuch zu finden.

Eine Reihe von Beispielen demonstriert die Nützlichkeit der einzelnen Sprachkonstrukte und ihr Zusammenwirken. Besonderes Gewicht wird auf einige Beispiele aus der Stadtkartendomäne gelegt – hier wurde die Einsetzbarkeit der Sprache durch Experimente validiert (s. auch Kap. 7).

Ich versuche im folgenden, die Sprache unabhängig von einer entsp. Sprachumgebung zu betrachten. Teilweise ist diese Trennung jedoch nicht sinnvoll. VISCO sollte als *visuelles System* betrachtet werden (s. Kap. 2). Mit einem visuellen System wird über eine visuelle Sprache kommuniziert (s. Kap. 2): die Aspekte dieser Kommunikation, welche losgelöst vom System selbst diskutiert werden können, werden hier dargestellt. Der Prototyp verlangt jedoch umfangreiche weitere Einschränkungen (s. Kap. 6): u.a. werden kongruente Punkte und Strecken verboten, und Komplexobjekte mit gemeinsamen Komponentenobjekten müssen stets zumindest eine nichtgemeinsame Komponente haben. Derartige Festlegungen würden hier jedoch zur Überspezifikation führen. Da letztlich ein Graphenabgleich (Graphmatching) zwischen VISCO-Objekten und dem räumlichen Datenbestand stattfinden soll, ist klar, daß beide Graphen die gleiche Struktur haben müssen (s. auch Kap. 3, „Spaghetti“- vs. topologisch strukturierte Vektordaten).

Zu den einzelnen Abbildungen ist zu sagen, daß sie nur dann vollständige und legale VISCO-Definitionen darstellen, wenn dies explizit erwähnt wird oder die Abbildung im Unterkapitel „Beispiele“ auftaucht. Nicht jedes hier vorgestellte Sprachelement ist in seiner vollen Funktionalität implementiert (s. Kap. 6), und einige der diskutierten Beispiele werden vom Prototyp nicht akzeptiert, da er nur eine entscheidbare Teilmenge der Sprache implementiert.

5.1 Allgemeines

VISCO (Vivid Spatial Constellations, belebte räumliche Konstellationen) ist eine visuelle Sprache zur Definition räumlicher Konstellationen.

Eine Konstellation wird durch Konstruktion definiert, wozu die von der Sprache angebotenen Primitive (Sprachelemente) verwendet werden. Die entstehenden Definitionen (oder auch Diagramme,

visuelle Sätze etc.) könnten aufgrund ihrer Semantik vielfach genutzt werden. Es wird die Sichtweise vertreten, daß die verwendeten VISCO-Sprachelemente räumliche Objekte der interessierenden Domäne *repräsentieren* können. Diese Domäne könnte z.B. der Datenbestand eines GIS sein. VISCO kann als Anfragesprache verwendet werden, wenn man verlangt, daß all die Objekte des Datenbestandes bzw. der Domäne aufgefunden werden sollen, die durch die VISCO-Objekte der Definition bzw. „Anfrage" repräsentiert werden. Es handelt sich dann sozusagen um die *Extension* des räumlichen Konzeptes, welches mit VISCO *intensional definiert* wurde. In der Regel kann ein VISCO-Objekt ganze Klassen von Objekten der Domäne (und somit räumliche Klassenbegriffe) repräsentieren. Die Interpretation von VISCO als visuelle Anfragesprache ist also nicht zwingend – es sind jedoch einige Konstrukte speziell für diesen Anwendungszweck vorhanden.

VISCO kann also als visuelle räumliche Anfragesprache genutzt werden, wenn man die *vom Benutzer konstruierte Konstellation als prototypisches Beispiel einer aus einem räumlichen Datenbestand zu gewinnenden Konstellation ansieht (insofern handelt es sich um eine Art „Query-by-Visual-Example"-Sprache, s. auch Kap. 4).* In der Regel beschreibt eine VISCO-Definition nicht eine Konstellation, sondern eine ganze Klasse möglicher Konstellationen. Wenn die Rede von „abzugleichenden Objekten" oder „zu bindenen Objekten" ist, so wird damit zum Ausdruck gebracht, daß Objekte der VISCO-Definition gegen Objekte des räumlichen Datenbestandes bzw. der Domäne *gebunden* werden sollen (ähnlich einer Variablen), bzw. ein *Strukturabgleich* geschehen soll (Matching). Ein VISCO-Objekt kann genau dann gegen ein Domänenobjekt gebunden werden, wenn es dieses auch repräsentiert.

VISCO ist als Experiment zu betrachten und momentan noch nicht auf eine konkrete Anwendungsdomäne hin zugeschnitten. Daher fehlen der Sprache auch bestimmte anwendungsspezifische Sprachmittel und Operatoren, wie sie z.B. im GIS-Bereich benötigt würden. Offensichtlich verwalten GIS aber räumliche Daten – gemeinsamer Bezugspunkt ist hier wieder das Attribut „räumlich", denn VISCO ist eine Sprache zur Definition räumlicher Konstellationen. Die anwendungsspezifischen Aspekte dieser Konstellationen werden im weiteren Verlauf weitgehend ausgeklammert. In erster Linie interessieren hier die räumlichen Aspekte, also die Konstellationen selbst. Insofern sehe ich Anwendungsmöglichkeiten für verschiedenste räumliche Datenbestände: Während die Sprache in ihrer jetzigen Form nur eine Basis darstellt, sollte sie für spezielle Einsatzkontexte um anwendungsspezifische Sprachmittel ergänzt werden. Die Verwendbarkeit der Sprache wurde experimentell durch die Stadtkarten-Domäne validiert (s. Kap. 7).

Ich möchte darauf hinweisen, daß die Sprache nicht ohne eine stark unterstützende Benutzungsoberfläche verwendet werden kann – insofern ist es sinnvoller, von einem VISCO-System zu sprechen (s. Kap. 6).

VISCO ist als Experiment zu betrachten: Insbesondere wurde versucht, einige Techniken einzusetzen, die in der Literatur als kritisch bekannt sind (so z.B. die Verdeckungen, s.u.).

5.2 Entwurfsentscheidungen

Folgende (nicht disjunkte) Punkte wurden beim Entwurf der Sprache versucht zu berücksichtigen:

Vagheit: Unbestimmtheiten verschiedener Art sollten zugelassen werden, so z.B. der Form, der Position bzw. Lage, der Orientierung, der Größe, etc. Eine attraktive Möglichkeit, diese Vagheiten für den Benutzer erfahrbar zu machen, wäre eine *Animationskomponente:* in einer Art Film könnten z.B. rotierbare Objekte rotieren, skalierbare Objekte ihre Größe ändern, formvariable Objekte ihre Form ändern, etc.

Ausdrucksmächtigkeit: Sowohl topologische als auch geometrische und metrische Beschränkungen sollten formulierbar sein – in einigen Kontexten kann es relevant sein, z.B. einen Kreis von einem Rechteck zu unterscheiden, während die Form von Objekten in anderen Kontexten vielleicht irrelevant ist. Offensichtlich ist es wiederum nur eine Frage der Wahl der

Interpretationsfunktion bzw. der relevanten Aspekte, wie die in einer bildlichen Darstellung ersichtliche Geometrie interpretiert wird. Da die Geometrie die Topologie (auf $\mathbb{R}^2$) impliziert, kann natürlich auch eine topologische Interpretation gewählt werden, welche in der Regel sehr viel stärker abstrahiert.

Abstraktionsmöglichkeiten: Wie lassen sich nichtintendierte räumliche Relationen und Eigenschaften von intendierten unterscheiden – m.a.W., welche Möglichkeiten zur Forumulierung räumlicher „Don't Care"-Bedingungen gibt es (s. auch Kap. 4)?

Verständlichkeit und Transparenz: Hier könnten Metaphern eine tragende Rolle spielen – Voraussetzung ist jedoch, daß eine Reihe geeigneter Metaphern gefunden wird. Metaphern dürfen nicht überstrapaziert werden. Zudem sollten die einzelnen Sprachelemente möglichst orthogonal und ohne Sonderregeln kombinierbar sein. Die Bedeutung einer Anfrage sollte sich aufgrund sehr weniger elementarer Regeln ergeben.

Prinzipiell besteht eine VISCO-Definition aus einer *Sequenz von Diagrammen (Def. „Diagramm" s. Kap. 2)*, wobei die Sequenz die zeitliche Abfolge der Konstruktionsschritte widerspiegelt. Einen ersten Eindruck vom Erscheinungsbild der Sprache vermittelt die Definition in Abb. 5.1: hier wird das räumliche Konzept „Eine Kirche in der 'Nähe' einer U-Bahn-Station" konstruiert (die U's an den oberen Rändern haben keinen Bezug zur U-Bahn-Station, sondern stehen für Universum, s.u.). In der Definition in Abb. 5.2 soll die Kirche „östlich und in der Nähe" der U-Bahn-Station liegen. Man beachte, daß die natürlichsprachlichen Umschreibungen die in diesen Definitionen ersichtlichen Bedingungen (s.u.) nur sehr unzureichend charakterisieren. Während in Def. 5.1 „in der Nähe" noch durch „im Umkreis von 220 m" präzisiert werden könnte, ist eine äquivalente natürlichsprachliche Beschreibung des in Abb. 5.2 dargestellten räumlichen Konzeptes schlichtweg unmöglich. Natürlichsprachliche Präpositionen sind inhärent vage – ihre Bedeutung muß jedoch im Rahmen einer räumlichen Anfragesprache eindeutig festgelegt werden. Bereits bei der Diskussion der erweiterten SQL-Dialekte wurde erwähnt, daß die Bedeutungen von in diesen Sprachen eingebetteten räumlichen Prädikaten wie „adjazent", „westlich" etc. von Spracherweiterung zu Spracherweiterung differieren. Derartige Probleme können bei einer direkten Modellierung räumlicher Konzepte durch räumliche Konzepte selbst nicht auftreten, da keine textuell-sprachlichen Umschreibungen dieser Konzepte benötig werden.

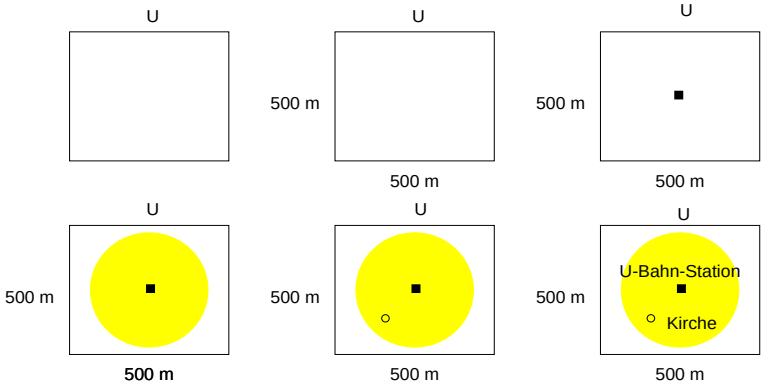


Abbildung 5.1: Eine Kirche in der Nähe einer U-Bahn-Station

Jeder Konstruktionsschritt erzeugt also ein neues Diagramm: die Semantik einer VISCO-Definition ergibt sich schrittweise und kann (im allgemeinen) auch nur schrittweise rekonstruiert werden (denn es können Verdeckungen auftreten, s.u.). Die Sequenz von Diagrammen wird am bequemsten mit einem syntaxgesteuerten Grafikeditor erzeugt, bzw. mit einer den Benutzer aktiv unterstützenden Oberfläche konstruiert (s. Kap. 6, Kap. 7).

Soll die Semantik einer VISCO-Definition rekonstruiert werden, so muß die komplette Konstruktionssequenz vorliegen, da VISCO im Sinne der Definition aus Kap. 2 nicht visuell vollständig ist;

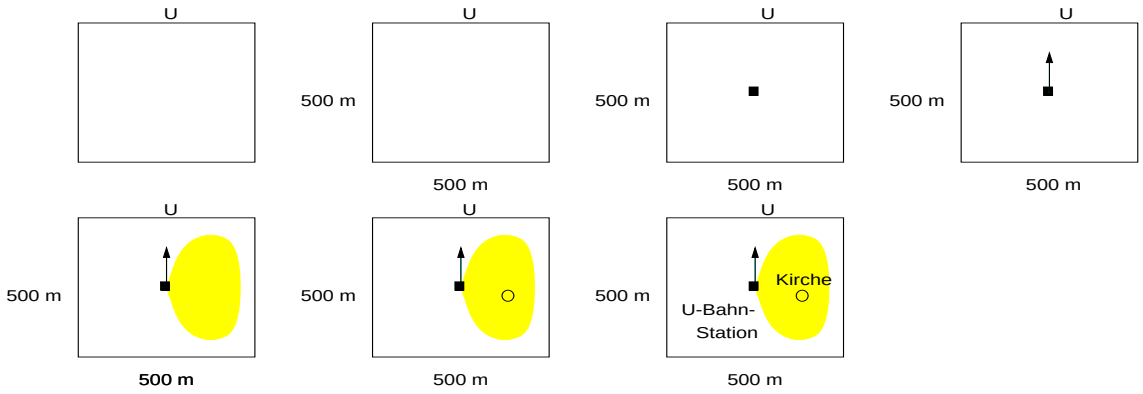


Abbildung 5.2: Eine Kirche in der östlichen Nähe einer U-Bahn-Station

m.a.W., anhand des letzten Diagrammes kann die Bedeutung einer VISCO-Definition i. allg. nicht rekonstruiert werden. Betrachtet man jedoch die Entstehungsgeschichte der Definition (den Konstruktionsprozess) als visuellen Satz (und nicht nur das letzte Diagramm), so ist VISCO prinzipiell visuell vollständig (schließlich könnte man ja alle Diagramme in *ein* Diagramm zeichnen).[1]

Prinzipiell gibt es nun zwei mögliche Konstruktionsschritte:

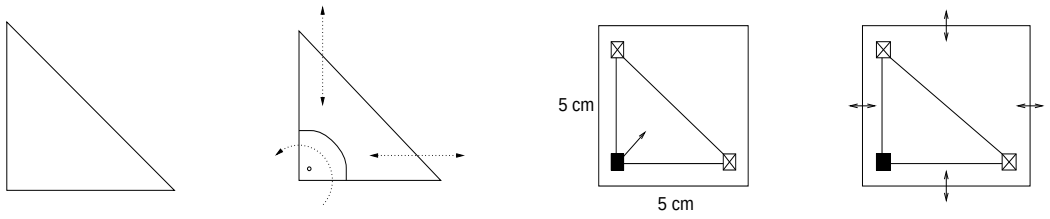
- Objekterzeugung und
- Metaobjekterzeugung (grafische Annotation).

In den obigen Bsp. sind die einzigen „Objekte“ der kleine Kreis und das kleine schwarze Rechteck, die mit „Kirche“ bzw. „U-Bahn-Station“ beschriftet sind. Alle anderen Objekte (Pfeile, Gebiete, Beschriftungen) sind Metaobjekte.

Sowohl die Objekterzeugung als auch die Metaobjekterzeugung erzeugen neue grafische Objekte im aktuellen Diagramm und somit das nächste Diagramm der Sequenz. Während jedoch die *Objekte* an sich für interessierende Objekte der Domäne (z.B. Geo-Objekte in einem GIS, wie die Kirche oder die U-Bahn-Station in Abb. 5.1) stehen, dienen *Metaobjekte* dazu, *visuelle Anzeichen für eine bestimmte Interpretation des Diagrammes zu liefern.* Sie repräsentieren somit nicht Objekte der Domäne, sondern machen Aussagen *über* die anderen Objekte bzw. ihre Interpretation. Meistens liefern sie Anzeichen für das Vorhandensein zusätzlicher Einschränkungen (Constraints), oder relaxieren implizit vorhandene Einschränkungen. So wird z.B. die Position eines Punktes durch ein *Gebiet* aufgeweicht, und die feste Position durch eine „Enthalten in“-Bedingung ersetzt – dies ist z.B. für die Kirche in obigen Beispielen der Fall. Auch das grafische Textobjekt „Kirche“ ist ein Metaobjekt: seine Anwesenheit ermöglicht die Interpretation des kleinen Kreises als punktförmigen Repräsentanten einer Kirche.

Während ein Dreieck zunächst nur für sich selbst steht (dies ist die natürlichste Interpretation, s. Abb. 5.3(a)), kann ein metagrafisch annotiertes Dreieck für die Klasse der rechtwinkligen, skalier- und rotierbaren Dreiecke stehen (Abb. 5.3(b)). Natürlich könnte man auch auf die metagrafischen Annotationen verzichten, um per Konvention stets diese Interpretation zu wählen – für einen uninformierten Betrachter gibt die annotierte Version jedoch Hinweise auf die Wahl der wahrscheinlich

[1] Der syntaxgesteuerte Grafikeditor des Prototypen (s. Kap. 6) zur Konstruktion von VISCO-Definitionen ist mit einer „Stöber-Komponente“ (Browser) versehen, durch deren Benutzung die komplette Konstruktionshistorie inspiziert werden und somit die Semantik einer Definition schrittweise rekonstruiert werden kann. Stöbern (Browsing) wird oft als Methode zur Übersichtsgewinnung in großen Datenbeständen bezeichnet – schrittweise kann der Benutzer durch gezielte Inspektion einzelner Fragmenten einen Überblick über das Ganze bekommen. Da es sich bei einem Diagramm auch um eine große Datenquantität handelt und der Mensch Fokusierungstechniken anwendet, um es portionsweise zu interpretieren, bietet eine Browsing-Komponente die Möglichkeit, ihn hierbei zu unterstützen, indem momentan irrelevante Informationen bzw. Teile des Diagrammes ausgeblendet werden.



(a) (b) (c) (d)

Abbildung 5.3: Dreiecke

richtigen Bedeutung des Bildes (s. auch die Diskussion von Ikonen in Kap. 2 und 4). Die Konzepte aus Abb. 5.3(a) und 5.3(b) könnten in VISCO wie in Abb. 5.3(c) und 5.3(d) angegeben werden.

Da VISCOs Sprachelemente geometrische Objekte sind (Punkte, Linien, Polygone etc.), wird die Geometrie dieser Objekte auch ernst genommen, d.h., es wird primär eine geometrische Interpretationsfunktion verwendet. Dies unterscheidet die Sprache von den in Kap. 4 vorgestellten visuellen räumlichen Anfragesprachen, die stets eine topologische Interpretation der dargestellten Geometrie der visuellen Anfragen wählen. Insbesondere bei Egenhofer wurde jedoch deutlich, daß auch geometrische Eigenschaften und Relationen von Belang sind – durch metrische Verfeinerungen von topologischen Relationen nähert er sich wieder der Geometrie.

Während man diese Vorgehen als „Top-Down“ (vom Abstrakten zum Konkreten bzw. von der Topologie über metrische Verfeinerungen zur Geometrie) bezeichnen könnte, wird hier ein „Bottom-Up“-Ansatz vertreten: die Geometrie der Anfrage wird zunächst ernst genommen, kann jedoch durch metagrafische Annotationen vom Benutzer *aufgeweicht (relaxiert)* werden. Diese Aufweichungen können so stark werden, daß nahezu topologische Beschreibungen entstehen. Dieser Ansatz ist also in gewisser Weise diametral.

5.2.1 Geometrische Objekte

Zunächst sollen die Objekte betrachtet werden. Objekte in VISCO sind in erster Linie *geometrische Objekte* – durch Komposition dieser werden schrittweise Konstellationen (oder Diagramme) gebildet. Jedes Objekt ist mit einer *physikalischen Metapher* versehen – Objekte in VISCO sind in zweiter Linie *physikalische Objekte.*

Komplexe geometrische Objekte

Objekte in VISCO sind Komplexe, die aus Komponentenobjekten aufgebaut werden. So bestehen

- Aggregate aus Ansammlungen beliebiger VISCO-Objekte,
- Polygone und Ketten aus Segmenten bzw. Strecken und schließlich
- Segmente bzw. Strecken aus Endpunkten.

Die Endpunkte eines Segmentes, die Segmente eines Polygones oder einer Kette und die Objekte eines Aggregates werden auch als Komponentenobjekte des Objektes bezeichnet. Komponentenobjekte sind vollwertige Objekte. Ein Objekt, welches nicht Komponente irgendeines anderen ist, wird *Primärobjekt* genannt. Als Besonderheit sei darauf hingewiesen, daß Polygone zunächst *kein Inneres haben* – insofern sollte man vielleicht passender von geschlossenen Ketten reden. Bei Bedarf muß ihr Inneres explizit gemacht werden durch ein sog. „Inneres Gebiet“.

Die Objekte stehen somit in einer „hat Komponente“-Beziehung und bilden (bzgl. dieser Partonomie) einen gerichteten azyklischen Graphen (DAG). Ob es sich vielleicht sogar um einen Wald von Bäumen handelt, wird hier zunächst nicht festgelegt, ebensowenig die Frage, inwieweit eine topologische Strukturierung (sowohl des Datenbestandes als auch der VISCO-Definition) vorliegen soll, oder ob es sich um ein „Spaghetti“-Vektordatenmodell handelt (s. Kap. 3). Ein Objekt kann (muß jedoch nicht) Komponente einer beliebigen Anzahl von Komplexobjekten sein. Für Aggregate gilt die noch stärkere Einschränkung, daß je zwei Aggregate keine gemeinsamen Komponentenobjekte haben. Weitere Einschränkungen gelten zunächst nicht. Jedes VISCO-Objekt (mit Ausnahme der Folien selbst) ist Komponente genau einer Folie.

Die in Kap. 2 diskutierten Visualisierungsprobleme bzw. die visuelle Nichtentscheidbarkeit best. Fragen („Handelt es sich um ein explizites oder um ein implizites Dreieck?“) wird in VISCO mit Hilfe eines (expliziten) ikonischen Repräsentanten für ein eigentlich implizit definiertes Objekt (im Bsp. das Dreieck) gelöst.

Physikalische Objekte

Eine Metapher für Aggregate: Zur Definition der verwendeten geometrischen Objekte wird stets ein *lokales kartesisches Koordinatensystem* benötigt. Dieses kann zunächst an jede mögliche Position der zweidimensionalen Welt bzw. interessierenden Domäne gesetzt werden kann. Bezüglich dieses *Weltkoordinatensystemes* kann das lokale Koordinatensystem also *translatiert,* aber bei Bedarf auch *skaliert* und *rotiert* werden. Eine Menge von Objekten, die bzgl. eines gemeinsamen lokalen Koordinatensystemes definiert werden, wird als Aggregat bezeichnet.

Für diese translatier-, evtl. skalier- und rotierbaren Aggregate wird die Metapher einer (skalierbaren) *Overheadprojektorfolie (kurz Folie)* verwendet: Objekte, die auf eine Overheadprojektorfolie gemalt bzw. konstruiert werden, können natürlicherweise bzgl. des Weltkoordinatensystems rotiert und auch translatiert werden. Diese Transformationen benötigen natürlichen einen Fixpunkt: der Fixpunkt ist stets der Ursprung der Folie, der gesondert dargestellt wird, in Abb. 5.1 und 5.2 als kleines schwarzes gefülltes Rechteck. Die Folien selbst werden als rechteckige, entsp. annotierte Rahmen dargestellt.

Soll nun z.B. Rotierbarkeit der Folie verboten werden, so erscheint es sinnvoll, dies durch zusätzliche Metaobjekte explizit zu machen. In Abb. 5.2 kann nur dann von „östlich“ geredet werden, wenn die Folie bzgl. des Weltkoordinatensystemes nicht rotiert werden darf. Dies wird durch den *kleinen Pfeil am Ursprung der Folie (kleines schwarzes Rechteck)* visualisiert.

Die Skalierbarkeit einer Overheadprojektorfolie ist zwar physikalisch nicht gegeben, die Metapher wird aber dennoch verwendet. Wie alle Metaphern hat auch diese ihre Grenzen.

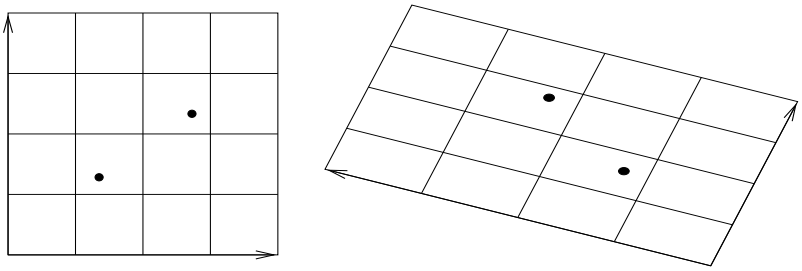
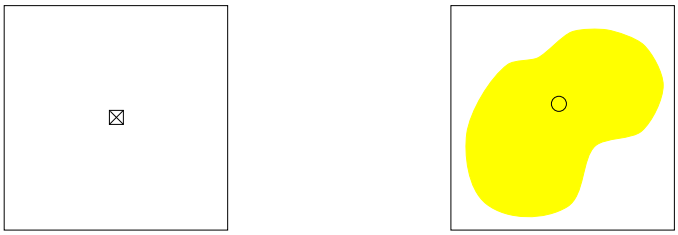


Abbildung 5.4: Transformationen

Jedes der bisher diskutierten Objekte muß also auf genau eine Overheadprojektorfolie „gemalt“ bzw. konstruiert werden. Overheadprojektoren sind durchsichtig, haben eine beliebige rechteckige Form und können (wie in der Realität) *aufeinandergestapelt* werden. Diese Eigenschaften werden noch von Relevanz sein (s.u.).

Es sollte noch erwähnt werden, daß die euklidische Metrik einer Folie durch die Transformationen

nicht verändert wird (s. Abb. 5.4). Somit sind lokale Positionen, Entfernungen, Orientierungen etc. bzgl. dieser Metrik invariant gegenüber derartiger Transformationen. Die beiden Punkte in Abb. 5.4 z.B. haben sowohl vor als auch nach der Transformation die selbe Entfernung.



(a) Quantitative relative Position (b) Qualitative relative Position

Abbildung 5.5: Punkte

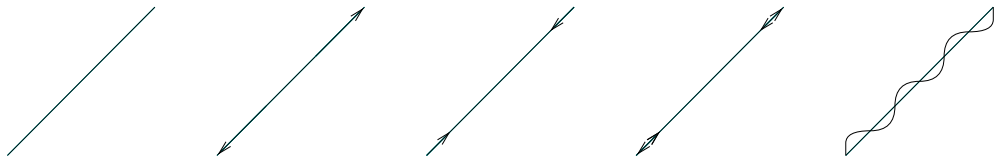
Eine Metapher für Punkte: Betrachtet man nun einen Punkt, der auf einer Overheadprojektorfolie bzw. bzgl. eines lokalen Koordinatensystemes definiert wird, so ergibt sich aufgrund der Geometrie zunächst natürlicherweise nur die identische bzw. natürliche Interpretation dieses Punktes als Repräsentanten eines Punktes, der bzgl. dieses Koordinatensystemes genau die visualisierte Position hat (s. Abb. 5.5(a) – die Folie selbst wird durch den rechteckigen Rahmen visualisiert). Ein solcher Punkt repräsentiert also Punkte mit bzgl. des lokalen Koordinatensystemes identischer Position. Da die Overheadprojektfolie stets translatierbar ist, wird somit bereits eine Klasse von Punkten in der Welt beschrieben. Hierbei handelt es sich also um *exakte bzw. quantitative relative Position* in Bezug auf das Koordinatensystem der tragenden Overheadprojektorfolie.

Oftmals will man aber auch *vage bzw. qualitative relative Positionen* formulieren können: hierzu muß also die Position eines Punktes bzgl. des lokalen Koordinatensystemes entsp. vage interpretiert werden. Malt man einen Punkt in ein *Gebiet*, so könnte man sich vorstellen, daß hierdurch die *möglichen Aufenthaltsorte* eines Punktes dargestellt werden. Läßt man die Fläche des Gebietes gegen Null gehen (im Grenzübergang), so wird deutlich, daß es sich bei dieser Interpretation um eine qualitative Generalisierung der Position zugunsten einer *vagen oder unscharfen Position* handelt (s. Abb. 5.5(b)). Genauso wurde ja auch die Position der „Kirche“ in obigen Beispielen interpretiert. Offensichtlich wird hier also die „hat Position (x,y)“-Einschränkung eines Punktes gegen eine „Enthalten in Gebiet“-Einschränkung ersetzt. Letztere kann als Disjunktion einer (überabzählbaren) Anzahl von Einschränkungen ersterer Art betrachtet werden.

Läßt man nun also verschiedene Interpretationen zu, so muß es eine Möglichkeit geben, anhand visueller Anzeichen die richtige Interpretation zu wählen. Eine *treffende Metapher* kann die Wahl der richtigen Interpretation (und somit der Bedeutung) nahezu zwingend machen: versieht man die Punkte nun mit einer physikalischen Bedeutung, so kann anhand der Eigenschaften dieser metaphorischen physikalischen Objekte auf die Eigenschaften der Repräsentanten geschlossen werden. Während viele Metaphern auf den ersten Blick etwas albern erscheinen, ist ihre Nützlichkeit dennoch unbestritten (s. Kap. 2).

Eine (sehr kleine) *Murmel* könnte ein punktförmiges Objekt repräsentieren, dessen Position innerhalb eines bestimmten eingezäunten Gebietes liegen darf. Die Murmel rollt umher und kann innerhalb dieses Gebietes jede Position einnehmen – dies ist in Abb. 5.5(b) der Fall. Ein *Nagel* hingegen hat diese Freiheiten nicht – er hat stets eine feste Position (auch wenn er in einem Gebiet liegen sollte), s. Abb. 5.5(b). Murmeln und Nägel werden zweidimensional visualisiert, so daß sie anhand ihrer Form *visuell unterschieden* werden können. Die sich daraus ergebende Problematik wird ignoriert (man bedenke, daß die verwendeten Visualisierungen andere räumliche Relationen eingehen können als echte Punkte – im VISCO-System könnte hier z.B. ein *Gitter* helfen. Die Maschenweite des Gitters müsste größer sein als die flächigen Ausdehnungen der für die Punkte

verwendeten Visualisierungen, so daß durch die Rasterung sichergestellt wird, daß die Visualisierungen von zwei nicht-identischen Punkten sich z.B. nicht „Schneiden“ können).



(a) (b) (c) (d) (e)

Abbildung 5.6: Strecken

Eine Metapher für Linien: Betrachtet man eine auf eine Overheadprojektorfolie gemalte Strecke als Repräsentanten eines eindimensionalen Objektes, so ergeben sich u.a. folgende Interpretationsmöglichkeiten:

1. Die Strecke repräsentiert alle Strecken *identischer* Länge (bzgl. der euklidischen Metrik des Koordinatensystemes). Die Orientierung des Repräsentanten könnte als Orientierung der repräsentierten Strecke verstanden werden (s. Abb. 5.6(a)).

2. Die Strecke repräsentiert alle Strecken, die *mindestens* die angegebene Länge haben. Die Orientierung des Repräsentanten könnte als Orientierung der repräsentierten Strecke verstanden werden (s. Abb. 5.6(b)).

3. Die Strecke repräsentiert alle Strecken, die *höchstens* die angegebene Länge haben. Die Orientierung des Repräsentanten könnte als Orientierung der repräsentierten Strecke verstanden werden (s. Abb. 5.6(c)).

4. Die Strecke repräsentiert alle Strecken, die eine *beliebige* Länge haben. Die Orientierung des Repräsentanten könnte als Orientierung der repräsentierten Strecke verstanden werden (s. Abb. 5.6(d)).

5. Die Strecke repräsentiert keine Strecke, sondern eine *beliebige topologische Linie bzw. Kurve bzw. einen Weg zwischen den beiden Endpunkten* (also eine topologische Struktur, die homöomorph zur dargestellten Strecke ist). Eigenschaften wie Orientierung und Länge des Repräsentanten haben dann keine Bedeutung (s. Abb. 5.6(e)). Im folgenden wird sogar verlangt, daß es sich bei der repräsentierten Kurve um eine Kette handelt (prinzipiell könnten Gummibänder aber auch z.B. Kreisbögen o.ä. repräsentieren).

Fall 1 kann z.B. durch einen „Holzstab“, Fall 2 durch eine „nur ausfahrbare Teleskopantenne“, Fall 3 durch eine „nur zusammenschiebbare Teleskopantenne“, Fall 4 durch eine „Teleskopantenne“ und Fall 5 durch ein „Gummiband“ metaphorisiert werden.

Die *Endpunkte* dieser physikalischen Objekte sollen jeweils „Murmeln“ oder „Nägel“ sein – offensichtlich ergeben sich hierdurch weitere komplexe Einschränkungen für die repräsentierten Objekte. Natürlich spielen auch die Eigenschaften der tragenden Folie eine wichtige Rolle.

Zusammenwirken der Metaphern: Polygone und Ketten werden nun aus Holzstäben, Teleskopantennen und Gummibändern zusammengesetzt, deren Endpunkte wiederum Nägel und Murmeln sind (s. Abb. 5.7, auch ein expliziter ikonischer Repräsentant für das gesamte Polygon selbst ist hier eingezeichnet, s. Kap. 2). Der Benutzer baut also eine Art „Maschine“ aus diesen physikalischen Grundbestandteilen zusammen. Die verwendete Metaphorik erlaubt es ihm, das

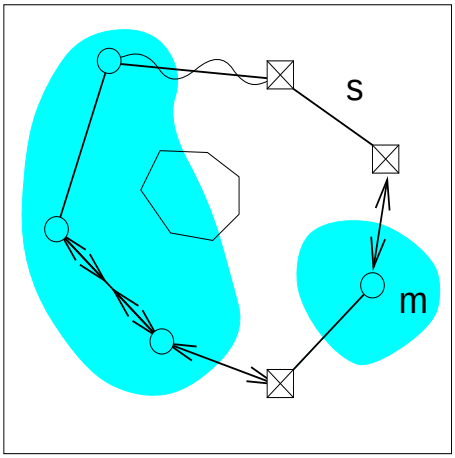


Abbildung 5.7: Definition einer komplizierten Klasse von Polygonen

Verhalten der Maschine anhand *mentaler Animation* zu verstehen, da das Verhalten der verwendeten physikalischen Objekte ihm aus der realen Welt bekannt ist (physikalisches Verständnis wird jedoch vorausgesetzt).

So kann der Benutzer z.B. inferieren, daß ein Holzstab, der zwischen zwei Nägeln befestigt ist, weder seine Orientierung noch seine „Position“ noch seine Länge (bzgl. der tragenden Folie) ändern kann (s. Strecke s in Abb. 5.7). Ein Holzstab zwischen einem Nagel und einer Murmel hingegen kann kreisförmig um den Nagel rotieren (s. Murmel m in Abb. 5.7; der mögliche Aufenthaltsbereich für m ist durch das grau schattierte Gebiet angedeutet).

Wie erwähnt, ist für das VISCO-System eine Animationskomponente vorgesehen (die jedoch noch nicht implementiert werden konnte).

Es folgt eine Übersicht über die bisher diskutierten VISCO-Objekte:

Nagel: Punkt mit bekannter Position bzgl. der tragenden Folie.

Murmel: Punkt mit vager Position bzgl. der tragenden Folie. Ein Murmel muß durch mindestens ein sogenanntes Gebiet (s.u.) eingegrenzt werden.

Gummiband: Beliebige Linie (homöomorph zur dargestellten Strecke) zwischen den beiden Endpunkten.

Holzstab: Strecke mit fester Länge (bzgl. der Metrik des Koordinatensystems der tragenden Folie).

Teleskopantenne: Strecke mit beliebiger Länge.

Zusammenschiebbare Teleskopantenne, kurz $\leq$-Teleskopantenne: Strecke mit höchstens der angegebenen Länge.

Ausfahrbare Teleskopantenne, kurz $\geq$-Teleskopantenne: Strecke mit mindestens der angegebenen Länge.

Overheadprojektorfolie, kurz Folie: Aggregat mit lokalem kartesischen Koordinatensystem. Jedes der obigen Objekte muß innerhalb der rechteckigen Grenzen der Folie bleiben, auf die es konstruiert wurde. Eine Folie kann translatiert, rotiert und skaliert werden. Hierfür lassen sich wieder Einschränkungen definieren, s.u.

Nägel und Murmeln werden als Punkte oder Punktkomponenten bezeichnet, und Holzstäbe und die verschiedenen Teleskopantennen als atomare Segmente oder Strecken. Gummibänder werden

auch nichtatomare Segmente bzw. Strecken (weil sie als Strecken visualisiert werden) genannt. Jedes der Objekte (mit Ausnahme der Folien) kann über die „hat Komponente“-Relation Komponente anderer Objekte werden, aber weder Punkte noch Segmente müssen notwendigerweise Komponentenobjekte von Segmenten bzw. Polygonen oder Ketten sein. Die einzigen Primärobjekte sind die Folien. Jedes Objekt ist Komponentenobjekt seiner tragenden Folie. Jedes Objekt (mit Ausnahme der Folien selbst) ist Komponente genau einer Folie.

5.2.2 Metaobjekte

Während die bisher diskutierten *Objekte* zur Repräsentation von interessierenden Objekten der Domäne dienen (ähnlich repräsentationalen Ikonen, s. Kap. 2), ist dies für Metaobjekte nicht der Fall. Eine spezielle Klasse der Metaobjekte sind die *geometrischen Hilfsobjekte*, eine andere Klasse von Metaobjekten wird durch die Objekte zur *metagrafischen Annotation* gebildet (z.B. Pfeile, Texte und Gebiete).

Geometrische Hilfsobjekte

Geometrische Hilfsobjekte dienen dazu, zusätzliche Beschränkungen für die *geometrischen Objekte* zu definieren. Es handelt sich bei ihnen also um spezielle Metaobjekte, aber auch um spezielle geometrische Objekte. Geometrische Hilfsobjekte sind nicht notwendigerweise im Datenbestand vorhanden. In der Klasse der geometrischen Hilfsobjekte sind u.a. alle *impliziten geometrischen Objekte des räumlichen Datenbestandes* enthalten. Jeder räumliche (Vektor-)Datenbestand muß zumindests Punkte und Strecken explizit machen – alles weitere (z.B. Polygone) ist evtl. jedoch nur implizit vorhanden. Unter Umständen will man z.B. drei Strecken im Datenbestand, die implizit (aber nicht explizit) ein Dreieck bilden, als Dreieck behandeln – dies setzt die *Erkennung* des Dreiecks voraus.[2] Für diesen Zweck kann in VISCO ein Polygon-Hilfsobjekt gebildet werden, wodurch das implizit im Datenbestand vorhandene Dreieck explizit als Polygon behandelbar würde. Dies geschieht durch Aggregierung der Linien. Dennoch ist das Dreieck nicht explizit als Polygon-Knoten im Datenbestand vorhanden – jedoch kann man sagen, daß dieses Dreieck bereits im *Universum der geometrischen Objekte explizit vorliegt.* Hierbei handelt es sich um die Menge aller geometrisch („wohlgeformten“) Objekte. Der aktuelle räumliche Datenbestand bzw. die zu repräsentierenden Objekte der interessierenden Domäne werden dann als spezielle (endliche) Teilmenge dieses Universums angesehen.

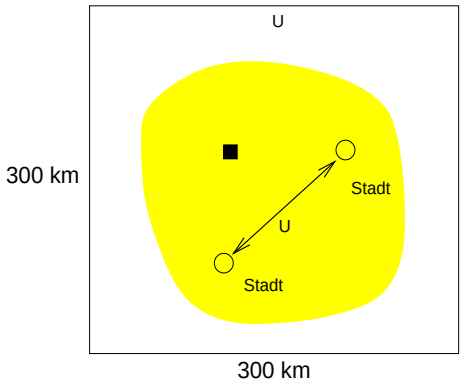


Abbildung 5.8: Verwendung eines geometrischen Hilfsobjektes

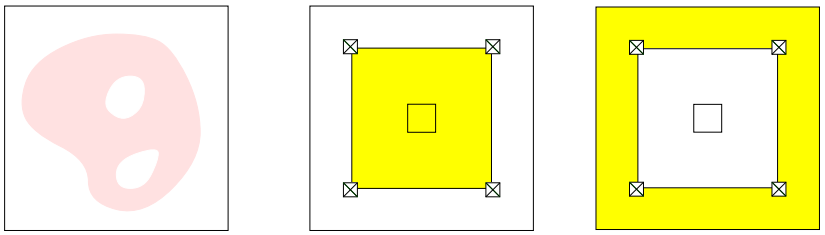
Hilfsobjekte sind aber nicht auf die Explizierung impliziter geometrischer Objekte des Datenbestandes beschränkt: so kann z.B. auch eine Strecke zwischen zwei im Datenbestand vorhandenen

[2]Eine solche Erkennung ist natürlich wesentlich aufwendiger, als lediglich bereits explizite Polygone auf Dreieckigkeit zu überprüfen.

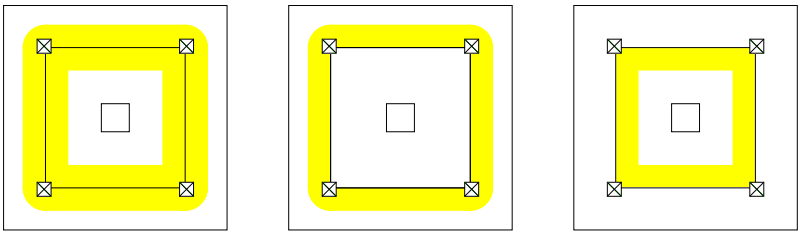
Punkten *konstruiert* werden (dies bedeutet nicht, daß die konstruierte Strecke in den Datenbestand aufgenommen wird!). Diese Strecke ist nicht einmal implizit im Datenbestand vorhanden, wiederum jedoch bereits explizit im Universum aller Strecken. In Abb. 5.8 wird so eine Mindestentfernung zwischen zwei Städten gefordert.

Während also geometrische Objekte einer VISCO-Definition geometrische Objekte im Datenbestand bzw. der interessierenden Domäne repräsentieren, können geometrische Hilfsobjekte Objekte im gesamten Universum geometrischer Objekte repräsentieren – somit umfassen sie auch erstere. Geometrische Hilfsobjekte werden im folgenden – wie in Abb. 5.8 – stets mit einem „U“ für „Universum“ beschriftet.

Gebiete



(a) Konstantes Gebiet (b) Inneres Gebiet (c) Äußeres Gebiet



(d) ϵ-Gebiet (e) $\epsilon^{\oplus}$-Gebiet (f) $\epsilon^{\ominus}$–Gebiet

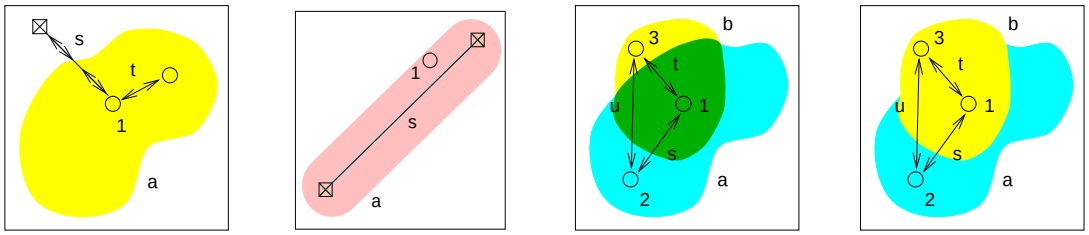
Abbildung 5.9: Typen von Gebieten

Eine andere Art von Metaobjekten sind die *Gebiete:* die Position eines Punktes kann aufgeweicht werden, indem er als Murmel innerhalb eines Gebietes repräsentiert wird. Innerhalb des Gebietes kann die Murmel jede Position einnehmen. Es gibt verschiedene Arten von Gebieten:

Skizzierte oder konstante Gebiete werden vom Benutzer gemalt. Sie sind unveränderlich bzgl. der tragenden Folie (s. Abb. 5.9(a)).

Berechnete oder abgeleitete Gebiete werden für ein spezielles Objekt erzeugt und sind somit von diesem abhängig. So ist es z.B. möglich, das Innere oder Äußere eines Polygones (s. Abb. 5.9(b,c)) sowie die ϵ-Umgebung eines Objektes als Gebiet zu erzeugen (s. Abb. 5.9(d,e,f)). Sind die Argumente z.B. Form- oder Positionsvariabel, so auch die abgeleiteten Gebiete.

Abgeleitete Gebiete werden von den skizzierten Gebieten visuell unterscheidbar dargestellt (in einer anderen Farbe o.ä.). Gebiete können *transparent* oder *opak (undurchsichtig)* sein – ein undurchsichtiges Gebiet kann bereits erzeugte Objekte (partiell oder total) verdecken. Hierbei handelt es sich um einen nützlichen Abstraktionsmechanismus (s.u.) von nichtintendierten räumlichen Relationen. *Generell erzeugt ein Gebiet die topologische „Enthalten in“-Einschränkung für alle Objekte, die vollständig sichtbar in ihm enthalten sind.*



(a) (b) (c) (d)

Abbildung 5.10: Opake und transparente Gebiete

Einige Beispiele zeigt Abb. 5.10:

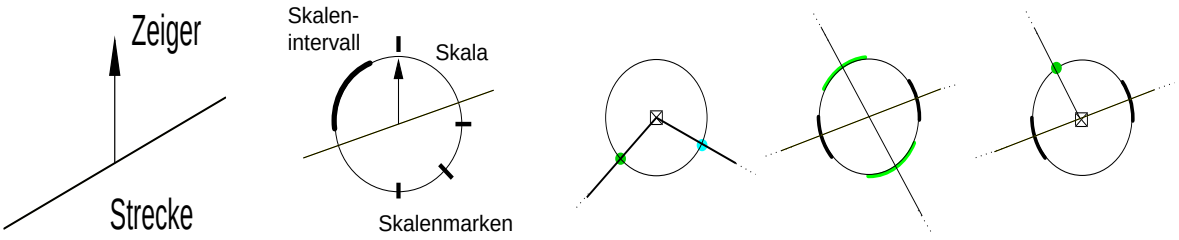
(a) Murmel 1 und Strecke t müssen innerhalb des konstanten Gebietes a enthalten sein (man beachte, daß a nicht konvex ist – daher impliziert das Enthaltensein der Endpunkte von t nicht das Enthaltensein von t selbst!), nicht jedoch der Nagel und die Strecke s.

(b) Murmel 1 muß innerhalb der $\epsilon(10)$-Umgebung a der Strecke s enthalten sein.

(c) Murmel 1 muß sowohl in a als auch in b, Murmel 2 in a, Murmel 3 in b enthalten sein – das transparente Gebiet b überlappt Gebiet a, so daß Gebiet a hindurchscheint. Strecke s muß in a und Strecke t in b enthalten sein. Strecke u muß weder in a noch b enthalten sein.

(d) Hier verdeckt das undurchsichtige Gebiet b das Gebiet a partiell, so daß die „Enthalten in“-Bedingung für Gebiet a stellenweise nicht sichtbar ist. Daraus folgt, daß Murmel 1, 3 und Strecke t in b enthalten sein müssen und Murmel 2 in Gebiet a. Die Strecken s und u müssen weder in a noch in b enthalten sein.

Zeiger, Skala und Winkeleinschränker



(a) Zeiger (b) Skala (c) Winkeleinschränker

Abbildung 5.11: Zeiger, Skala und Winkeleinschränker

Zeiger und Skala: Ein *Zeiger* kann an einem atomaren Segment befestigt werden, und eine *Skala* an einem solchen Zeiger (ein atomares Segment ist, s. S. 103, eine der drei Teleskopantennen oder ein Holzstab): *der Zeiger beschränkt die möglichen Orientierungen (bzgl. des Koordinatensystemes der tragenden Folie) dieses Segmentes.* Der Zeiger behält dabei stets die gezeigte relative Orientierung zum Segment bei (wie ein Normalenvektor, der immer senkrecht steht; der Zeiger muß jedoch nicht unbedingt senkrecht angebracht werden). Auf der (evtl. vorhandenen) kreisrunden Skala des Zeigers sind in Form von Skalenmarken und Skalenintervallen die möglichen erlaubten Stellungen des Zeigers angegeben – die Skala behält stets ihre eingezeichnete Orientierung zur Folie bei, während der Zeiger sich bei Orientierungsänderungen des Segmentes ja „mitdreht“ (insofern

besteht Ähnlichkeit zu einem „inversen“ Kompaß – die Skala orientiert sich also immer Richtung „Norden“ bzgl. der tragenden Folie).

Ein Zeiger *ohne* Skala steht für genau eine mögliche Orientierung (nämlich die dargestellte).

Zeiger und Skala können auch am sogenannten Ursprung einer Folie angebracht werden (s.u.) – so wird die Rotierbarkeit der ganzen Folie eingeschränkt.

Winkeleinschränker: Der *Winkeleinschränker* bezieht sich auf die *relative Orientierung zweier atomarer Segmente:* er schränkt den *Winkel* zwischen diesen auf einen festen Wert oder ein Toleranz-Intervall ein – die Visualisierungen (s. Abb. 5.11(c)) sprechen für sich selbst.

Weitere Metaobjekte

Andere grafische Metaobjekte visualisieren z.B. die Thematik eines Objektes (z.B. „Haus“), Skalierbarkeit von Folien (Pfeile), etc.

5.2.3 Relationen und Beschränkungen

Mit dem vorhandenen Inventar an Objekten lassen sich bereits sehr vage (bzw. große Klassen von) Konstellationen definieren. Nun bietet es sich an, wieder zusätzliche Einschränkungen formulieren zu können: so sollte es z.B. forderbar sein, daß sich die Ränder der beiden formvariablen Polygone in Abb. 5.12 nicht schneiden dürfen. Wie bereits erwähnt, haben Polygone zunächst kein Inneres.

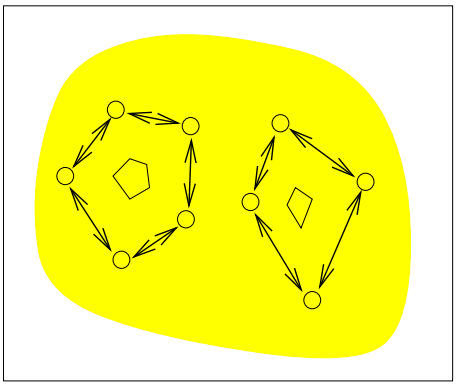


Abbildung 5.12: Zwei Polygone, deren Ränder sich nicht schneiden sollen

In diesem Fall benötigt man eine zusätzliche „Schneidet nicht“-Bedingung zwischen den Rändern, wodurch sich überlappende oder berührende Polygone ausgeschlossen werden (nicht jedoch einander enthaltene Polygone, da ausschließlich der *Rand* der Polygone berücksichtigt wird). Hierzu dienen *topologische Einschränkungen (Constraints),* die anhand der Geometrie einer VISCO-Definition abgeleitet werden: in der Geometrie ersichtliche räumliche Relationen werden also als notwendige Bedingungen interpretiert bzw. anhand der Geometrie abgeleitet. Eine topologische Einschränkung wurde bereits erwähnt: die „Enthalten in“-Bedingung zwischen Gebieten und geometrischen Objekten.

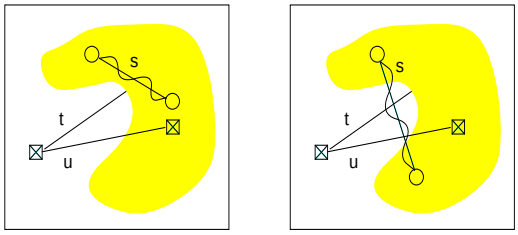
Abgeleitete Einschränkungen

Wird nun ein neues Objekt eingeführt, so werden anhand der implizit im Diagramm ersichtlichen topologischen Relationen Einschränkungen für dieses neue Objekt ermittelt: Wird z.B. eine neue Murmel in ein bereits vorhandenes Gebiet gemalt, so wird eine „Enthalten in“-Einschränkung für diese Murmel erzeugt (vorausgesetzt, das Gebiet ist an der Stelle der Murmel *sichtbar*, s.u.).

Die Möglichkeit, bereits definierte Objekte durch *undurchsichtige Gebiete verdecken zu können,* stellt einen natürlichen *Abstraktionsmechanismus* dar: bereits vollständig verdeckte Objekte können nicht mehr wahrgenommen werden. Sie haben somit keine Relevanz mehr für noch einzuführende neue Objekte, von ihrer Existenz wird vollständig abstrahiert. So werden weder räumliche Relationen zu ihnen etabliert, noch können Operatoren auf sie angewendet werden, noch können sie aggregiert werden. Ein neu eingeführtes Objekt ist hingegen stets vollständig sichtbar, da es zuoberst liegt.

Bei der Diskussion einiger visueller räumlicher Anfragesprachen in Kap. 4 wurde deutlich, daß ein Abstraktionsmechanismus von nichtintendierten bzw. unerwünschten räumlichen Relationen und Eigenschaften zur Verfügung stehen sollte. Meyer verwendet mehrere Visualisierungsebenen, um räumliche „Don't Care"-Relationen auszudrücken; Lee und Chin verwenden das Vordergrundkonzept (Foreground Concept): alle Objekte, zu denen für ein neues Objekt anhand der Geometrie räumliche Einschränkungen abgeleitet werden sollen, müssen zuvor explizit selektiert werden. Meyers Konzept der Ebenen ist relativ schwer zu durchschauen, da mehrere Repräsentanten ein und desselben Objektes in verschiedenen Ebenen auftauchen, wodurch die gesamte Konstellation zerstückelt wird und mental (anhand der einzelnen Ebenen) rekonstruiert werden muß. Das Vordergrundkonzept von Lee und Chin erscheint recht lästig, da jedesmal alle relevanten Objekte selektiert werden müssen. Zudem sind bedeutungstragende und nicht-bedeutungstragende räumliche Relationen alsdann ununterscheidbar.

Für VISCO stellt sich zudem die Frage, wie *partiell* verdeckte Objekte bezüglich räumlicher Relationen behandelt werden sollen (oder ob man ausschließlich totale Verdeckungen zulassen sollte), was im folgenden diskutiert wird:



(a) (b)

Abbildung 5.13: Abgeleitete topologische Einschränkungen

Einige einfache Regeln definieren nun, welche räumlichen Einschränkungen anhand welcher räumlicher Relationen *für ein neu eingeführtes Objekt* erzwungen werden. Zudem ist zu beachten, daß *jedes Objekt* berücksichtigt wird (auch Komponentenobjekte):

Schneidet nicht: aus einer sichtbaren „Schneidet nicht"-Relation wird genau dann eine „Schneidet nicht"-Einschränkung abgeleitet bzw. expliziert, wenn das andere Objekt vollständig sichtbar ist und keines der beiden Objekte als identitätslos bzw. Stellvertreter für sein Komplexobjekt betrachtet wird (s.u.). Eine „Schneidet nicht"-Einschränkung / Relation bezieht sich ausschließlich auf den Rand der Objekte. In Abb. 5.13(a) wird verlangt, daß das neu eingeführte Objekt Gummiband s Holzstab u nicht schneidet, denn u ist vollständig sichtbar. Zu Holzstab t wird keine „Schneidet nicht"-Einschränkung etabliert, da der eine Endpunkt von t unter dem Gebiet liegt und t daher nicht vollständig sichtbar ist. Tatsächlich wird zwischen s und t natürlich entweder ein (unsichtbarer) Schnitt oder nicht vorliegen – „ohne Gedächtnis" bzw. Darstellung der Konstruktionsgeschichte ist diese Frage jedoch nicht beantwortbar.[3]

[3]Tatsächlich kann im allgemeinen die Eigenschaft „ist vollständig sichtbar" nicht visuell entschieden werden, da die Entscheidung dieser Frage die Kenntnis von *Objektidentitäten* erfordert. In Kap. 2 wurden diesbzgl. diverse Probleme diskutiert, s. z.B. Abb. 2.22. Maßnahmen zur Sicherstellung der Entscheidbarkeit derartiger Fragen wurden jedoch im VISCO-Prototypen getroffen.

Schneidet: eine „Schneidet“-Einschränkung wird genau dann abgeleitet, wenn es mindesten einen sichtbaren Schnittpunkt mit dem anderen Objekt gibt (in diesem Fall gibt es ein sichtbares Anzeichen für das Vorliegen des Schnittes) und keines der beiden Objekte als identitätslos bzw. Stellvertreter für sein Komplexobjekt betrachtet wird (s.u.). Eine „Schneidet“-Relation bezieht sich ausschließlich auf den *Rand* der Objekte (Polygone haben zunächst kein Inneres). In Abb. 5.13(b) soll neue Objekt Gummiband s sowohl Holzstab t als auch u schneiden, da zu beiden ein sichtbarer Schnitt vorliegt. In Abb. 5.13(a) wird zwischen s und t weder eine „Schneidet nicht“- noch eine „Schneidet“-Bedingung etabliert (s.o.).

Enthalten in: Eine „Enthalten in“-Einschränkung wird ausschließlich zu Gebieten etabliert. Sie wird genau dann erzeugt, wenn das Gebiet an allen Stellen des neuen Objektes vollständig sichtbar ist. Ein Gebiet ist an einer Stelle (x, y) genau dann sichtbar, wenn diese Stelle nicht von über diesem Gebiet liegenden anderen undurchsichtigen Gebieten verdeckt wird.

Enthält: Eine „Enthält“-Einschränkung wird ausschließlich von Gebieten etabliert. Eine „Enthält“-Bedingung für ein neues Gebiet wird genau dann erzeugt, wenn das neue Gebiet transparent ist und das enthaltene Objekt an allen Stellen des neuen Gebietes vollständig sichtbar ist.

Zusätzlich gilt:

- Alle Objekte müssen stets innerhalb der rechteckigen Grenzen der tragenden Folie bleiben.
- Es wird niemals eine Einschränkung zwischen einem Objekt und einem seiner (direkten oder indirekten) Komponentenobjekte etabliert (s. auch „Sketch!“ von Bernd Meyer). Einschränkungen zwischen Komponentenobjekten untereinander werden jedoch eingeführt (anhand obiger Regeln) – so z.B. zwischen den Segmenten eines Polygones.
- Zwischen Gebieten werden keinerlei Einschränkungen etabliert.

Wem diese Regeln zu kompliziert sind, der kann entweder ausschließlich transparente Gebiete verwenden (in diesem Fall geht die Abstraktionsmöglichkeit verloren, und es ist wahrscheinlich, daß überspezifizierte Definition entstehen), oder aber undurchsichtige Gebiete stets so verwenden, daß sie keine partiellen (sondern nur totale) Verdeckungen bewirken. Die Diskussion zeigte, daß speziell die partiellen Verdeckungen kritisch sind.

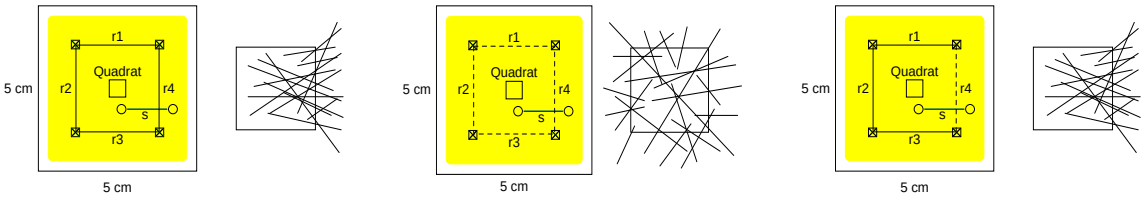


Abbildung 5.14: Relaxierte „Schneidet“- und „Schneidet nicht“-Relationen

Zusätzlich ist es möglich, die „Schneidet“- und „Schneidet nicht“-Relationen zu ignorieren: so sind in Abb. 5.14(a) anhand obiger Regeln u.a. die Einschränkungen „Schneidet(s,Quadrat)“ und „Schneidet(s,r4)“ ablesbar (genaugenommen impliziert die zweite Einschränkung ja die erste – Komponentenobjekte werden jedoch als vollwertige Objekte behandelt, s.o.). Hier stellt sich die Frage, ob man die Komponentenobjekte *wirklich* als Objekte mit eigener Identität betrachten will: würde man von der Identität der Komponentenobjekte abstrahieren, so bliebe nur noch der Einschränkung „Schneidet(s,Quadrat)“ übrig, und eine größere Klasse von Konstellationen wäre beschrieben. Tatsächlich müssten dann aber die ebenfalls anhand von Abb. 5.14(a) ableitbaren „Schneidet nicht“-Bedingungen zu den anderen drei Rechtecksseiten entfernt werden (also „Schneidet nicht(s,r1)“, „Schneidet nicht(s,r2)“, „Schneidet nicht(s,r3)“), s. Abb. 5.14(b).

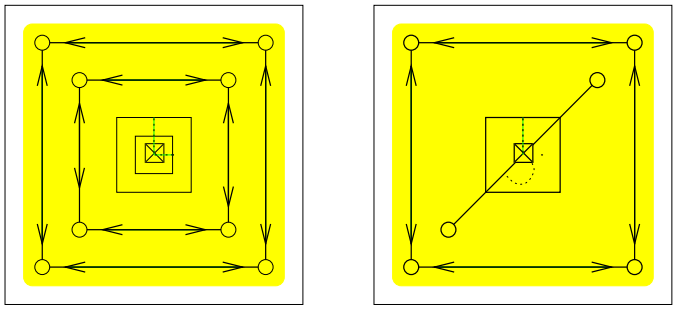
In VISCO ist es möglich, für Komponentenobjekte von evtl. sichtbaren „Schneidet“- oder „Schneidet nicht“-Relationen zu abstrahieren. Die Komponentenobjekte müssen hierzu wieder metagrafisch annotiert werden: man könnte sie z.B. gestrichelt (als eine Art „Geisterobjekt“) darstellen, s. Abb. 5.14(b) (der Prototyp verwendet hingegen ein helles Grau, während Objekte mit Identität stets schwarz dargestellt werden). Diese Relaxierung funktioniert nur, wenn alle Komponentenobjekte eines Komplexobjektes als „identitätslos“ (in obigem Sinne) betrachtet werden: aufgrund der „Schneidet nicht“-Bedingungen zu den drei nicht-gestrichelten Seiten $r1$, $r2$ und $r3$ würde die Relaxierung in Abb. 5.14(c) wirkungslos.

Explizit zu machende Einschränkungen

Während die topologischen Einschränkungen automatisch anhand der Geometrie und der Sichtbarkeitsregeln abgeleitet werden, müssen die folgenden Einschränkungen explizit gemacht werden. „Explizit machen“ bedeutet hier, zusätzliche visuelle Anzeichen (in Form von Metaobjekten) für das Vorhandensein einer Eigenschaft oder Einschränkung zu geben. Teilweise handelt es sich um zusätzliche geometrische Einschränkungen, teilweise um strukturelle oder thematische. Die Einschränkungen werden jeweils durch Metaobjekte visualisiert.

So ist es u.a. möglich

- die Orientierung eines atomaren Segmentes einzuschränken (hierzu dienen Zeiger und Skala, s.o.);
- den Winkel zwischen zwei atomaren Segmenten (die sich kreuzen oder berühren) einzuschränken (hierzu dient der Winkeleinschränker, s.o.);
- die Translations-, Rotations- und/oder Skalierungsmöglichkeit einer Folie einzuschränken;
- die Thematik eines Objektes, welches ein Objekt im räumlichen Datenbestand bzw. der interessierenden Domäne repräsentiert, einzuschränken; und
- die maximale Anzahl der Segmente eine Gummibandes festzulegen.



(a) Mittelpunkt zweier Vierecke

(b) Mittelpunkt eines Viereckes und einer Strecke

Abbildung 5.15: Mittelpunkte

Die räumliche Relation „Mittelpunkt von“ muß vom Benutzer ebenfalls explizit gemacht werden (sie wird nicht anhand der Geometrie abgeleitet): hier bietet sich ein Operator an, der sowohl den Mittelpunkt erzeugt als auch die Einschränkung einträgt. Da es sich um eine *sehr spezielle Relation* handelt, scheint es sinnvoll, sie vom Benutzer explizit machen zu lassen. Bei dem erzeugten Mittelpunkt handelt es sich um ein *abgeleitetes Objekt:* so darf dieser Punkt im Grafikeditor des Prototypen z.B. nicht verschoben werden, da es anderseits zu Inkonsistenzen kommen könnte (die

Selbstkonsistenz des Diagrammes ginge verloren – natürlich darf aber das Argumentobjekt modifiziert werden, wodurch der Mittelpunkt entsp. neuberechnet würde). Der erzeugte Mittelpunkt könnte wieder entsprechend annotiert werden, damit er von einem Punkt, der nur „zufällig“ in der Nähe des geometrischen Mittelpunktes eines Objektes liegt, unterschieden werden kann (im implementierten Prototypen ist dies zugunsten einer übersichtlicheren Darstellung nicht der Fall). Insbesondere ist daran zu denken, daß ein Punkt Mittelpunkt *mehrerer* Objekte werden kann (s. Abb. 5.15).

5.3 Die Sprache VISCO

Nachdem nun die wesentlichsten Entwurfsentscheidungen und hinter VISCO stehenden Ideen erläutert wurden, möchte ich im folgenden die Sprache VISCO vollständig – aber informell – vorstellen.

5.3.1 Das konzeptuelle Datenmodell

Ebenso wie SQL nur für das relationale Datenmodell geeignet ist, ist VISCO nur für folgendes konzeptuelle Datenmodell geeignet. Die Einschränkung auf ein spezielles Datenmodell impliziert, daß nur solche Konstellationen definiert werden können, die innerhalb des Modelles beschrieben werden können. Gleiches gilt z.B. auch für SQL und relationale Datenbanken. Da es sich um ein logisches bzw. konzeptuelles Datenmodell handelt, wird über die konkrete physikalische Implementierung dieser Strukturen nichts ausgesagt.

Prinzipiell wird zwischen zwei Seiten unterschieden: dem Modell für die Repräsentanten (also die VISCO-Sprachelemente) und dem Modell für die repräsentierten Objekte. Für beide Seiten gilt, daß die räumlichen Objekte die Struktur von höchstens vierstufigen DAGs bzgl. der „hat Komponente“-Relation haben. Polygone und Ketten aggregieren eine Reihe von Strecken, Aggregate eine beliebige Anzahl von Objekten, jedoch keine Aggregate. Polygone und Ketten müssen einfach sein – sie dürfen sich nicht selbst überschneiden oder berühren. Objekte können Komponenten einer beliebigen Anzahl von Komplexobjekten sein.

Für die VISCO-Objekte gilt zusätzlich die Einschränkung, daß jedes Objekt (mit Ausnahme der Folien) Komponente *genau einer* Folie sein muß: daher sind die DAGs mindestens zweistufig.

Definition (Direkte Komponenten eines geometrischen Objektes): Die *direkten Komponentenobjekte* einer Strecke sind die Endpunkte, die direkten Komponenten einer Kette oder eines Polygones die Streckensegmente, und die direkten Komponenten eines Aggregates (Folie) die aggregierten Objekte.

Definition (Primäres geometrisches Objekt): Ein *Primärobjekt* ist ein geometrisches Objekt, welches nicht Komponente irgendeines anderen geometrischen Objektes ist. In einer VISCO-Definition sind die einzigen Primärobjekte die verwendeten Folien, in der betrachteten Domäne können aber auch Punkte oder Strecken Primärobjekte sein.

Definition (Universum geometrischer Objekte): Das *Universum* ist die Menge aller wohlgeformter geometrischer Objekte obiger Art. Eine spezielle Teilmenge hiervon ist stets die interessierende Domäne bzw. der räumliche Datenbestand.

Ich möchte betonen, daß es sich bei den verwendeten Objekttypen in gewisser Weise um „generische“ räumliche Konzepte handelt – jedes vektorbasierte GIS benutzt derartige Konzepte. Jedes vektorbasierte GIS muß zumindest Punkte und Strecken explizit machen.

Beim Formulieren von Anfragen mit VISCO an einen räumlichen Datenbestand ergibt sich daß Problem, daß der Benutzer das (externe) Datenmodell kennen muß (dies gilt übrigens auch für SQL). Eine stark unterstützende Oberfläche könnte hier umfangreiche Hilfestellung bieten: so könnte z.B. anhand thematischer Aspekte vom System gleich der richtige Objekttyp ausgewählt

werden. So weiß der naive Benutzer in der Regel nicht, ob er einen Fluß als Kette oder als Polygon repräsentieren soll. Die richtige Wahl könnte jedoch anhand des „Datenwörterbuches“ des GIS (also anhand der GIS-Metadaten) ermittelt werden (s. auch Kap. 6). Zudem könnte der Compiler entsprechende Transformationen vornehmen. Kritisch ist auch, daß hier die Kenntnis der „hat Komponente“-Relation bzw. der partonomischen Struktur der Objekte vom Benutzer verlangt wird. Auch die Unterscheidung von impliziten und expliziten Objekten (sind z.B. Wiesen im Datenbestand explizit als Polygone vorhanden oder lediglich die Begrenzungslinien?) ist kritisch. Auch hier kann die Oberfläche Unterstützung bieten.

5.3.2 Allgemeines und Definitionen

Definition (VISCO-Definition): Eine VISCO-*Definition* besteht aus einer Sequenz von Diagrammen, die die Konstruktionsgeschichte darstellen. In jedem Konstruktionsschritt wird genau ein neues *Objekt* hinzugefügt – das Konstruieren selbst geschieht am besten mit einer stark unterstützenden Oberfläche. Die Bedeutung der Sequenz kann *prinzipiell* auch ohne die Oberfläche rekonstruiert werden, wenn alle semantikrelevanten Aspekte visualisiert werden. *Die Konstrukion einer Sequenz ohne erstellendes Werkzeug ist jedoch nicht möglich.* Prinzipiell gibt es zwei mögliche Konstruktionsschritte:

- Objekterzeugung und
- Metaobjekterzeugung (grafische Annotation).

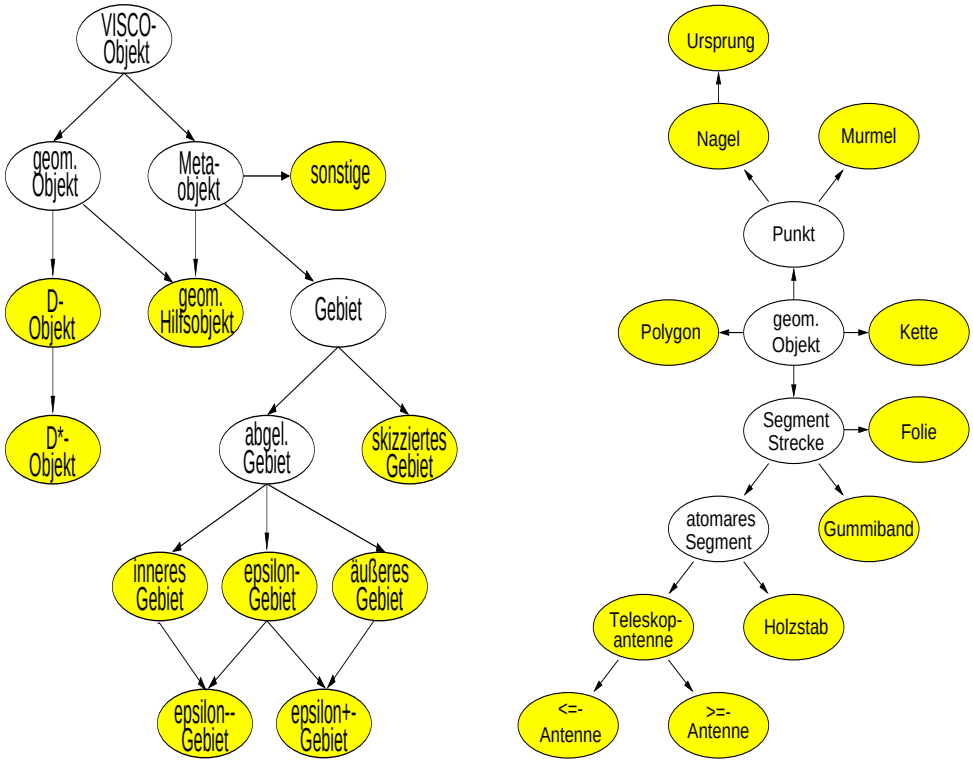


Abbildung 5.16: Terminologie

Definition (VISCO-Objekt): Prinzipiell gibt es zwei VISCO-Objektarten: *geometrische Objekte* und *Metaobjekte*. Geometrische Objekte kommen sowohl als D- als auch als spezielle Metaobjekte,

sog. geometrische Hilfsobjekte vor. Andere Metaobjekte sind die Gebiete und sonstige Metaobjekte (s. Abb. 5.16(a), konkrete Klassen sind grau hinterlegt).

Geometrische Objekte sind *Punkte (Murmeln, Nägel, Ursprünge), Strecken bzw. Segmente (Gummibänder, Holzstäbe, Teleskopantennen), Polygone, Ketten und Aggregate (Overheadprojektorfolien).* Holzstäbe und Teleskopantennen werden auch als *atomare Segmente* bezeichnet (s. Abb. 5.16(b)).

Die Klasse der geometrischen Objekte wird weiter aufgeteilt:

D-Objekte repräsentieren die geometrischen Objekte der interessierenden **D**omäne, z.B. die aufzufindenden Geo-Objekte in einem GIS. D-Objekte haben in der Regel (aber nicht notwendigerweise) eine Thematik (z.B. „See"). **Unter einem D-Objekt wird also im folgenden ein geometrisches Objekt verstanden, das ein explizites geometrisches Objekt des räumlichen Datenbestandes bzw. der interessierenden Domäne repräsentiert bzw. gegen ein solches gebunden werden soll.** D-Objekte müssen stets explizit im Datenbestand vorhanden sein (Ausnahme: D-Gummibänder). Da D-Objekte Objekte im Datenbestand repräsentieren, müssen alle (evtl. vorhandenen) Komponentenobjekte notwendigerweise ebenfalls D-Objekte sein (Top-Down). Eine *spezielle Teilmenge der D-Objekte* sind die

D*-Objekte: sie repräsentieren ebenfalls geometrische Objekte der Domäne, jedoch wird verlangt, daß die repräsentierten Domänenobjekte *primäre* Objekte sind. Somit repräsentieren sie eine spezielle Teilmenge der D-Objekte. **D*-Objekte werden mit einem „$\star$" annotiert.** Da D*-Objekte primäre Objekte im Datenbestand repräsentieren, müssen alle (evtl. vorhandenen) Komponentenobjekte notwendigerweise D-Objekte sein (Top-Down). Natürlich darf ein D-Objekt (und somit auch ein D*-Objekt) keine D*-Komponentenobjekte haben. Alle *Elternkomplexobjekte* müssen zudem notwendigerweise *geometrische Hilfsobjekte* sein.

Wird von D-Objekten geredet, so sind damit auch D*-Objekte gemeint.

Geometrische Hilfsobjekte repräsentieren Objekte im gesamten *Universum geometrischer Objekte* (Def. s.o.) – sie umfassen somit auch die Klasse der D-Objekte (und daher auch die Klasse der D*-Objekte). Ein Hilfsobjekt ist somit ein *Metaobjekt.* **Hilfsobjekte werden mit einem „U" (für Universum) annotiert.** Hilfsobjekte dienen dazu, zusätzliche Einschränkungen für die D-Objekte definieren bzw. konstruieren zu können. Alle Elternkomplexobjekte eines Hilfsobjektes müssen ebenfalls Hilfsobjekte sein (Bottom-Up) – für die Komponentenobjekte gelten hingegen keine diesbezüglichen Einschränkungen.

Teilweise ist es sinnvoll, *geometrische Objekte zu berechnen,* wie z.B. den Mittelpunkt eines Objektes. In diesem Fall spreche ich auch von einem *abgeleiteten Objekt.* Zum Beispiel muß die räumliche Relation „Mittelpunkt von" explizit gemacht werden.

Die Komponentenobjekte eines Objektes können bzgl. der topologischen Relationen „Schneidet" und „Schneidet nicht" als nicht vorhanden betrachtet werden: dies gilt entweder für alle oder keines der Komponenten eines Komplexobjektes.

Gebiete sind spezielle Metaobjekte und repräsentieren Gebiete im Universum geometrischer Objekte. Ein Gebiet ist dabei ein Objekt, welches andere Objekte *enthält.* Gebiete können entweder vom Benutzer skizziert oder anhand anderer geometrischer Objekte berechnet werden (in diesem Fall spreche ich auch von einem abgeleiteten Gebiet).

Andere Metaobjekte repräsentieren keine Objekte, sondern visualisieren (bzw. repräsentieren) spezielle Einschränkungen.

Alle diese Objektarten müssen visuell unterschieden werden können. Jedes dieser Objekte wird auf *genau eine Folie* konstruiert (mit Ausnahme der Folien selbst). Alle Komponentenobjekte eines komplexen Objektes befinden sich stets auf der selben Folie wie das Komplexobjekt selbst – lediglich für den Ursprung einer Folie gibt es eine Ausnahmeregelung (s.u.).

Während bisher definiert wurde, welche VISCO-Objekte wofür stehen, muß nun festgelegt werden, welches die relevanten Eigenschaften und Relationen zwischen ihnen sind. Die relevanten Eigenschaften und Relationen der Repräsentanten werden als notwendige zu erfüllende Beschränkungen für bzw. zwischen den durch sie repräsentierten Objekten interpretiert.

Definition (Beschränkungen [Constraints]): Prinzipiell gibt es drei Arten von Beschränkungen: *topologische, metrische / geometrische* und *strukturelle Einschränkungen.* Dabei wird zwischen implizit und explizit repräsentierten Einschränkungen bzw. Aspekten unterschieden. Für die expliziten Einschränkungen müssen zusätzliche visuelle Anzeichen (in Form von Metaobjekten) angegeben werden, was für die impliziten Einschränkungen nicht der Fall ist.

Topologische Einschränkungen: Implizit anhand der Geometrie repräsentiert werden die Relationen „Enthalten in“, „Enthält“, „Schneidet“ und „Schneidet nicht“. Diese Beschränkungen werden anhand der Geometrie abgeleitet, also explizit gemacht.

Metrische / geometrische Einschränkungen: Implizit repräsentiert werden:

- Die Position eines Nagels,
- die Länge eines Holzstabes,
- die maximale Länge einer $\leq$-Teleskopantenne und
- die minimale Länge einer $\geq$-Teleskopantenne.

Alle geometrischen Attribute beziehen sich ausschließlich auf das lokale kartesische Koordinatensystem der tragenden Folie. *Explizit* gemacht werden müssen die metrischen bzw. geometrischen Einschränkungen

- „Mittelpunkt von“, die
- Orientierungseinschränkungen von Folien und atomaren Segmenten, die
- Winkeleinschränkungen zwischen atomaren Segmenten und schließlich die
- Skalierungseinschränkungen von Folien.

Strukturelle Einschränkungen: Implizit repräsentiert wird die

- „hat Komponente“-Relation – sie erzeugt eine „hat Komponente“-Einschränkung, bestimmt also also die Struktur des DAG.

Explizit gemacht werden müssen die

- Thematik von D-Objekten, sowie die
- maximale Anzahl erlaubter Komponentenobjekte von Aggregaten, Polygonen, Ketten und Gummibändern, sowie die maximale Anzahl erlaubter enthaltener Objekte eines Gebietes.

Folgende Regeln bestimmen, unter welchen Umständen eine topologische Relation als entsp. topologische Beschränkung für ein neu eingeführtes Objekt etabliert wird:

Schneidet nicht: aus einer sichtbaren „Schneidet nicht“-Relation wird genau dann eine „Schneidet nicht“-Einschränkung abgeleitet bzw. expliziert, wenn das andere Objekt vollständig sichtbar ist und keines der beiden Objekte als identitätslos bzw. Stellvertreter für sein Komplexobjekt betrachtet wird (s.u.). Eine „Schneidet nicht“-Einschränkung / Relation bezieht sich ausschließlich auf den Rand der Objekte.

Schneidet: eine „Schneidet“-Einschränkung wird genau dann abgeleitet, wenn es mindestens einen sichtbaren Schnittpunkt mit dem anderen Objekt gibt (in diesem Fall gibt es ein sichtbares Anzeichen für das Vorliegen des Schnittes) und keines der beiden Objekte als identitätslos bzw. Stellvertreter für sein Komplexobjekt betrachtet wird (s.u.). Eine „Schneidet“-Relation bezieht sich ausschließlich auf den Rand der Objekte (der Rand eines Punktes ist der Punkt selbst; analoges gilt für Strecken – ein Punkt „Schneidet“, wenn er auf dem anderen Objekt liegt).

Enthalten in: Ein „Enthalten in“-Einschränkung wird ausschließlich zu Gebieten etabliert. Sie wird genau dann erzeugt, wenn das Gebiet an allen Stellen des neuen Objektes vollständig sichtbar ist. Ein Gebiet ist an einer Stelle (x, y) genau dann sichtbar, wenn diese Stelle nicht durch über diesem Gebiet liegende andere undurchsichtige Gebiete verdeckt wird.

Enthält: Eine „Enthält“-Einschränkung wird ausschließlich von Gebieten etabliert. Eine „Enthält“-Bedingung für ein neues Gebiet wird genau dann erzeugt, wenn das neue Gebiet transparent ist und das enthaltene Objekt an allen Stellen des neuen Gebietes vollständig sichtbar ist.

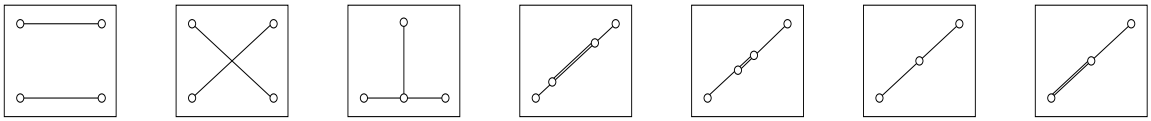


Abbildung 5.17: Nichtäquivalente Strecke-Strecke-Konstellationen

Da die Komponentenobjekte als vollwertige Objekte betrachtet werden, lassen sich alle in Abb. 5.17 angegebenen Strecke-Strecke-Konstellationen voneinander unterscheiden (sie haben unterschiedliche Beschreibungen). Die Konstellationen (f) und (g) unterscheiden sich von den anderen Konstellationen, da hier ein gemeinsamer Komponentenpunkt vorliegt. In (e) überlappen sich die Strecken, während in Konstellation (d) die eine Strecke innerhalb der anderen verläuft.

Ließe man kongruente Punkte und Strecken zu, so könnte man diese Relationen z.B. anhand einer Matrix charakterisieren, in der 1 für „Schneidet“ und und 0 für „Schneidet nicht“ steht (s. auch die 9er-Schnittmatrix von Egenhofer). Die Matrix listet dabei die Relationen zwischen den einzelnen Bestandteilen auf ($p1(s)$ und $p2(s)$ bezeichnet die Endpunkte von s):

	s	p1(s)	p2(s)
t	1	0	0
p1(t)	0	0	0
p2(t)	0	0	0

Diese Matrix beschreibt die Konstellation (b).

Die Matrizen

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

beschreiben die Konstellation (c) – es ergeben sich hier vier Matrizen aufgrund unterschiedlicher Benennungen der Objekte. Zur Charakerisierung der Konstellation (a) hingegen wird nur eine Matrix (nämlich die Null-Matrix) benötigt.

Die Matrizen spielen im folgenden keine Rolle mehr, sondern sollten lediglich die *Differenzierungsmächtigkeit der verwendeten Relationen unter der Betrachtung von Komponentenobjekten als eigenständige Objekte verdeutlichen.* Hier ist zu beobachten, daß teilweise *zu stark differenziert* wird: hier kann jedoch die bereits erwähnte Relaxierungsmöglichkeit der „Schneidet“ und „Schneidet nicht“-Relationen helfen.

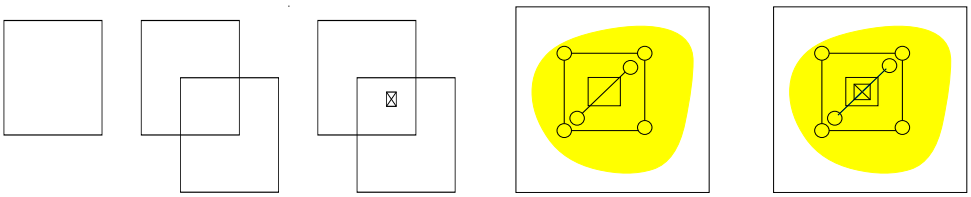
Definition (Freiheitsgrade geometrischer Objekte): Unter den *Freiheitsgraden eines geometrischen Objektes* werden alle erlaubten Abweichungen der visualisierten bzw. dargestellten Geometrie dieses Objektes verstanden. Hierzu gehören Vagheiten

- der Form,
- der Lage,
- der Orientierung (bei fester Form),
- der Größe bzw. Skalierung (bei fester Form), etc.

So ist der Freiheitsgrad einer Murmel z.B. die Position, der Freiheitgrad einer Teleskopantenne die Länge, der Freiheitsgrad eines Gummibandes die Form, etc.

Probleme mit Mehrdeutigkeiten

Oben wurde ausgesagt, daß die Semantik einer VISCO-Definition *prinzipiell* durch Interpretation der Diagrammsequenz auch ohne eine stark unterstützende Oberfläche ermittelt werden kann. Prinzipiell sind zur Auflösung von Mehrdeutigkeiten dann jedoch einige zusätzliche Konventionen oder metagrafische Annotationen notwendig – in Kap. 2 wurden einige potentielle Mehrdeutigkeiten ausführlicher diskutiert. Im implementierten Prototypen wurden einige Mehrdeutigkeiten in den visuellen Repräsentationen zugunsten einer übersichtlicheren Darstellung zugelassen. Diese Mehrdeutigkeiten können jedoch durch die dynamischen Inspektionsmöglichkeiten und den integrierten Stöberer stets aufgelöst werden. Betrachtet man jedoch ausschließlich die statische Sequenz der Diagramme des Prototypen (z.B. einen Papierausdruck), so ist VISCO hier weder vollständig noch wahrheitsgetreu (s. Kap. 2).



(a) (b)

Abbildung 5.18: Mehrdeutigkeiten

So ist beim Betrachten der Sequenz aus Abb. 5.18(a) nicht klar ist, ob der Nagel nun auf der ersten oder zweiten Folie konstruiert wurde. Führt man die Konvention ein, daß neue Objekte stets nur auf der zuletzt erzeugten Folie konstruiert werden dürfen oder annotiert man die gemeinte Folie z.B. mit einem Pfeil, so ist dies wieder eindeutig entscheidbar. Ein andere Möglichkeit wäre, jeder Folie eine Farbe zu geben und zu fordern, daß alle Komponenten die gleiche Farbe haben müssen (s. auch Kap. 2).

Eine vergleichbare Problematik ergibt sich für abgeleitete Objekte: so ist z.B. in der (Teil)Sequenz 5.18(b) nicht klar, ob der Mittelpunkt nun Mittelpunkt des Rechteckes oder der Strecke (oder gar beider) sein soll. Auch hier würde wieder eine Konvention helfen (z.B., daß ein Operator nur auf das zuletzt konstruierte Objekt angewendet werden darf). Ebenso könnte man aber eine metagrafische Annotation vorsehen wie in Abb. 5.15.

Noch komplizierter wird die Situation, wenn man kongruente Objekte zuläßt. Im Prototypen werden kongruente Punkte und Strecken ausgeschlossen (selbiges gilt auch für den Datenbestand). Zwei Komplexobjekte mit gemeinsamen Komponenten haben zumindest eine nicht-gemeinsame Komponente.

5.3.3 Ausführliche Vorstellung der Sprachelemente

Es folgt nun eine ausführliche Vorstellung der einzelnen Primitive. Zunächst werden die Overheadprojektorfolien und Gebiete vorgestellt, dann die restlichen geometrischen Objekte und weitere Metaobjekte.

Prinzipiell gilt, daß jedes geometrische Objekt genau ein geometrisches Objekt im Universum geometrischer Objekte (bzw. der interessierenden Domäne im Falle eines D-Objektes) repräsentiert bzw. gegen ein solches gebunden werden soll. Keine zwei VISCO-Objekte repräsentieren das gleiche Objekt.

Overheadprojektorfolien

Die *Overheadprojektorfolie (kurz Folie)* ist der Träger für alle weiteren VISCO-Objekte – diese werden auf Folien konstruiert. Folien sind durchsichtig und rechteckig. Auf eine Folie läßt sich eine beliebige Anzahl von Objekten konstruieren – alle Objekte (und auch die abzugleichenden Objekte) müssen stets innerhalb der rechteckigen Grenzen der Folie bleiben. Eine Folie definiert somit ein Aggregat. Jedes Objekt ist ausschließlich auf einer Folie definiert (mit Ausnahme des Ursprungs, s.u.). Alle Komponenten eines Objektes sind auf der gleichen Folie definiert (so z.B. die beiden Endpunkte einer Strecke).

Folien können *Hilfsobjekte* oder D*-Objekte sein. Folien sind stets Primärobjekte. Folien repräsentieren Aggregate: eine D*-Folie repräsentiert ein Aggregat der Domäne bzw. des Datenbestandes, während eine U-Folie ein Aggregat im Universum geometrischer Objekte repräsentiert. Normalerweise wird man U-Folien verwenden.

Folien haben bestimmte *Freiheitsgrade,* denn sie lassen sich

- Translatieren,
- Skalieren (evtl. nur proportional) und
- Rotieren.

Ohne weitere Einschränkungen in Form zusätzlicher Metaobjekte läßt sich jede Folie beliebig translatieren und rotieren.

Ein Koordinatensystem erfordert einen *Ursprung.* Der Ursprung einer Folie ist bzgl. der Folie selbst stets ein *spezieller Nagel,* denn er hat bzgl. der Folie die feste Position $(0, 0)$.

Zeiger und Skala (s.u.) können am Ursprung einer Folie befestigt werden: so wird die Rotierbarkeit der Folie eingeschränkt. Die Skalierbarkeit wird durch Metaobjekte in Form *zusätzlicher Pfeile,* und Proportionalität durch *Führungslinien* visualisiert. Die erlaubten Skalierungsbereiche werden textuell durch *Intervalle an den entsp. Seiten* visualisiert. Es sollte klar sein, daß diese Elemente weitgehend orthogonal kombiniert werden können – einige Beispiele sind in Abb. 5.19 zu sehen.

Folien können übereinandergestapelt werden – hier zählen dann lediglich die topologischen Relationen (s. Sichtbarkeitsregeln): so soll z.B. in Definition 5.20 das kreisartige Gummiband-Polygon auf der rechten Folie die linke Rechtecksseite auf der linken Folie berühren. Beide Folien lassen sich beliebig skalieren (aber nicht rotieren) – lediglich die eingezeichneten topogischen Bedingungen müssen dabei erhalten bleiben.

Man kann jedoch auch die *Geometrie bzw. Metrik von Folien aufeinander beziehen:* Folien können miteinander *verknüpft* werden, denn *der Ursprung einer Folie kann eine Murmel oder ein Nagel einer weiter unterliegenden Folie sein.* In Abb. 5.21(b) werden die beiden Folien aus Abb. 5.21(a) über einen *gemeinsamen Ursprung* verknüpft. Dies muß vom Benutzer explizit gemacht werden (z.B. per Operatoranwendung) – der Ursprung ist somit das einzige Objekt, welches Komponente mehrerer Folien (Aggregate) sein darf. Die beiden auf diese Art verknüpften Folien werden dann auch Eltern- und Kindfolie genannt. Der Vorgang kann beliebig oft wiederholt werden, so

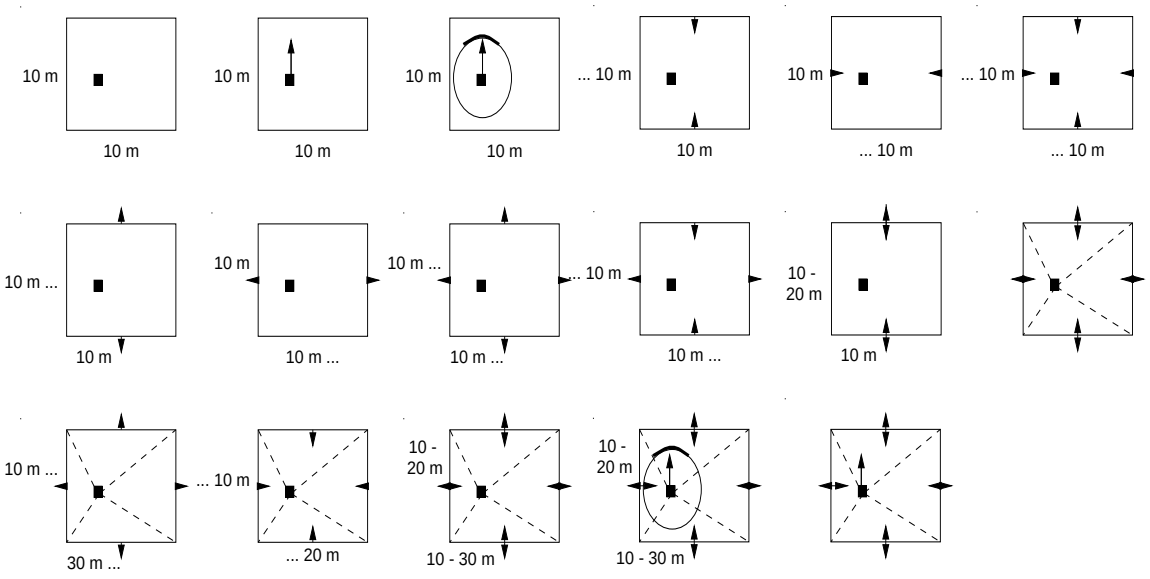


Abbildung 5.19: Einige Beispiele für Overheadprojektorfolien

daß eine strikte Hierarchie von Folienverknüpfungen entsteht. Die verwendeten Visualisierungen für Ursprünge sind in Abb. 5.21(c) angegeben, v.l.n.r und o.n.u.: normaler Ursprung (kein Verknüpfung), Ursprung der Kindfolie ist Murmel der Elternfolie, Ursprung der Kindfolie ist Nagel der Elternfolie, Ursprung der Kindfolie ist Ursprung der Elternfolie. Für jede weitere Verknüpfung bzw. Kindfolie wird um den so geteilten Punkt ein weiteres Quadrat gemalt (s. zweite Reihe in Abb. 5.21(c)). Klar ist, daß bzgl. der Kindfolie der Ursprung immer die Rolle eines Nagels spielt, da er die feste Position $(0, 0)$ hat.

Ist nun der Ursprung der Kindfolie eine Murmel der Elternfolie, so ist durch diese Verknüpfung ein möglicher Translationsbereich der Kindfolie relativ zur Elternfolie definiert. Ist der Ursprung hingegen ein Nagel der Elternfolie, so besteht keine Translationsmöglichkeit relativ zu Elternfolie. Murmeln bzw. Nägel, die in dieser Form als Ursprünge benutzt werden, werden gesondert visualisiert (s. Abb. 5.21(c)). Aber nicht nur der erlaubte Translationsbereich der Kindfolie ist dann *relativ zur Elternfolie* definiert (hierfür würde ja bereits eine „Enthalten in“-Bedingung zwischen dem Ursprung der Folie und einem Gebiet einer anderen Folie ausreichen, s. Abb. 5.20): **auch die Rotations- und Skalierungsparameter der Kindfolie werden dann auf die Elternfolie bezogen und als *relativ* betrachtet.** Formal handelt es sich hier um eine Verknüpfung bzw. Komposition der Transformationsmatrizen der beiden Folien. Während für eine Folie *ohne Elternfolie* die erlaubten Skalierungsbereiche in Metern in der Welt angeben werden (wie 100 m., 0 – 100 m., 100 – ∞ m., 100 – 200 m.), wird die Geometrie einer Kindfolie bzgl. ihrer Elternfolie ernst genommen – entsp. Skalierungsbereiche werden dann *prozentual zur Geometrie der Elternfolie* angegeben (z.B. 0.8 – 1.2).

Relative Orientierungseinschränkungen von Folien werden in Abb. 5.22 verdeutlicht: während in Abb. 5.22(a) die Orientierung der Elternfolie (bzgl. des Weltkoordinatensystemes) festgelegt wird und die Orientierung der Kindfolie beliebig ist, ist in 5.22(b) die Orientierung der Elternfolie beliebig und die Orientierung der Kindfolie relativ zur Elternfolie fixiert – somit hat die Kindfolie immer die gleiche Ausrichtung wie die Elternfolie. Schließlich haben beide Folien in Abb. 5.22(c) keine Orientierungsfreiheit mehr.

Eine Folie ohne Elternfolie bezieht sich implizit auf das Weltkoordinatensystem: ausschließlich für eine solche Folie können

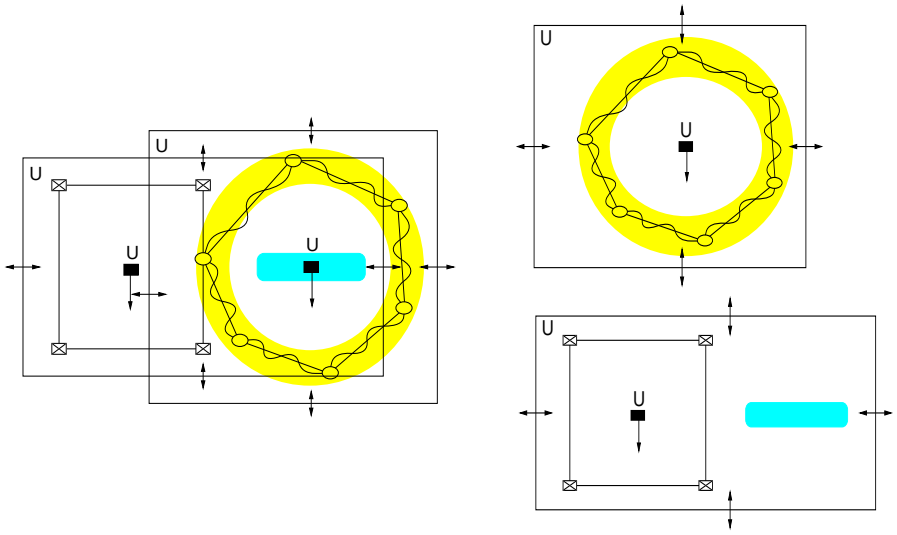
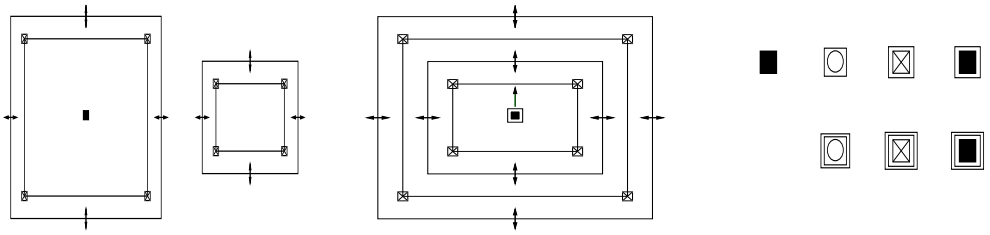


Abbildung 5.20: Topologische Relationen zwischen Objekten auf verschiedenen Folien



(a) Zwei Folien ... (b) ... werden verknüpft (c) Ursprünge

Abbildung 5.21: Verknüpfte Folien

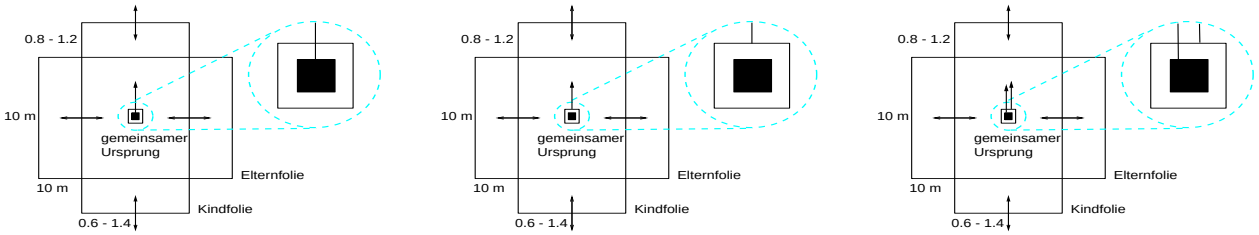
- Breite und
- Höhe (z.B. in Metern)

in der Natur bzw. der Welt angegeben werden (s.o.). Der Translationsbereich für den Ursprung einer solchen Folie ist das gesamte Weltkoordinatensystem (es sei denn, eine „Enthält"-Bedingung zu einem Gebiet einer anderen Folie existiert). Für Folien mit Elternfolie hingegen werden die Skalierungsbereiche relativ auf die Geometrie bzw. die Abmessungen der Elternfolie bezogen und daher prozentual angegeben.

Für Folien läßt sich die *maximale Anzahl von Komponentenobjekten* durch eine *Maximumseinschränkung* festlegen (dies ist in erster Linie für D-Objekte sinnvoll). Eine Festlegung von n bedeutet, daß die durch diese Folie repräsentierten Aggregate nicht mehr als n Komponentenobjekte haben dürfen. Die Maximumseinschränkung wird einfach durch die entsp. Zahl visualisiert.

Gebiete

Gebiete sind Metaobjekte – sie repräsentieren Gebiete im Universum geometrischer Objekte. Gebiete *enthalten* geometrische Objekte. Es gibt zwei Arten von Gebieten:



(a) Rotierbare Kindfolie (b) Rotierbare Elternfolie (c) Beide nicht rotierbar

Abbildung 5.22: Relative Orientierungseinschränkungen

Skizzierte oder konstante Gebiete werden vom Benutzter skizziert (es handelt sich um Polygone mit optionalen Löchern, s. Abb. 5.23(a)).

Berechnete oder abgeleitete Gebiete entstehen durch Operatoranwendung auf Objekten (den *Argumenten* der Gebiete). Form und Position sind somit nicht notwendigerweise fest, sondern können variieren (falls die Argumente irgendwelche Freiheitsgrade haben). Die Klasse der berechneten Gebiete wird weiter unterteilt:

Innere Gebiete denotieren das Innere von Polygonen (s. Abb. 5.23(b)).

Äußere Gebiete denotieren das Äußere von Polygonen bzgl. der tragenden Folie (s. Abb. 5.23(c)).

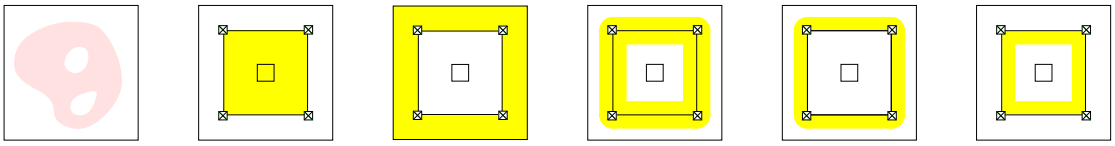
$\epsilon(r)$**-Gebiete** denotieren alle Punkte des lokalen Koordinatensystemes, die nicht weiter als r vom Argument (bzgl. der lokalen Metrik) entfernt, aber natürlich innerhalb der Grenzen der tragenden Folie sind (s. Abb. 5.23(d)). In der GIS-Literatur wird auch von *Pufferzonen* gesprochen.

$\epsilon^{\oplus}(r)$**-Gebiete** sind nur für Polygon-Argumente definiert: sie sind sowohl $\epsilon(r)$-Gebiete als auch innere Gebiete (s. Abb. 5.23(e)).

$\epsilon^{\ominus}(r)$**-Gebiete** sind ebenfalls ausschließlich für Polygon-Argumente definiert: sie sind sowohl $\epsilon(r)$-Gebiete als auch äußere Gebiete (s. Abb. 5.23(f)).

Die Argumente müssen stets auf der selben Folie wie das Gebiet selbst definiert sein.

Gebiete werden auf Folien konstruiert und färben den von ihnen denotierten Bereich farblich ein. *Berechnete Gebiete* werden stets so dargestellt, daß sie von den *konstanten Gebieten* visuell unterschieden werden können. Gebiete dienen zur Visualisierung von „Enthalten in"- bzw. „Enthält"-Bedingungen – diese Bedingungen werden ausschließlich zu bzw. von Gebieten etabliert (s. obige Regeln). Diese Relationen gelten nicht nur für Murmeln, sondern für alle geometrischen Objekte (also auch Strecken, ganze Ketten und Polygone und sogar ganze Folien). *Opake (undurchsichtige) Gebiete* können bereits vorhandene andere Objekte verdecken, was für *transparente Gebiete* nicht der Fall ist.



(a) (b) (c) (d) (e) (f)

Abbildung 5.23: Gebiete

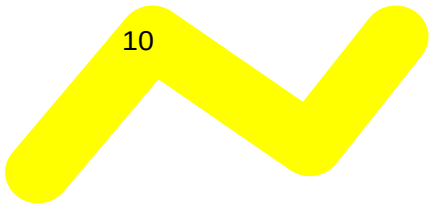
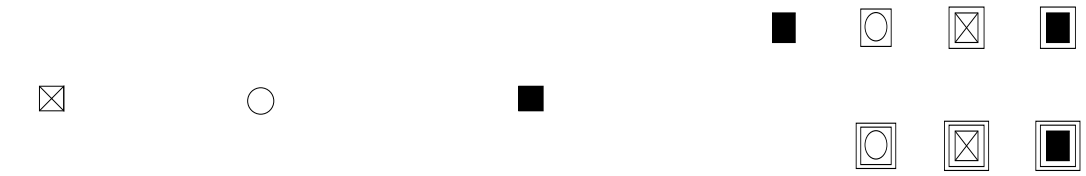


Abbildung 5.24: Ein opakes ϵ-Gebiet mit $n = 10$

Für Gebiete läßt sich eine *Maximumseinschränkung* angegeben: sie beschränkt die Anzahl der in diesem Gebiet liegenden primären geometrischen D-Objekte auf ein bestimmtes Maximum. Eine *Maximumseinschränkung* von n bedeutet, daß die durch dieses Gebiet repräsentierte Fläche nicht mehr als n primäre geometrische Domänenobjekte enthalten darf. Abb. 5.24 zeigt ein opakes ϵ-Gebiet für eine Kette, indem sich nicht mehr als 10 Objekte des Datenbestandes aufhalten sollen. Manchmal wird man $n = 0$ setzen wollen – man beachte, daß dies z.B. für ϵ-Gebiete von D-Objekten nicht möglich ist, da das Argumentobjekt selbst notwendigerweise immer im Gebiet selbst enthalten ist.

Punkte



(a) Nagel (b) Murmel (c) normaler Ursprung (d) Ursprünge

Abbildung 5.25: Punkte

Es gibt zwei Arten von Punkten: Murmeln und Nägel. Der Ursprung einer Folie ist ein spezieller Nagel – für jede Folie wird stets ein Ursprung benötigt. Ein Punkt kann direkte Komponente von Strecken oder Folien sein. Ein Punkt kann entweder ein *D-Objekt (bzw. D*-Objekt)* oder ein *Hilfsobjekt* sein. Ein D-Punkt repräsentiert Punkte der Domäne, während ein Hilfspunkt beliebige Punkte (aus der Menge aller Punkte) repräsentiert. Ein D*-Punkt repräsentiert primäre Punkte der Domäne (im Sinne von primären Objekten).

Ein Punkt kann Mittelpunkt eines Objektes sein: in diesem Fall wird die spezielle Einschränkung „Mittelpunkt von“ erzwungen. Ein Punkt kann Schnittpunkt zweier Segmente sein: zwischen diesen darf dann kein eindimensionaler Schnitt vorliegen. Dies ist nur für nicht topologisch strukturierte Vektordaten sinnvoll (s. Kap. 3), da anderseits die beiden Kanten aufgetrennt und der „Schnittpunkt“ somit in der „Komponente von“-Relation zu den beiden Kanten stehen würde.

Nagel: Ein Nagel (s. Abb. 5.25(a)) repräsentiert einen Punkt mit (bezogen auf das tragende Folienkoordinatensystem) exakt bekannter Position („Position ist (x, y)“-Einschränkung). *Hilfsnägel* sind gut zum Konstruieren weiterer Einschränkungen (bzw. geometrischer Hilfsobjekte) zu gebrauchen (s. folgende Beispiele).

Murmel: Eine Murmel (s. Abb. 5.25(b)) repräsentiert einen Punkt mit partiell unbekannter bzw. vager Position; *eine Murmel muß mindestens in einem Gebiet ersichtlich enthalten sein.* Dieses Gebiet muß nicht notwendigerweise auf der selben Folie wie die Murmel sein – lediglich auf die Sichtbarkeit an dieser Stelle kommt es an. Die Murmel kann innerhalb des denotierten Gebietes jede Position einnehmen, sie darf es jedoch nicht verlassen (dies gilt auch entsp. für alle anderen Objekte bzgl. der „Enthält“ / „Enthalten in“-Bedingung).

Für jede sichtbare „Enthalten in“- bzw. „Enthält“-Relation wird eine entsp. Einschränkung erzeugt.

Eine *Hilfsmurmel* ist in der Regel unbrauchbar, es sei denn, sie steht in einer sehr speziellen Relation (wie „Mittelpunkt von“ oder ist z.B. Schnittpunkt zweier Strecken). Der Prototyp verlangt an dieser Stelle weitere Einschränkungen, s. Kap. 6 und 7 sowie Anhang.

Ursprung: Der Ursprung definiert den Ursprung des lokalen Folienkoordinatensystemes. Der Ursprung ist entweder (s. 5.25(d))

1. auf der Folie selbst definiert,
2. ein Nagel einer unterliegenden Folie (evtl. auch ein Ursprung), oder
3. eine Murmel einer unterliegenden Folie.

Der Ursprung ist das einzige Objekt, welches Komponente mehrerer Folien sein darf (in der „Komponente von“-Beziehung).

Im 2. und im 3. Fall wird die bereits besprochene Kopplung der beiden so verknüpften Folien vorgenommen.

Am Ursprung kann bei Bedarf *pro Folie* (höchstens) ein Zeiger angebracht werden, und an einem Zeiger (höchstens) eine Skala: so wird die Rotierbarkeit der Folie (relativ zur Elternfolie oder dem Weltkoordinatensystem) eingeschränkt.

Atomare Strecken bzw. Segmente

(a) Holzstab (b) Teleskopantenne (c) $\leq$-Antenne (d) $\geq$-Antenne

Abbildung 5.26: Atomare Strecken bzw. Segmente

Es gibt vier Arten *atomarer Strecken: Holzstäbe, Teleskopantennen* sowie $\leq$- und $\geq$-*Teleskopantennen* (s. Abb. 5.26).

Eine *atomare Strecke* kann direkte Komponente beliebiger Polygone oder Ketten oder einer Folie sein. Eine atomare Strecke kann entweder ein *D-Objekt* oder ein *Hilfsobjekt* sein. Eine atomare D-Strecke repräsentiert Strecken der Domäne, während eine Hilfsstrecke beliebige Strecken (aus der Menge aller Strecken) repräsentiert. Ein D*-(atomare) Strecke repräsentiert primäre Strecken der Domäne (im Sinne von primären Objekten). Strecken sind geradlinige Verbindungen zwischen den beiden Endpunkten. Die beiden Endpunkte einer Strecke liegen auf der gleichen Folie wie die Strecke selbst. Zwei Strecken dürfen sich schneiden bzw. kreuzen, ohne daß die Notwendigkeit besteht, die Strecken unter Verwendung eines Schnittpunkt-Knotens aufzuspalten. Der Schnittpunkt zweier Strecken kann explizit gemacht werden. Eindimensionale Schnitte sind auch erlaubt – dann kann jedoch kein Schnittpunkt erzeugt werden.

An einer atomaren Strecke kann bei Bedarf (höchstens) ein *Zeiger* angebracht werden, und am Zeiger eine *Skala* (s. Beschreibung dort). Zeiger und Skala schränken die möglichen Orientierungen der atomaren Strecke bzgl. des lokalen Folienkoordinatensystemes ein.

Zwischen zwei in der Relation „Schneidet“ stehenden atomaren Strecken auf der gleichen Folie kann bei Bedarf (höchstens) ein *Winkeleinschränker* konstruiert werden (s. dort). So können die erlaubten Winkel zwischen beiden eingeschränkt werden.

Holzstab: Holzstäbe repräsentieren Strecken mit einer (bezogen auf das lokale Folienkoordinatensystem bzw. dessen euklidische Metrik) festen Länge (s. Abb. 5.26(a)).

Teleskopantenne: Teleskopantennen repräsentieren Strecken mit einer undefinierten Länge (s. Abb. 5.26(b)).

Schrumpfende Teleskopantenne ($\leq$-Teleskopantenne): Schrumpfende Teleskopantennen repräsentieren Strecken mit einer (bezogen auf das lokale Folienkoordinatensystem) maximalen Länge (s. Abb. 5.26(c)).

Wachsende Teleskopantenne ($\geq$-Teleskopantenne): Wachsende Teleskopantennen repräsentieren Strecken mit einer (bezogen auf das lokale Folienkoordinatensystem) minimalen Länge (s. Abb. 5.26(d)).

Gummiband

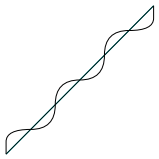


Abbildung 5.27: Gummiband

Ein *Gummiband* kann direkte Komponente beliebiger Polygone, Ketten oder einer Folie sein. Ein Gummiband repräsentiert *Verbindungen* oder *Wege in Form von Ketten zwischen den durch seine Endpunkte repräsentierten Punkten*. Daher werden Gummibänder auch als nicht-atomare Segmente bezeichnet. Die Wege sind also *Ketten im Universum geometrischer Objekte* und somit nicht notwendigerweise als Ketten im räumlichen Datenbestand bzw. der interessierenden Domäne vorhanden. Ein Weg darf sich weder selbst überkreuzen noch berühren und somit auch keinen Kreis bilden – wie alle Ketten muß er *einfach* sein.

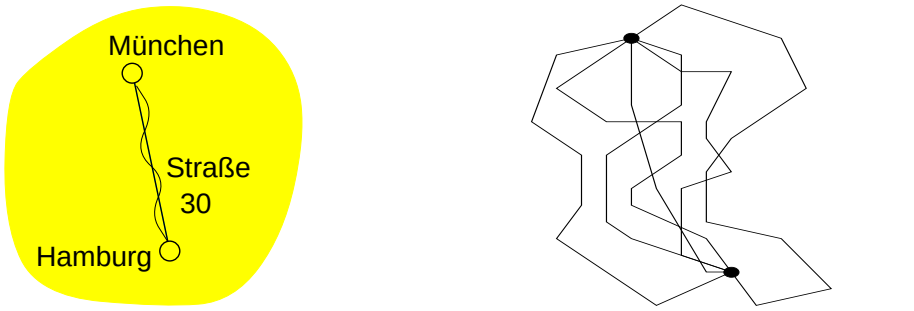
Die „Komponente von“- / „Hat Komponente“-Einschränkungen und Thematik von D-Gummibändern *übertragen sich auf die Komponenten-Segmente des repräsentierten Weges:* hat also ein D-Gummiband die Thematik „Straße“, so muß jedes Segment des repräsentierten Weges diese Thematik tragen. Ist ein Gummiband „Komponente von“ einem Polygon, so muß jedes Segment des repräsentierten Weges Komponente des repräsentierten Polygones sein. Die für das Gummiband abgeleiteten topologischen Einschränkungen wie „Schneidet„, „Schneidet nicht“ etc. beziehen sich jedoch stets auf den gesamten Weg.

Für ein *D-Gummiband* wird deutlich, daß die repräsentierten Wege (also Ketten) *selbst* nicht explizit in Form von Kettenobjekten in der Domäne vorliegen müssen, da ein Gummiband Ketten im Universum geometrischer Objekte und somit nicht notwendigerweise Ketten in der speziellen Teilmenge des räumlichen Datenbestandes bzw. der interessierenden Domäne repräsentiert. Jedoch müssen *die Segmente* des durch ein D-Gummiband repräsentierten Weges explizit im Datenbestand bzw. der interessierenden Domäne vorhanden sein. Ein D-Gummiband repräsentiert somit keine D-Ketten, sondern stets Ketten im Universum geometrischer Objekte („U-Ketten“). Die Segmente dieser Ketten sind jedoch Elemente der Domäne selbst (also explizit) – da das Universum aller geometrischen Objekte u.a. *alle* Ketten enthält, sind diese Ketten dort sicherlich auch vorhanden.

Für ein *D*-Gummiband* ergibt sich aufgrund obiger Interpretation die Einschränkung, daß die verwendeten Segmente der repräsentierten Ketten *Primärobjekte* sein müssen. Objekte, die eine D*-Gummiband-Komponente haben, sind daher stets *Hilfsobjekte* (wie erwähnt, haben Polygone oder Ketten niemals Ketten als Komponenten, sondern stets nur Segmente – s. Abschnitt 5.3.1).

Die Anzahl der Segmente der durch ein Gummiband repräsentierten Ketten kann auf ein Maximum beschränkt werden: eine *Maximumsbeschränkung* von n bedeutet, daß die Kette bzw. der repräsentierte Weg nicht mehr als n Segmente bzw. Streckenkomponenten haben darf. Im implementierten Prototypen stellen Maximumsbeschränkungen für Gummibänder ein wesentliches Sprachmittel zur Eingrenzung der Suchkomplexität dar.

Das Gummiband in Abb. 5.28(a) repräsentiert alle Wege zwischen seinen beiden Endpunkten mit höchsten 30 Segmenten. Einige dieser Wege sind in Abb. 5.28(b) dargestellt.

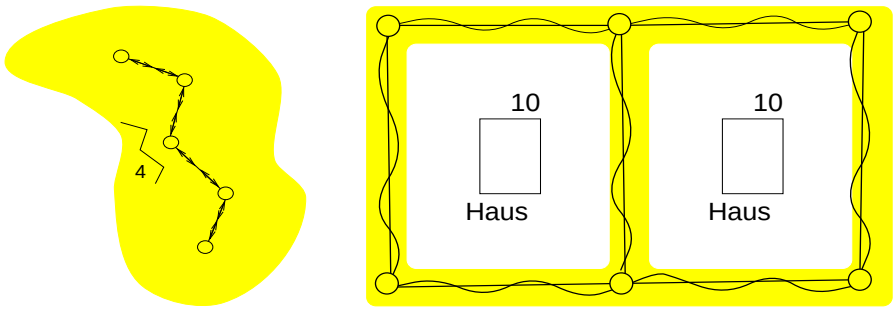


(a) Gummiband zwischen zwei Murmeln (b) Einige der so repräsentierten Wege

Abbildung 5.28: Gummiband und Wege

Die Komponenten eines durch ein *Hilfsgummiband* repräsentierten Weges müssen nicht im Datenbestand bzw. der Domäne vorliegen. Verwendet man VISCO als Anfragesprache, so können Hilfsgummibänder nicht sinnvoll verwendet werden. Wie bereits erwähnt, wird hier die Sichtweise vertreten, daß VISCO-Definitionen *an sich* etwas beschreiben bzw. repräsentieren, unabhängig davon, ob die repräsentierten Objekte nun in einem Datenbestand explizit vorliegen bzw. aufgefunden werden sollen oder nicht (s. Abschnitt 5.1). Die Interpretation von VISCO als Anfragesprache ist somit nicht zwingend, obwohl einige der hier diskutierten Sprachmittel bzw. -elemente nur dann sinnvoll verwendbar sind.

Ketten und Polygone



(a) Eine Kette (b) Zwei adjazente Polygone

Abbildung 5.29: Ketten und Polygone

Eine *Kette* oder ein *Polygon* ist immer eine direkte Komponente einer Folie. Eine Kette oder ein Polygon kann entweder ein *D-Objekt* oder ein *Hilfsobjekt* sein. Ketten und Polygone haben Segmente als Komponenten: hierzu zählen Gummibänder und alle atomaren Strecken. Ketten und Polygone müssen *einfach* sein: sie dürfen sich weder selbst überkreuzen noch berühren. Für Ketten und Polygone wird stets ein ikonischer Repräsentant eingezeichnet (s. Kap. 2). Einschränkungen, die sich auf das ganze Objekt beziehen, werden in der Nähe des ikonischen Repräsentanten eingezeichnet. So kann z.B. die Thematik eines Polygones (z.B. „Haus“) von der Thematik seiner Segmente (z.B. „Hauskante“) getrennt werden.

Eine *D-Kette* oder ein *D-Polygon* repräsentiert Ketten bzw. Polygone der Domäne, während eine *Hilfskette* oder ein *Hilfspolygon* beliebige Ketten bzw. Polygone (aus der Menge aller Ketten bzw. Polygone) repräsentiert.

Die Anzahl der Segmente der durch diese Objekte repräsentierten Ketten bzw. Polygone kann auf ein Maximum beschränkt werden (man bedenke, daß Polygone und Ketten Gummibänder als Segmente haben können): eine *Maximumsbeschränkung* von n bedeutet, daß die repräsentierten Objekte nicht mehr als n Segmente bzw. Streckenkomponenten haben dürfen. Im implementierten Prototypen stellen Maximumsbeschränkungen für Polygone und Ketten ein wesentliches Sprachmittel zur Eingrenzung der Suchkomplexität dar.

In Abb. 5.29(b) sollen zwei aneinandergrenzende Häuser eine nahezu rechteckige Form haben (man könnte auch von *qualitativer Form* sprechen). Die rechteckige Form soll jedoch durch höchstens 10 Streckensegmente beschreibbar sein. Insbesondere könnte man auch den einzelnen Gummibandsegmenten weitere individuelle Maximumsbeschränkungen auferlegen, aber auch eine Thematik wie „Hauskante“ (beides nicht dargestellt). Die Kette in Abb. 5.29(a) könnte das Sternenbild „Kassiopeia“ repräsentieren. An dieser Stelle muß auf einen subtilen Unterschied zwischen Ketten und Polygonen hingewiesen werden: ohne die Maximumsbeschränkung von $n = 4$ könnte die dargestellte Kette in Abb. 5.29(a) beliebig um weitere Segmente erweitert werden und würde somit auch längere Ketten repräsentieren. Im Sternenbild „Kassiopeia“ gibt es aber genau vier Strecken.

Werden Komplexobjekte repräsentiert, so wird also nicht die Annahme gemacht, daß *ausschließlich* die dargestellten Komponenten vorliegen (also keine „closed world assumption“) – stattdessen können die repräsentierten Komplexobjekte evtl. weitere Komponenten haben, für die jedoch keine Repräsentanten angegeben wurden. Während dies für Strecken und (einfache) Polygone kein Unterschied ist (denn Polygone sind „geschlossen“!), so jedoch für Aggregate und Ketten. **Statt einer „es existieren ausschließlich die angegebenen Komponenten“-Semantik liegt hier also stets eine „es existieren mindestens die angegebenen Komponenten“-Semantik zugrunde.**

Man mag sich fragen, worin genau die Unterschiede zwischen Gummibändern und Ketten liegen: zunächst können Gummibänder natürlich *Komponenten* von Ketten sein, Ketten jedoch nicht. Die durch ein Gummiband repräsentierten Wege bzw. Ketten müssen nicht explizit im Datenbestand vorliegen (sie liegen jedoch im Universum explizit vor). Genau dies kann natürlich für D-Ketten gefordert werden. Sollen alle Segmente eines Gummibandes Komponenten *eines* Objektes im Datenbestand sein, so muß daß Gummiband Komponente einer D-Kette werden: wie bereits erwähnt, übertragen sich alle für Gummibänder gemachten strukturellen Einschränkungen – etwa die „Komponente von“-Einschränkung – auf die *Segmente* der durch das Gummiband repräsentierten Kette (die im Universum vorliegt). Zudem wird für Ketten ja eine Mindestanzahl an Segmenten gefordert, was für Gummibänder nicht der Fall ist.

Zeiger und Skala

Ein *Zeiger* kann an einem *Ursprung* oder einem *atomaren Segment* befestigt werden.

Pro Objekt kann höchstens ein Zeiger angebracht werden (Ausnahme s.u.). An einem Zeiger kann höchstens eine *Skala* angebracht werden. Auf einer Skala werden *Skalenmarken und -intervalle* eingetragen: diese visualisieren die erlaubten Stellungen des Zeigers. Eine Skala muß mindestens einen Skalenwert oder -intervall haben. Zeiger und Skala sind Metaobjekte.

Wird ein Zeiger an einem Ursprung befestigt, so werden damit die erlaubten Rotationsbereiche relativ zur Elternfolie (bzw. relativ zum Weltkoordinatensystem, falls es keine Elternfolie gibt) eingeschränkt. Da ein Punkt der Ursprung mehrerer Folien sein darf (s.o.), können für einen solchen Nullpunkt mehrere Zeiger (und somit Skalen) – nämlich einer pro Folie – definiert werden (s. Abb. 5.22(c)).

Wird ein Zeiger an einem atomaren Segment befestigt, so werden die erlaubten Orientierungen des Segmentes relativ zur tragenden Folie eingeschränkt.

Es gibt keine Skala ohne Zeiger. Ein Zeiger ohne Skala hingegen ist erlaubt und schränkt die möglichen Orientierungen auf genau eine ein, nämlich die dargestellte.

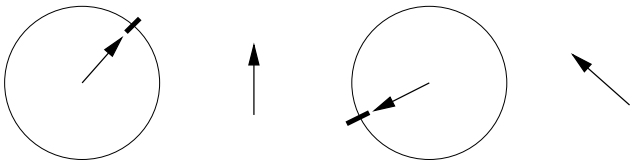
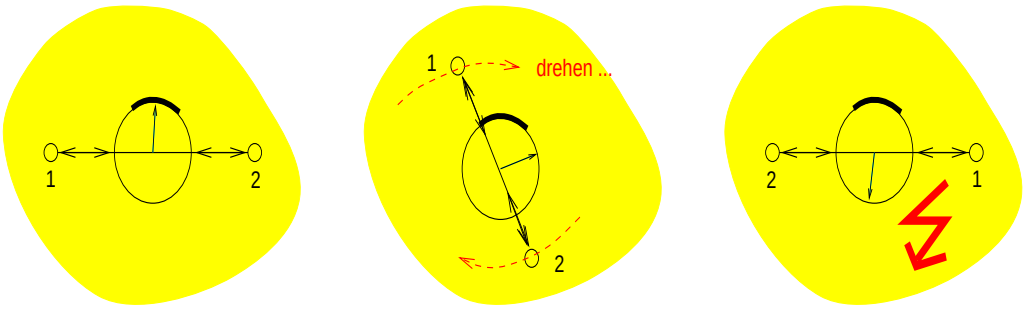


Abbildung 5.30: Äquivalente Orientierungseinschränkungen

Die Skala behält nun stets die eingezeichnete Ausrichtung relativ zur tragenden Folie oder Elternfolie bei. Der Zeiger jedoch behält relativ zu dem Objekt, an dem er befestigt ist, die gleiche Lage, während die Skala ihre Orientierung relativ zur Folie bzw. Elternfolie beibehält – insofern besteht gewisse Ähnlichkeit zu einem „inversen“ Kompaß. Da es nur auf die Relation zwischen Zeiger und Skala ankommt, sind alle Darstellungen in Abb. 5.30 äquivalent.



(a) (b) (c)

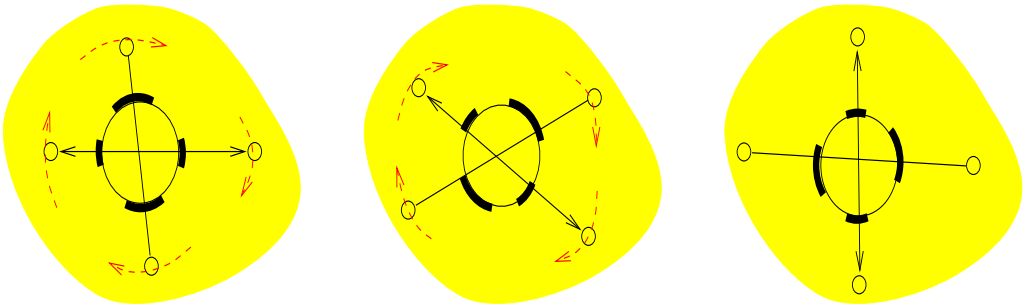
Abbildung 5.31: Illustration Zeiger und Skala

Wird ein Zeiger an einem atomaren Segment befestigt, so muß beachtet werden, daß hiermit dem Segment eine *Richtung* gegeben wird (wie ein Vektor): während in Abb. 5.31(a) die Positionen der Endpunkte der dargestellten Teleskopantenne ohne eine Orientierungseinschränkung beliebig sein dürften, wird durch die Orientierungseinschränkung auch verlangt, daß Murmel 1 immer „links“ von Murmel 2 sein muß. Versucht man nun, durch „mentale Animation“ die Telekopantenne so zu drehen, daß Murmel 1 rechts von Murmel 2 zu liegen kommt (s. Abb. 5.31(b)), so wird klar, daß der Zeiger dann genau in die entgegengesetzte Richtung zeigt, was nicht erlaubt ist, da für diese Zeigerstellung keine Skalenmarke eingetragen ist (Abb. 5.31(c)).

Winkeleinschränker

Der *Winkeleinschränker* wird zwischen zwei atomaren (sich kreuzenden oder berührenden) Segmenten befestigt (die beiden Segmente dürfen auch einen gemeinsamen Komponentenpunkt haben). Die beiden Segmente müssen also in der topologischen Relation „Schneidet“ stehen – diese darf nicht ignoriert bzw. relaxiert worden sein, so daß tatsächlich die Einschränkung „Schneidet“ zwischen den beiden individuellen Strecken vorliegen muß. Zudem muß der Schnitt *nulldimensional* sein und die beiden atomaren Segmente müssen auf der selben Folie liegen. Zwischen je einem Paar atomarer Segmente derselben Folie darf höchstens ein Winkeleinschränker konstruiert werden.

Der Winkeleinschränker ist ein Metaobjekt und schränkt die möglichen relativen Orientierungen bzw. Winkel zwischen zwei Segmenten bzw. Strecken ein. Der Winkel kann auch vollständig festgelegt werden.



(a) (b) (c)

Abbildung 5.32: Illustration Winkeleinschränker

Anders als die Skala, die ihre relative Ausrichtung bzgl. der tragenden Folie stets beibehält, paßt sich der Winkeleinschränker bei Bedarf den Lagen der Argumente an (im Rahmen der durch ihn erlaubten Winkeltoleranzen). Es muß beachtet werden, daß durch die Festlegung oder Einschränkung der relativen Orientierung bzw. des Winkels zweier Strecken diesen auch eine Richtung gegeben wird (s. auch Zeiger und Skala). Einige Zwischenbilder einer „mentalen Animation“ zeigt Abb. 5.32.

Thematik von D-Objekten

Die *Thematik eines D-Objektes* wird einfach durch einen beschreibenden Text in der Nähe des entsp. Objektes dargestellt (wie „Haus“). Dieses Textobjekt ist ein Metaobjekt.

Ein D-Objekt kann eine beliebige Anzahl von solchen thematischen Eigenschaften haben. Inkonsistenzen werden (momentan noch) nicht erkannt: so kann man einer Kette sowohl die Thematik „Straße“ als auch die Thematik „Fluß“ geben.

Die Thematik von geometrischen Hilfsobjekten (U-Objekten) kann nicht festgelegt werden, da sie keine Objekte der Domäne, sondern Objekte des Universum geometrischer Objekte repräsentieren.

Maximumsbeschränkungen

Die *Maximumsbeschränkung* eines geometrischen Objektes wird einfach durch eine Zahl in der Nähe des Objektes dargestellt, die ein Metaobjekt ist.

Maximumsbeschränkungen gibt es in zwei Arten: zum einen kann für Gummibänder, Ketten, Polygone und Folien die maximale Anzahl der entsp. Komponenten des so repräsentierten Objektes

festgelegt werden (s.o.), zum anderen können Gebiete mit Maximumseinschränkungen versehen werden – diese legt dann die maximale Anzahl möglicher enthaltener primärer Domänenobjekte fest. U.a. können so objektfreie Zonen deklariert werden: in dem in Abb. 5.33 dargestellten Gebiet dürfen keine Domänen-Objekte (des Datenbestandes) enthalten sein. Da ein beliebiges Gebiet natürlich eine *unendliche Anzahl Objekte des geometrischen Universums enthält,* kann sich die Maximumsbeschränkung nur auf Objekte der betrachteten (endlichen) Domäne beziehen.

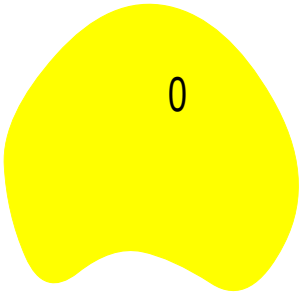


Abbildung 5.33: Ein Gebiet, daß keine Domänenobjekte enthalten darf

Natürlich sollten alle Maximumsbeschränkungen konsistent mit dem dargestellten Objekt selbst sein (ein Polygon mit vier Seiten darf keine Maximumsbeschränkung von „3“ erhalten). Legt man die maximale Anzahl Segmente eines Gummibandes auf „1“ fest, so hat man eine Teleskopantenne.

5.4 Beispiele

Die folgenden Beispiele stellen zwar legale VISCO-Definitionen dar, es wird jedoch stets nur das letzte Diagramm der Konstruktionsgeschichte gezeigt, da keine Mehrdeutigkeiten aufgelöst werden müssen. Ergänzende Beschreibungen diskutieren einige Problempunkte. Die natürlichsprachlichen Titulierungen der Definitionen und auch Umschreibungen sollten nicht allzu genau genommen werden (die entsp. Problempunkte hierbei wurden ja bereits diskutiert).

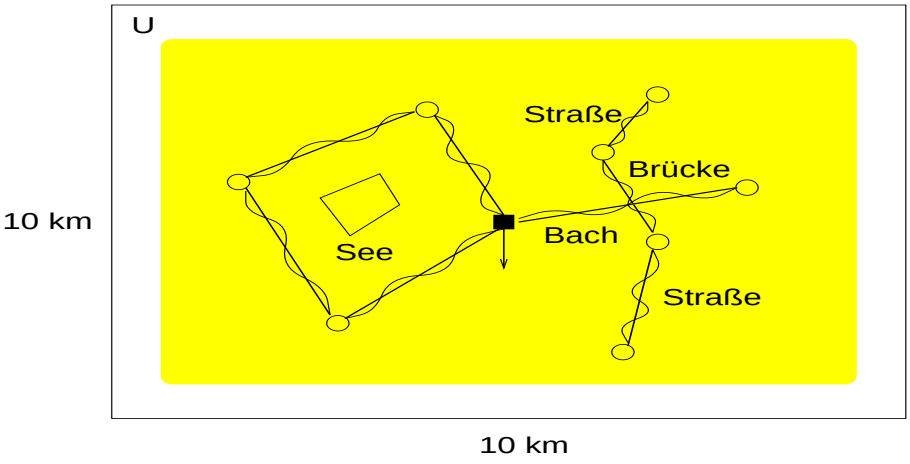


Abbildung 5.34: Ein Bach, der in einen See fließt und von einer Brücke überquert wird

1. Ein Bach, der in einen See fließt und von einer Brücke überquert wird

Die Objekte sind auf einer 10 km^2 großen Folie definiert, die weder skalier- noch rotierbar ist, s. Abb. 5.34. Die Folie ist mit einem „U“ annotiert: daher liegt dieses Aggregat u.U. nicht explizit im Datenbestand vor, sondern muß „zusammengesucht“ werden. Es sei angenommen, daß es sich

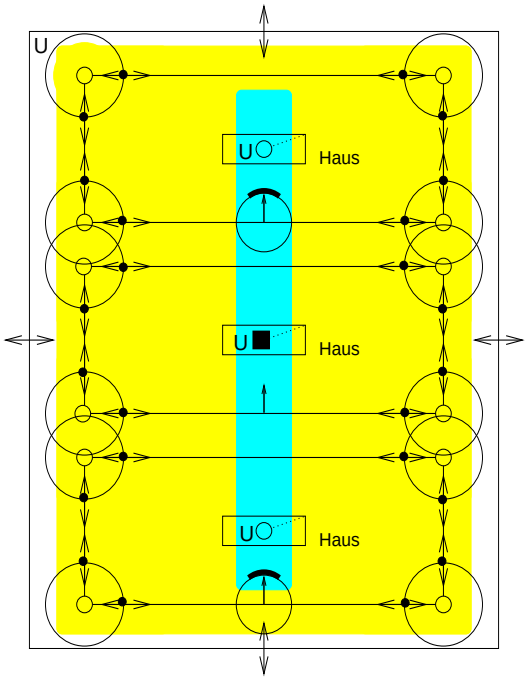


Abbildung 5.35: Drei in einer Linie liegende, nahezu parallel orientierte rechteckige Häuser

bei dem zugrundeliegenden Datenbestand um topologisch-strukturierte Vektordaten (s. Kap. 3) handelt: alsdann kann man davon ausgehen, daß sich der Bach und der See tatsächlich in einem gemeinsamen Knoten treffen. Für die „Bach“- und „Brücke“-Gummibänder wird gefordert, daß sie sich schneiden. Dies ist natürlich auch der Fall, wenn sie einen gemeinsamen Knoten haben (da es sich hier um topologisch strukturierte Vektordaten handelt, ist letzteres – und somit ersteres – der Fall).

2. Drei in einer Linie liegende, nahezu parallel orientierte rechteckige Häuser

Hier sind drei Rechtecke dargestellt, die Häuser repräsentieren sollen. Die Abmessungen der Rechtecke können variieren, solange sie rechteckig bleiben (s. Winkeleinschränker), s. Abb. 5.35. Während das mittlere Rechteck relativ zur Folie seine Orientierung beibehält (s. Pfeil), können das obere und untere Rechteck geringfügig in der Orientierung variieren (s. Pfeil und Skala). Die Mittelpunkte dieser Objekte sind auf ein kleineres Gebiet als die Eckmurmeln eingeschränkt: die Mittelpunkte sollen nicht zu sehr von einer gemeinsamen Ausrichtungslinie abweichen. Die Mittelpunkt müssen nicht im Datenbestand vorhanden sein (es handelt sich um Hilfs- bzw. U-Objekte): hier zeigt sich wieder, daß geometrische Hilfsobjekte nützlich zum Konstruieren weiterer Einschränkungen sind. Hilfsmurmeln sind nur dann sinnvoll verwendbar, wenn sie in einer sehr speziellen Relation stehen (wie z.B. „Mittelpunkt von“, was hier der Fall ist). Die Folie selbst ist beliebig rotier- und skalierbar und das durch sie repräsentierte Aggregat nicht notwendigerweise explizit im Datenbestand vorhanden. Der Ursprung der Folie ist gleichzeitig der Mittelpunkt des mittleren Rechteckes. Die anderen beiden Rechtecke müssen sich also bzgl. des mittleren Rechteckes ausrichten bzw. orientieren – dem mittleren Rechteck kommt die Rolle eines „Referenzobjektes“ zu.

3. Eine Stadt in der Nähe eines Grenzflusses

Zwei adjazente Gummiband-Polygone sollen Länder repräsentieren, s. Abb. 5.36. Die beiden Länder sollen eine gemeinsame Grenze haben, die hier durch eine gemeinsame Gummibandkomponente der beiden Polygone repräsentiert wird. Kongruente Strecken(verläufe) werden in topologisch strukturierten Vektordatenbeständen ausgeschlossen und durch gemeinsame Kanten repräsentiert,

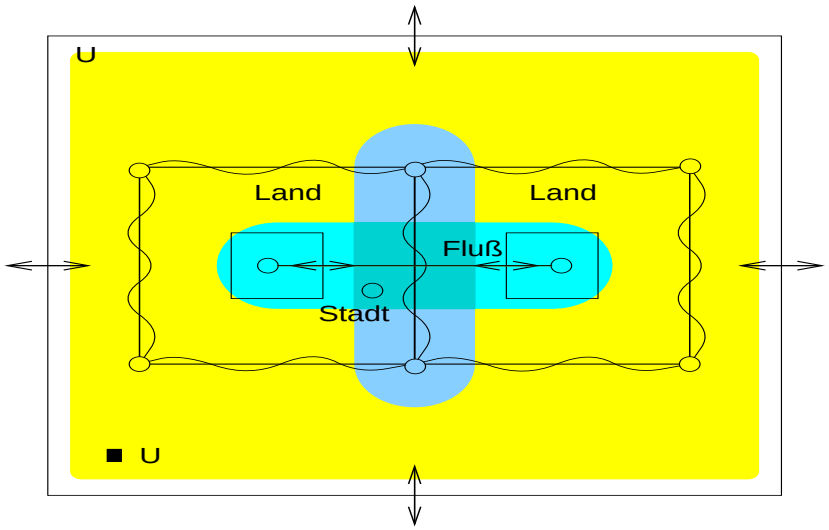


Abbildung 5.36: Eine Stadt in der Nähe eines Grenzflusses

so daß hier die Relation „adjazent“ durch die Relation „gemeinsame(r) Kante(nverlauf) von“ repräsentiert wird. Auch hier tritt also wieder das Problem „Identität“ vs. „Gleichheit“ bzw. „Kongruenz“ auf: eine gemeinsame Gummibandkomponente setzt einen topologisch strukturierten Datenbestand voraus. Ein Fluß soll nun die Grenze schneiden: sowohl um Fluß als auch Grenze werden transparente ϵ-Gebiete etabliert. Die Stadt muß somit in beiden ϵ-Gebieten enthalten sein. Ein Problem ergibt sich durch die Formvariabilität der Gummiband-Polygone in Verbindung mit den ϵ-Umgebungen: letztlich können die Polygone bzgl. der tragenden Folie natürlich beliebig klein werden (die Folie müsste dann entsp. groß skaliert werden), und die ϵ-Gebiete würden dann bzgl. der Größe der Polygone sehr große Gebiete überdecken. Von „in der Nähe“ kann dann nicht mehr gesprochen werden. Das Problem ergibt sich dadurch, daß die Polygone formvariabel sein sollen – doch Formvariabilität setzt Unbestimmtheit bzgl. der tragenden Folie voraus. Die ϵ-Umgebungen beziehen sich jedoch auf die Metrik der tragenden Folie, und nicht auf die Größe ihrer Argumente. Letztlich können ϵ-Umgebungen nur dann sinnvoll verwendet werden, wenn das Argumentobjekt bzgl. der tragenden Folie feste Form bzw. Größe hat. Verwendet man VISCO als Anfragesprache (wie in Abschnitt 5.1. diskutiert ist diese Interpretation nicht zwingend), so könnte man sich jedoch wieder vorstellen, daß Anfrageergebnisse bzgl. eines Ähnlichkeitsmaßes zur dargestellten Anfrage sortiert werden. Dann würde man zumindest Städte, die tatsächlich in der Nähe liegen, zuerst erhalten.

4. Eine Stadt in der Nähe der Landesmitte

Auch in diesem Beispiel tritt das Problem der formvariablen Objekte in Verbindung mit einer ϵ-Umgebung (um den Ursprung) auf, s. Abb. 5.37. Der Ursprung ist nicht notwendigerweise im Datenbestand vorhanden und gleichzeitig Mittelpunkt des formvariablen Polygones, welche ein Land repräsentieren soll. Hier wird demonstriert, daß bereits ein Gummiband ausreicht, um nahezu beliebige Polygone zu definieren. Durch die Geometrie der konstanten Gebiete wird nun jedoch erzwungen, daß das formvariable Polygon nicht kleiner als der „ausgestanzte weiße mittlere Kreis“ werden kann. Somit wird stets ein Mindestabstand zwischen der Stadt in der Nähe des Ursprunges bzw. Mittelpunktes und den Ländergrenzen des Landes erzwungen. Leider wird hierdurch aber auch die Anzahl möglicher Formen für die repräsentierten Länder stark eingeschränkt (da ein Kreis in dieses Polygon „hineinpassen“ muß). Die Folie selbst soll zusätzlich proportional skalierbar und eine Mindestausdehnung von 500 km^2 haben (zu kleine Länder werden ausgeschlossen).

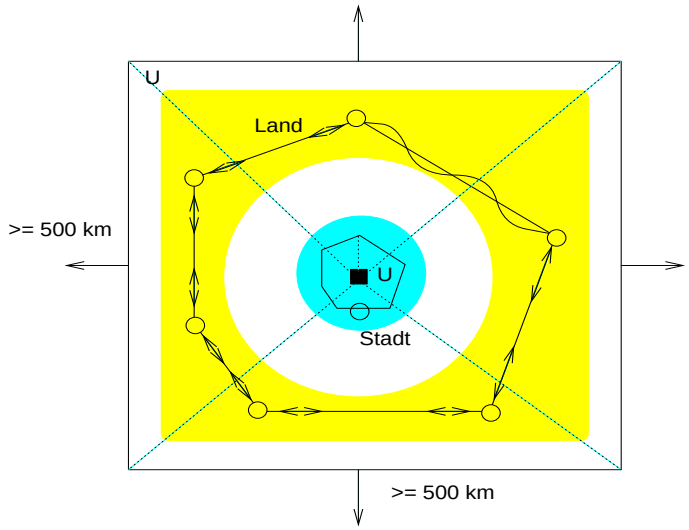
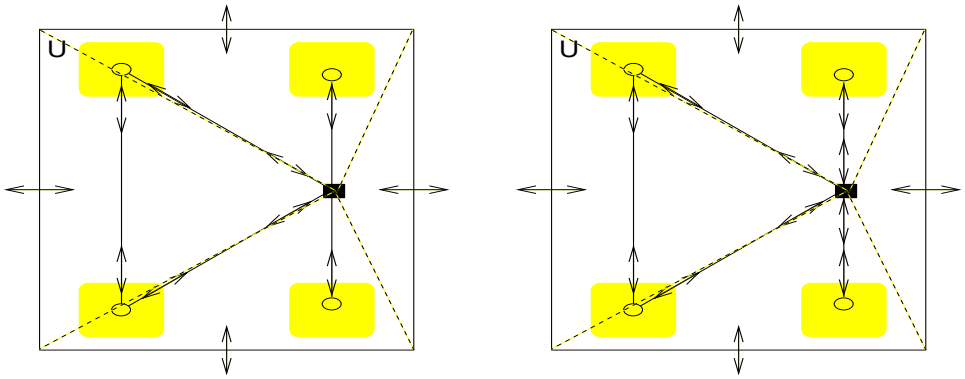


Abbildung 5.37: Eine Stadt in der Nähe der Landesmitte



(a) (b)

Abbildung 5.38: Eine Diode in einem CAD-Schaltplan

5. Eine Diode in einem CAD-Schaltplan

Dieses Beispiel verdeutlicht, daß auch relativ konkrete Formen u.U. von Belang sein können, s. Abb. 5.38. Kleinere Abweichungen der Eckpunkte bzw. Variationen von der Dreiecksgestalt werden jedoch zugelassen (vielleicht wurde hier eine Papierzeichnung vektorisiert). Wieder stellt sich die Frage, wie die Objekte bzgl. der „Komponente von“ bzw. „hat Komponente“ genau repräsentiert sind: in Abb. 5.38(a) liegt der Ursprung auf der senkrechten Linie (Katode), in Abb. 5.38(b) ist der Ursprung hingegen Komponente zweier miteinander (über ihn) verbundener Strecken. Es sei angenommen, daß Polygone nicht explizit vorliegen. Zudem müssen zwei an den Endpunkten aneinanderstoßende Strecken tatsächlich über ein gemeinsames Punktobjekt verknüpft sein.

6. Ein Haus am See

Das in Abb. 5.39 dargestellte Konzept bedarf keiner weiterer Erläuterungen. Auch hier stellt sich wieder die Frage, ob die Bedeutung der Präposition „am See“ von der Ausdehnung bzw. Größe des Sees abhängt. Man beachte, daß hier „am“ durch „im Abstand von ca. 150 m“ präzisiert wurde. Schrumpt der See jedoch auf die Größe eines Gartenteiches, so treten die bereits diskutierten Probleme auf. Unabhängig davon ist jedoch das kontextfreie Konzept „im Abstand von ca. 150 m“ sinnvoll (und somit die ϵ-Gebiete).

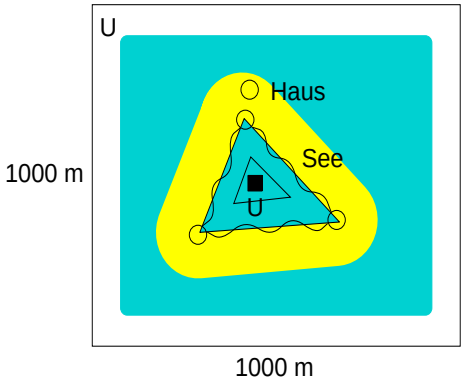


Abbildung 5.39: Ein Haus am See

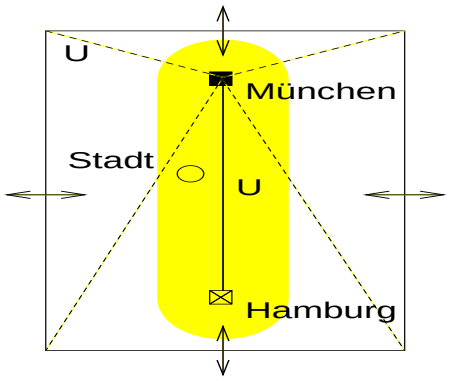


Abbildung 5.40: Eine Stadt in der Nähe der Luftlinie zwischen Hamburg und München

7. Eine Stadt in der Nähe der Luftlinie zwischen Hamburg und München

Die in diesem Beispiel – s. 5.40 – verwendete Luftlinien-Strecke (die natürlich nicht im Datenbestand vorhanden ist) wird ausschließlich zur Konstruktion der ϵ-Umgebung benötigt. Man beachte, daß diesesmal die Ausdehnung des ϵ-Gebietes anhand des Argumentobjektes bestimmt wird und die Folie zudem ausschließlich proportional skaliert werden kann. Schließlich soll eine Stadt in dieser Umgebung liegen. Man könnte nun auch Hamburg und München über diese Stadt durch zwei Straßen-Gummibänder miteinander verbinden: die so repräsentierten Wege zwischen Hamburg und München würden dann stets innerhalb der ϵ-Umgebung verlaufen und somit in jedem Punkt des Weges „nicht zu sehr von der Luftlinie entfernt“ verlaufen. Dennoch könnten die Wege aber beliebig hin-und-her oszillieren.

Kapitel 6

Struktur und Implementation des VISCO-Systemes

Nachdem in Kap. 5 die Sprache VISCO vorgestellt wurde, soll in diesem Kapitel der implementierte Prototyp diskutiert werden.

Unter einem „System“ wird hier nach der in der Informatik breit genutzten Definition von J. W. Forester eine Menge miteinander in Beziehung stehender Teile verstanden, die zu einem gemeinsamen Zweck interagieren.

Während VISCO in Kap. 5 als Sprache zur Definition räumlicher Konstellationen definiert wurde, soll in diesem Kapitel ausschließlich die Anwendbarkeit als visuelle räumliche Anfragesprache demonstriert werden. Es ist klar, daß eine effektive Nutzung dieser nur in einem integrativen Gesamtzusammenhang mit anderen Systemkomponenten geschehen kann (s. Kap. 7).

Zunächst wird ein logisches Architekturmodell für das Gesamtsystem vorgestellt – dieses Modell spannt einen Integrationsrahmen für die einzelnen Systemkomponenten auf. Die physikalische Struktur des Prototypen entspricht einer konkreten Ausprägung des Rahmens: Die einzelnen logischen Schichten bzw. Komponenten entsprechen (teilweise) implementierten Modulen des Prototypen. Insbesondere wurde auf eine softwaretechnisch saubere Trennung von Aufgabenbereichen und Zuständigkeiten Wert gelegt. Besonderes Gewicht wird auf die Vorstellung des syntaxgesteuerten Grafikeditors zum Konstruieren von Anfragen sowie des optimierenden Compilers gelegt.

Die Diskussion der Interna wird auf einer relativ abstrakten Ebene erfolgen – es ist nicht sinnvoll, Hunderte interner Details vorzustellen. Hier werden eher grundsätzliche architekturelle Entscheidungen und deren Umsetzung diskutiert.

6.1 Systemstruktur (Architekturmodell)

Abbildung 6.1 zeigt das logische Architekturmodell des VISCO-Systems. Prinzipiell sind zwei logische Schichten vorgesehen: Der „Räumliche Datenbestand“ sei als Abstraktion einer räumlichen Datenbank, eines GIS o.ä. betrachtet. Die von dieser Schicht verwalteten räumlichen Daten werden von Komponenten der „Anwendungsebene“ manipuliert, inspiziert und ausgewertet.

6.1.1 Räumlicher Datenbestand

Als Anfrageziel wird natürlich ein räumlicher Datenbestand benötigt. Dieser könnte z.B. in einem geographischen, räumlichen oder geometrischen Informationssystem (GIS, RI oder GI) gespeichert sein.

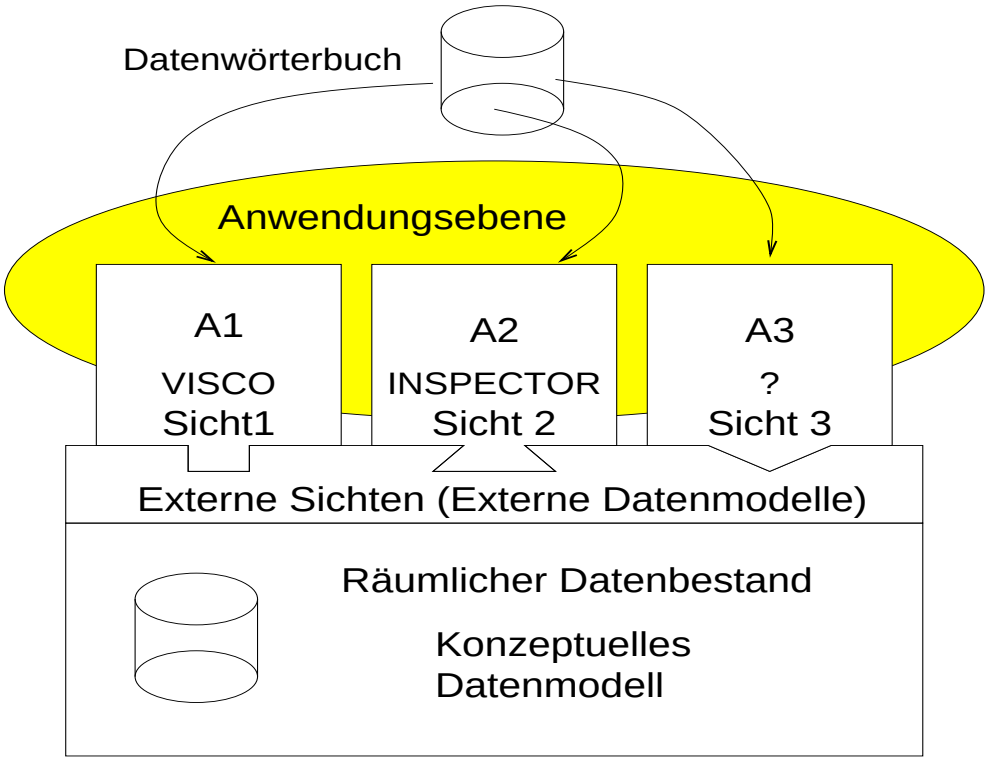


Abbildung 6.1: Architekturmodell

Etwas verallgemeinernd kann man sagen, daß ein *räumliches Informationssystem* ein Informationssystem ist, welches einen anwendungsneutraleren Rahmen für die räumlichen Daten und Anwendungen aufspannt als ein GIS. Somit sind die räumlichen Daten in einem GI oder RI nicht notwendigerweise geographische Objekte (Geo-Objekte). Daher wird einem GI bzw. RI meist charakteristische GIS-Funktionalität fehlen.

Bereits in Kap. 3 wurde die u.a. als Basismaschine für vektorbasierte GIS geeignete *räumliche Datenbank* lt. Güting wie folgt charakterisiert ([Güt94]):

1. A spatial database system is a database system.
2. It offers spatial data types (SDTs) in its data model and query language.
3. It supports spatial data types in its implementation, providing at least spatial indexing and efficent algorithms for spatial join.

Im Rahmen dieser Arbeit ist es nicht relevant, wie die Daten genau verwaltet und gespeichert werden, solange das in Kap. 5 vorgestellte logische „vektorbasierte“ Datenmodell *extern* verwendet wird, denn VISCO kann nur mit diesen Objekten operieren. Wesentlich ist also, daß eine entsp. *externe Sicht* auf die verwalteten räumlichen Daten zu Verfügung gestellt wird. Kann die verlangte externe Sicht auf die Daten nicht zur Verfügung gestellt werden, so gibt es immer noch die Möglichkeit, einen speziellen VISCO-Compiler für das verwendete Datenmodell zu implementieren.

Diese Sichtweise ist konform mit den üblichen Annahmen im Datenbankbereich (s. z.B. [Dat95, LS87] für eine Diskussion der Drei-Ebenen-ANSI/SPARC-Architektur): konventionelle Datenbankmanagementsysteme (DBMS) schaffen es, einen anwendungsunabhängigen Rahmen für konventionelle (alphanumerische) Anwendungen aufzuspannen. Sicherlich wird dies in naher Zukunft auch für räumliche Daten gelingen – eine räumliche Datenbank sollte einen anwendungsunabhängigen Rahmen für raumbezogene Anwendungen bereitstellen.

Ein wichtige Rolle wird auch das sog. *Datenwörterbuch (Data Dictionary, neuerdings auch Repository)* der räumlichen Datenbank spielen (s. [Dat95, LS87]): hier werden u.a. die Metadaten über den räumlichen Datenbestand abgelegt, also Daten, welche die Daten selbst beschreiben. Sollen mit VISCO Anfragen gestellt werden, so ist u.a. die Kenntnis der verwendeten thematischen (oder semantischen) Typen des räumlichen Datenbestandes notwendig – es macht keinen Sinn, ein „Haus am See“ in einem Datenbestand von CAD-Schaltplänen zu suchen. Mit Hilfe des Datenwörterbuches können Metaanfragen wie „Gibt es überhaupt Häuser in diesem Datenbestand?“ im vornherein beantwortet werden. Auch wird das verwendete *externe Datenmodell* im Datenwörterbuch dokumentiert sein. Somit ist das Datenwörterbuch für alle externen Anwendungen von essentieller Bedeutung, da hier alle Annahmen, die die Anwendungen über die Daten machen können, dokumentiert werden.

6.1.2 Anwendungsebene

In der Anwendungsebene sind verschiedenste Anwendungen beheimatet, die die Daten auf unterschiedlichste Art und Weise nutzen – in der Regel werden die Anwendungen auf einer externen Sicht dieser Daten arbeiten, so daß die Datenunabhängigkeit zwischen dem Datenbestand und den Anwendungen gewährleistet ist.

So finden sich in einem GIS in der Regel spezielle Anwendungskomponenten zur Erfassung, Verwaltung, Analyse und Präsentation der Daten (s. Kap. 3). Erst durch ihr Zusammenspiel mit einem menschlichen Benutzer kann man von einem *Informationssystem* sprechen.

Eine in allen Anwendungsszenarien benötigt Komponente wird eine grafische Inspektionskomponente sein: mit ihrer Hilfe kann der Benutzer einen Überblick über die gespeicherten Daten gewinnen. Da es sich hier um *inhärent räumliche Daten* handelt, ist es natürlich, daß die Inspektionskomponente grafische Darstellungen verwendet. Im GIS-Bereich bietet sich eine Kartenmetapher für die Darstellung an – der Benutzer kann so Teile der Karte ein- und ausblenden, einzelne Geo-Objekte inspizieren, den Ausschnitt der Karte verändern, die Darstellungsgenauigkeit variieren, etc. (s. [Woo93, Ege90]).

VISCO selbst soll als spezielle Anwendung in der Anwendungsebene vorgesehen werden, mit deren Hilfe interessante räumliche Konstellationen aufgefunden werden können. Da es sich um eine externe Anwendung handelt, muß der VISCO-Compiler letztlich ein Programm erzeugen, welches Anweisungen in der Anfragesprache der räumlichen Datenbank absetzt, um die interessierenden Konstellationen aus dem Datenbestand zu erhalten. Hierzu ist natürlich die Kenntnis des externen Datenmodelles notwendig.

Die spezielle Anwendung VISCO

Abbildung 6.2 zeigt eine Verfeinerung der VISCO-Komponente, so daß die logische Architektur von VISCO selbst sichtbar wird. Hier sind die bereits erwähnten – und im folgenden diskutierten – fünf Komponenten eingezeichnet. Ersichtlich arbeiten sie alle auf einer gemeinsamen Schicht, die als *abstrakter Syntaxgraph (ASG)* bezeichnet wird: hierbei handelt es sich um eine abstrakte Repräsentation der gerade vom Benutzer bearbeiteten VISCO-Definition (s.u.). Das verwendete Architekturmodell von VISCO wird auch als „Repository Model“ bezeichnet ([Som95, Kap. 13]); „Repository“ meint hier jedoch nicht das Datenwörterbuch der verwendeten räumlichen Datenbank!). Die ASG-Schicht ist also das Repository der VISCO-Anwendung.

Vor- und Nachteile dieses logischen Architekturmodelles sind lt. Sommerville ([Som95, Kap. 13]):

- Aufgrund eines gemeinsamen zentral verwalteten Datenbestandes tritt keine Notwendigkeit auf, größere Datenmengen zwischen Subsystemen auszutauschen. Integritätsprobleme durch Datenredundanz treten nicht auf.
- Nachteilig ist jedoch, daß unter allen das Repository benutzenden Komponenten Einigkeit

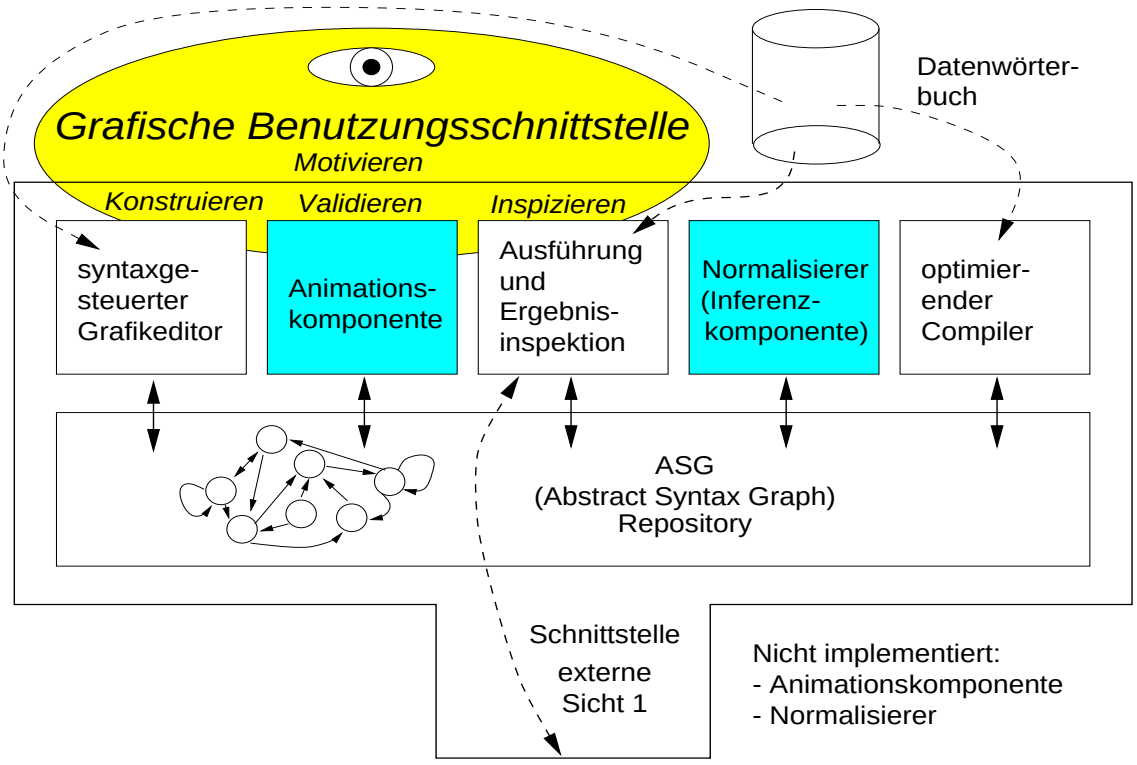


Abbildung 6.2: Architektur von VISCO

bzgl. des verwendeten Datenmodelles herrschen muß (keine Datenunabhängigkeit). Natürlich kann aber eine spezielle Komponente nur den für sich relevanten Teil des Datenmodelles nutzen, oder auch eine eigene Sicht darauf herstellen und lokal verwalten (wobei jedoch wieder Probleme durch Datenredundanz auftreten können).

- Es ist einfach, neue Subsysteme in das bestehende System zu integrieren oder auszutauschen.

Wie bereits diskutiert ist die Kenntnis von Metadaten des räumlichen Datenbestandes essentiell, denn folgende Fragen müssen beantwortet werden:

- Welches externe logische Datenmodell wird zur Verfügung gestellt, und wie kann es (vom Compiler) auf das VISCO-Datenmodell abgebildet werden?
- Welche Daten sind überhaupt im Datenbestand vorhanden (Häuser, Flüsse, etc.)?
- Wie sehen die Ausprägungen der Daten im Datenmodell aus (werden Flüsse als Ketten oder komplexe Flächen repräsentiert)?
- Welche räumlichen Relationen werden direkt durch die Anfragesprache der räumlichen Datenbank unterstützt (m.a.W., kann überhaupt eine Anfrage wie „liefere alle das Objekt xyz schneidenden Objekte“ gestellt werden, oder muß die räumliche Relation „schneidet“ anders umschrieben werden, wozu vom Compiler spezieller Code erzeugt werden müsste)?

In vielen Fällen mag – wie bereits erwähnt – eine speziell für die VISCO-Anwendung erstellte externe logische Sicht auf den räumlichen Datenbestand helfen.

Syntaxgesteuerter Grafikeditor: Der *syntaxgesteuerte (objektorientierte) Grafikeditor* ist für den Benutzer die wichtigste Komponente, da von seiner Gestaltung und Unterstützung die Sy-

stemakzeptanz des Benutzers abhängt – die Oberfläche *ist* das System (s. auch [Fra93]). Mit ihr kann der Benutzer interaktiv VISCO-Anfragen konstruieren.

Wichtige Ziele bei der Gestaltung einer grafischen Benutzungsoberfläche sind (frei) nach Foley et al. ([FDFH96, Chapter 8, 9, 10], s. auch [Cha90, Chapter 2]):

- **Konsistenz:** Das System sollte so gestaltet werden, daß einige wenige einheitliche Regeln (mit möglichst wenigen Ausnahmen) zur Bedienung des Systemes ausreichen. Konsistenz ermöglicht dem Benutzer, über die Grenzen seines bereits vorhandenen Systemwissens hinaus zu generalisieren. Hierzu gehören einheitliche Tastenbelegungen, konsistente Farbgebungen, einheitliche Dialoggestaltung, etc.

- **Rückkopplung (Feedback):** Die Interaktion mit dem System sollte so gestaltet werden, daß stets Rückkopplungen der Benutzeraktionen vom System erzeugt werden, z.B. in Form von Fehler- oder Bestätigungsmeldungen, Listen von anwendbaren Operationen, etc.

- **Minimierung möglicher Fehlbedienungen:** Es sollten keine „Fallen“ für den Benutzer aufgestellt werden – eine im momentanen Kontext nicht sinnvoll anwendbare Operation sollte gesperrt werden. Es ist nicht sinnvoll, den Benutzer eine nichtanwendbare Operation auswählen zu lassen, um ihm dann eine Fehlermeldung bzgl. der (momentanen) Nichtanwendbarkeit dieser Operation zukommen zu lassen.

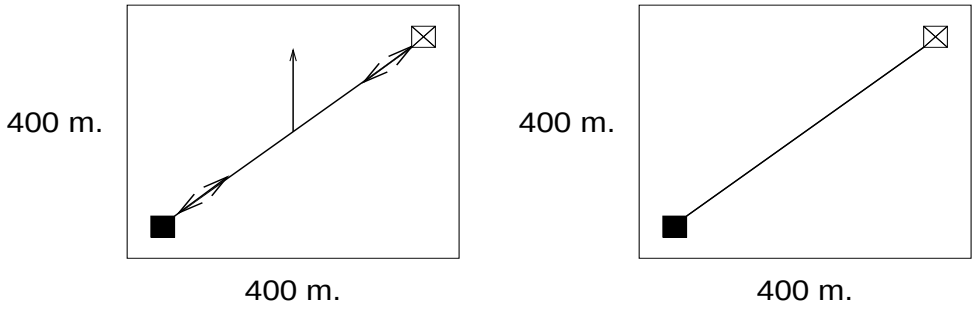
- **Fehlererholung:** Versehentlich ausgeführte Operationen sollten rückgängig gemacht werden können. Hierzu gehören die bekannten UNDO- und REDO-Operationen, sowie CANCEL zum Abbruch einer gerade stattfindenen Handlung bzw. Interaktion. Diese Operationen ermöglichen es dem Benutzer, *spielerisch durch Versuch und Irrtum mit dem System zu experimentieren* und so seine Benutzung zu erlernen.

- **Eignung für verschieden erfahrene Anwender:** Während Neulinge bei der Benutzung eines Systemes gerne auf (ikonische) Menüs zurückgreifen, sollten für erfahrenere Benutzer schnellere Interaktionsformen, z.B. über speziell belegte Tasten (Accelerators) bzw. Tastenkombinationen zur Verfügung gestellt werden (man denke an den bekannten „EMACS“-Editor). Auch eine Kommandozeile ist für erfahrenere Benutzer sinnvoll.

- **Entlastung des Benutzergedächtnisses:** Zum Gebrauch des Systemes sollte der Benutzer sich so wenig Dinge wie möglich merken müssen. Hier spielen natürlich Menüs eine wichtige Rolle, da zur Anwendung einer Operation deren Name nicht memoriert, sondern einfach ausgewählt werden kann. Auch ein kontextsensitives Hilfesystem sollte vorgesehen werden.

Diese Kriterien wurden bei der Gestaltung der Benutzungsoberfläche zu berücksichtigen versucht – das sehr wichtig Hilfesystem konnte jedoch nicht realisiert werden. Viele dieser Eigenschaften lassen sich durch Verwendung eines sehr mächtigen UIMS (User Interface Management Systems) wie CLIM (Common LISP Interface Manager, [Fra94]) relativ einfach implementieren – nichtsdestotrotz findet keine Unterstützung des Designprozesses statt. Eine umfassende Diskussion von UIMS findet man in [Möl97].

Da es sich hier um einen syntaxgesteuerten Grafikeditor handelt, kann und darf der Benutzer nicht zu jedem Zeitpunkt beliebige Operationen auf den dargestellten Objekten vornehmen. Letzlich wird durch die Interaktionen des Benutzers beim Konstruieren einer VISCO-Anfrage intern im ASG-Repository eine abstrakte syntaktische Repräsentation der Anfrage erzeugt (s. auch Kap. 2). Der abstrakte Syntaxgraph (ASG) wird also vom Grafikeditor erzeugt und dann auch von den anderen drei VISCO-Komponenten genutzt. Der Grafikeditor muß nun so gestaltet werden, daß stets *syntaktisch korrekte* VISCO-*Definition erzeugt werden.* Während der Benutzer also auf der Ebene der konkreten Syntax einer visuellen Sprache mit deren Sprachelementen operiert, manipuliert er letztlich die internen abstrakten Repräsentanten dieser Objekte in der ASG-Schicht.



(a) Anfrage (b) Normalisierte Anfrage

Abbildung 6.3: Normalisierungen

Während z.B. die Operation zum Verschieben eines Objektes für einen konventionellen Grafikeditor trivial zu implementieren ist, erfordert die Ausführung der selben Operation in einem syntaxgesteuerten Grafikeditor u.U. die komplette Rekonstruktion des abstrakten Syntaxgraphen. Ähnliches gilt auch für die Entferne (Delete)-Operation.

Operationen wie diese (mit eventuell gravierenden Auswirkungen) werden als *kritisch* bezeichnet. In jedem Fall muß eine mehrstufige Funktionalität zum Rückgängigmachen von Operationen vorgesehen werden (mehrstufiges UNDO und REDO).

Compiler: Der *Compiler* hat die Aufgabe, anhand der in der ASG-Schicht repräsentierten aktuellen VISCO-Definition ein Suchprogramm in der Anfragesprache der Basismaschine (räumlichen Datenbank) zu erzeugen. Dies setzt trivialerweise die Kenntnis des

- verwendeten externen Datenmodelles und der
- Anfragesprache

voraus.

Während der Compilerbau für konventionelle textuelle Sprachen gut untersucht und verstanden ist ([ASU88]), ist dies für visuelle Sprachen leider noch nicht der Fall – es gibt weder ein breit einsetzbares formales Rahmenwerk noch eine Standardarchitektur für entsp. „visuelle“ Compiler (s. Kap. 2).

Normalisierer (Inferenzkomponente): Letztlich handelt es sich bei einer VISCO-Definition um ein Constraintnetzwerk (also einen Graphen, zwischen dessen Knoten eine Reihe von Kanten verlaufen, die Einschränkungen repräsentieren): u.a. ist es möglich, *redundante Einschränkungen (Constraints)* anzugeben. Ein Beispiel hierfür wäre ein Zeiger an einer zwischen zwei Nägeln befestigten Teleskopantenne (s. Abb. 6.3(a)). Zunächst ist zu bemerken, daß die Telekopantenne eigentlich ein Holzstab ist (da die Länge durch die beiden fixen Endpunkte eindeutig festgelegt wird). Natürlich ist dann auch ein Zeiger (also eine Einschränkung, die die Orientierung des Segmentes auf genau den eingezeichneten Wert festlegt) redundant. Ein *Normalisierer* würde vielleicht die normalisierte Version in Abb. 6.3(b) erzeugen.

Nun ist aber auch daran zu denken, in welcher *Reihenfolge* die VISCO-Objekte gegen Objekte im Datenbestand abgeglichen werden: wird zuerst nach Strecken gesucht, so kann in der Version aus Abb. 6.3(a) die Einschränkung „Länge=...“ nicht zur Einschränkung des Suchraumes der zu betrachtenden Kandidaten verwendet werden, da sie *nicht explizit vorliegt*, sondern lediglich durch die Eigenschaften der Endpunkte impliziert wird, im Gegensatz zu Abb. 6.3(b), wo die Eigenschaft „Länge=“ inhärent für einen Holzstab ist. Analog verhält es sich mit dem entfernten

Zeiger aus Abb. 6.3(b): hier kann die entfernte Orientierungseinschränkung nicht mehr zur Begrenzung des Suchraumes verwendet werden. Werden hingegen *erst die beiden Nägel instantiiert,* so kann die Strecke natürlich direkt erhalten werden (man denke an die Graphstrukturen bzgl. der „Komponente von"-Relation). Die Überprüfung einer zuvor explizit gemachten „Länge="-Einschränkung (Abb. 6.3(b)) oder einer nicht entfernten „Orientierung="-Einschränkung (Abb. 6.3(a)) würde dann lediglich Zeit kosten und natürlich überflüssig sein, da bereits die „Position=" Einschränkungen für die Nägel überprüft wurden.

In Abhängigkeit von der Suchreihenfolge können sich also sowohl positive als auch negative Effekte ergeben. Es erscheint daher sinnvoll, zunächst eine Normalisierung vorzunehmen und alsdann *möglichst viele implizit vorhandene Einschränkungen durch Inferenz explizit zu machen.* In obigem Beispiel hieße dies, die Einschränkung „Länge=" durch Inferenz zu explizieren und die „Orientierung="-Einschränkung bestehen zu lassen. In *Abhängigkeit von der Suchreihenfolge* könnten dann einige dieser Einschränkungen wieder *ignoriert* werden. Redundante Einschränkungen können also u.U. einen sehr positiven Einfluß auf das Zeitverhalten der Suche haben. Während man in der Mathematik an minimalen Axiomensystemen interessiert ist, ist dies in der KI zugunsten des Zeitverhaltens entsp. Inferenzprozesse nicht immer der Fall (s. [RN95, S. 198]).

Es erscheint zumindest fraglich, ob man die vorgenommenen Normalisierungen bzw. Inferenzen auch dem Benutzer auf der Ebene der Benutzungsoberfläche sichtbar machen sollte, denn die Auswirkungen könnten recht drastisch sein. Schlechtestenfalls wird der Benutzer seine Anfrage nicht wiedererkennen, und die meisten durchgeführten Inferenzen wird er nicht nachvollziehen können (in Expertensystemen ist nicht zuletzt deswegen eine Erklärungskomponente vorzusehen).

Da die Normalisierung und Inferenz keinen Einfluß auf die Semantik einer VISCO-Definition hat, sondern lediglich die Effizienz des vom Compiler erzeugten Suchprogammes positiv beeinflußt, wurden hier keine weiteren Untersuchungen vorgenommen (und somit ist die Inferenzkomponente auch nicht implementiert). Das Explizieren implizit vorhandener Einschränkungen dürfte zudem einen nicht-trivialen (quantitativen) Kalkül erfordern, der hier nicht entwickelt werden konnte.

Ausführung & Ergebnispräsentation: Schließlich soll das vom Compiler erzeugte Suchprogramm *ausgeführt* und die aus dem räumlichen Datenbestand erhaltenen Konstellationen *dargestellt* werden – *das Suchprogramm erzeugt einen Strom von darzustellenden Konstellationen.*

Die einzelnen Konstellationen diese Stromes könnten z.B. als verkleinerte Abbilder (Kacheln) „mosaikartig" nebeneinander gesetzt und auf einzelne Seiten aufgeteilt werden. Einzelne Seiten könnten dann umgeblättert und gelöscht werden. Einzelne Kacheln oder Mengen von *markierten* Kacheln könnten gelöscht werden. Der Benutzer könnte einzelne Kacheln selektieren und genauer (in voller Größe) inspizieren.

Eine *Verifikationsmöglichkeit* sollte vorgesehen werden: so könnten z.B. die Objekte der aktuellen VISCO-Anfrage direkt in die momentan selektierte Konstellation deckungsgleich hineingeblendet werden – der Benutzer könnte so sehen, wohin die Murmeln rollten, wie die Folien rotiert und skaliert wurden, wie die Gummibänder sich anpassten, etc. Eventuell möchte man dann die so der selektierten Konstellation angepassten VISCO-Objekte bzw. die veränderte Anfrage um weitere Objekte verfeinern und erneut stellen.

Da eine Suche recht lange dauern kann, sollte sie vom Benutzer abgebrochen werden können.

Animationskomponente: Die *Animationskomponente* dient dazu, die Vagheiten der aktuellen Anfrage sichtbar zu machen: in einer Art Film könnte der Benutzer verschiedene Beispiele von generierten Konstellationen sehen. Die Bilder des Filmes stellen dabei einzelne Konstellationen aus der Extension der betrachteten VISCO-Definition dar. Dabei sollte nicht nur die Konstellation selbst animiert werden, sondern parallel dazu auch die VISCO-Definition selbst, so daß der Benutzer sehen kann, wie Murmeln ihre Position ändern, Folien rotieren, etc.

Die Implementierung einer solchen Komponente ist eine extrem komplizierte Aufgabe und konnte

im Rahmen dieser Arbeit nicht geleistet werden. Wahrscheinlich würde hier eine kontinuierliche Veränderung von einzelnen Parametern einzelner Objekte (z.B. der Position von Murmeln) und sofortiges Lösen eines Constraintsystemes (Constraint Solving) erforderlich werden (bzw. eine Layoutgenerierung für ein einzelnes Zwischenbild).

Zu diesen technischen Schwierigkeiten kommt erschwerend hinzu, daß eine *aussagekräftige Animation* erzeugt werden soll: alle erlaubten „Extreme“ einer Definition sollten im Film ersichtlich werden, m.a.W., die Extension des definierten räumlichen Konzeptes sollte im Film möglichst flächendeckend dargestellt werden.

6.2 Implementation des Prototypen

Es wurde bereits deutlich, daß die vollständige Implementation der vorgestellten Architektur im Rahmen einer Diplomarbeit nicht geleistet werden kann. Zuviele Fragen sind noch ungeklärt und konnten nicht bearbeitet werden. Durch *Prototyping* konnte jedoch die Realisierbarkeit und Nützlichkeit der bisher entwickelten Ideen bzw. der Sprache VISCO selbst validiert werden. Prototyping wird i. d. R. als Methode der Anforderungsvalidierung im Software-Entwicklungsprozeß gesehen (s. [Som95]).

Aufgrund der exponierten Rolle der Benutzungsoberfläche einer visuellen Sprache ([Gra90]) wurde für ihre Realisierung ungefähr die Hälfte der Gesamtimplementierungszeit investiert. Die andere Hälfte wurde zu ca. 70 Prozent für die Implementierung des Compilers und der ASG-Schicht verwendet.

Der implementierte Prototyp ist zwar in Funktionalität und Performanz momentan noch stark eingeschränkt, beantwortet aber bereits hinreichend komplexe Anfragen innerhalb akzeptabler Wartezeiten (s. Kap. 7). Dazu tragen nicht zuletzt der einfache Optimierer und der (primitive) räumliche Index bei.

Zunächst möchte ich noch bemerken, daß eine teilweise tiefergehende Implementation des Architekturmodelles unter Benutzung bereits existierender kommerzieller Produkte bzw. Komponenten nicht möglich war, da weder ein kommerzielles GIS-System noch eine räumliche Datenbank als Basismaschinen verfügbar waren. Auch hätte (als eine weniger problemadäquate Basismaschine) eine *objektorientierte Datenbank* dienen können ([Heu97]). Lediglich ein LISP-System und das CLIM-Framework sowie die Daten des Hamburger Vermessungsamtes standen zur Verfügung.

Die in der Praxis extrem wichtige *Kopplungsproblematik* zwischen heterogenen Komponenten (verschiedenster Hersteller) wurde somit aus dem Kontext der Arbeit ausgeklammert. In der Praxis muß teilweise erheblicher Aufwand zur Sicherstellung der Interoperabilität betrieben werden, wie es z.B. umfangreiche Datenaustauschsprachen wie „EXPRESS“ (als Teil von „STEP“) oder auch der „SDTS“ (Spatial Data Transfer Standard) eindrucksvoll belegen. Die oben gemachten Bemerkungen zum Thema Datenwörterbuch etc. sind daher nur theoretischer bzw. konzeptueller Natur. Dennoch halte ich es für plausibel, VISCO als Anwendung auf eine räumliche Datenbank aufzusetzen – die Integrationsfähigkeit konnte jedoch nicht experimentell validiert werden.

Der Prototyp wird durch ca. 640 kB LISP-Code implementiert. Über die Sprache Common LISP kann sich der Leser u.a. in [Gra96, Ste90] informieren. Gute Lehrbücher sind [WH93, Gra96, Sla98]. Fortgeschrittene Programmierung wird in [Nor92, Gra94], das Common LISP Object System (CLOS) in [Kee89, KRB91] gelehrt. Bezüglich CLIM gibt es (außer der Spezifikation) leider nur das Handbuch ([Fra94]) – grundlegende CLIM-Konzepte werden jedoch in [Möl95, Möl97] verständlich dargestellt.

6.2.1 Nutzbarmachung eines räumlichen Datenbestandes

Da kein GIS und auch keine räumliche Datenbank zur Verfügung standen, gab es nur die Möglichkeit, selbst einen Datenbestand in eine effektiv nutzbare Form zu bringen. Nachdem vom Hambur-

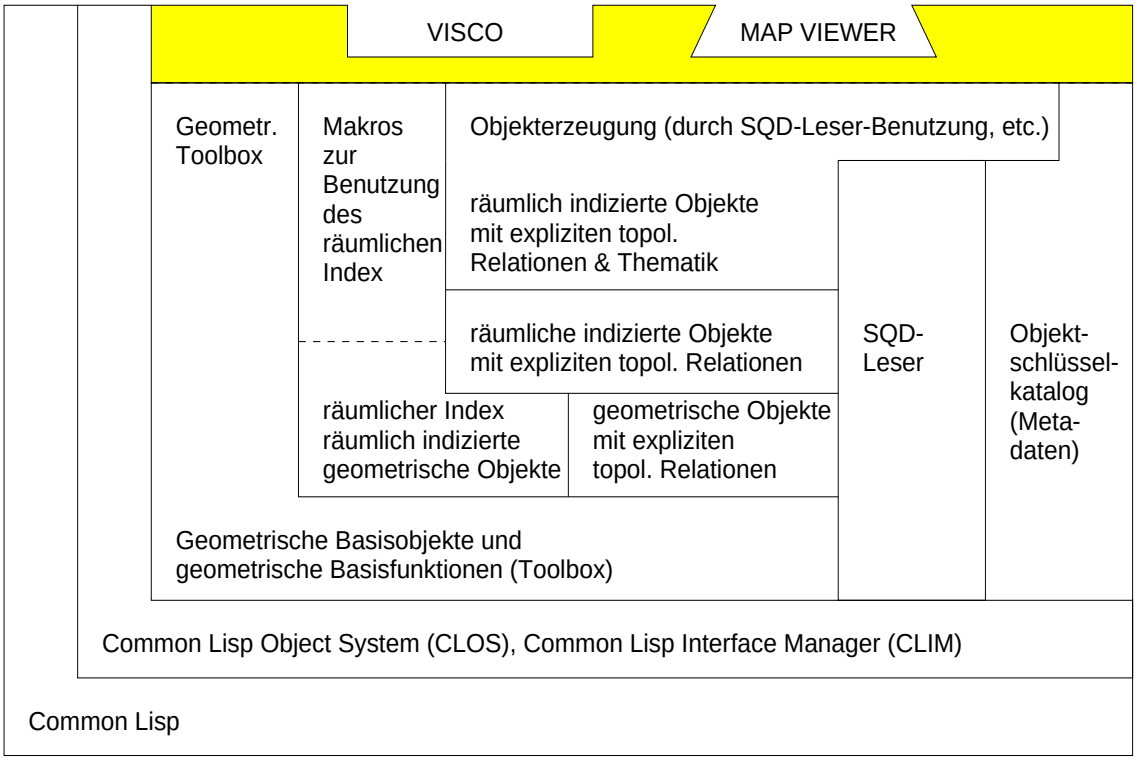


Abbildung 6.4: Verfeinerung Schicht „Räumlicher Datenbestand“

ger Vermessungsamt freundlicherweise ein Ausschnitt der DISK in Form einer SQD-Datei beschafft werden konnte (s. Kap. 3), mußte zunächst ein spezielles Einleseprogramm geschrieben werden. Die SQD-Datei zeigt einen kleinen Ausschnitt des Hamburger Stadtteils Öjendorf (es handelt sich um die im Anhang wiedergegebene Karte „7434 - Öjendorf“). Da es in diese Arbeit primär um räumliche Konstellationen geht, ist es sekundär, daß es sich bei den DISK-Daten lediglich um *grafische* Daten zur Kartendarstellung handelt. Die Daten der DISK haben somit nicht den Status von Katasterdaten, wo es auf exakt vermessene Geometrie ankommt. Dennoch beinhaltet die DISK-Karte interessante räumliche Konzepte, und dies ist der wesentliche Punkt.

Letztlich werden die eingelesenen SQD-Daten als CLOS-Objekte im LISP-Speicher (Image) gehalten – es ist für diese Arbeit nicht wesentlich, daß es sich nicht um eine echte räumliche Datenbank handelt. Natürlich wäre es kein Problem, diese Daten in einer räumlichen Datenbank zu halten und entsp. Schnittstellen wie im obigen Architekturmodell vorzusehen.

Oben wurde die Schicht „Räumlicher Datenbestand“ als Abstraktion einer räumlichen Datenbank bezeichnet: insofern sollte sie lt. Güting ja räumliche Datentypen, räumliche Indizierung (Spatial Indexing) und räumliche Verknüpfung (Spatial Join) unterstützen. Diese Forderungen werden vom Prototypen tatsächlich erfüllt (dennoch möchte ich nicht behaupten, daß es sich deswegen um eine räumliche Datenbank handelt - dies ist sicherlich nicht der Fall). Die implementierte Funktionalität ist jedoch ausreichend, um den Datenbestand effektiv nutzbar zu machen.

Abbildung 6.4 verdeutlicht die Implementation der Schicht „Räumlicher Datenbestand“: insgesamt sind neun Komponenten zur Realisierung der Dienstleistungen dieser Schicht vorgesehen.

LISP & CLOS: Das Fundament wird durch LISP und CLOS gebildet.

Geometrische Basisobjekte und -funktionen: Das Geometriemodul stellt eine Sammlung von als CLOS-Klassen implementierten SDTs zur Verfügung. Zu einem SDT gehören (im Sinne eines abstrakten Datentypen) ja auch stets die entsprechenden Operationen, welche als generische Funktionen bzw. Methoden für diese Klassen implementiert sind. Eine besondere Klasse von Funktionen wird von Prädikaten zur Errechnung interessanter räumlicher Relationen gebildet (hier stellten sich die in CLOS verfügbaren Multimethoden als überaus praktisch heraus). Die implementierten SDTs sind

- Punkte,
- Strecken,
- Ketten und Polygone sowie
- Aggregate beliebiger Punkte, Strecken, Ketten und Polygone.

Diese Objekte werden schrittweise aufgebaut: Strecken aus zwei Endpunkten, Ketten und Polygone aus bereits erzeugten Strecken, etc. Auf diese Art wird der DAG bzgl. der „Komponente von"-Relation aufgebaut. Beim Erzeugen der Objekte werden eine Reihe von Konsistenzbedingungen überprüft:

- Strecken haben eine Länge größer Null.
- Polygone und Ketten sind einfach (s. Kap. 2), Polygone sind zusätzlich geschlossen.
- Je zwei benachbarte Segmente einer Kette oder eines Polygones haben genau eine gemeinsame Punktkomponente, etc.

Für alle mindestens eindimensionalen SDTs werden automatisch die kleinsten umschließenden Rechtecke (Minimum Bounding Rectangles, MBRs, s. Kap. 3) sowie die Mittelpunkte verwaltet (diese werden bei Bedarf errechnet). Das Modul implementiert zudem Transformationen auf diesen SDTs, wie

- Rotationen,
- Skalierungen und
- Translationen.

Ein spezieller Mechanismus sorgt dafür, daß die MBRs lediglich bei Bedarf und wenn notwendig neuberechnet werden: hierzu muß erkannt werden, ob das Objekt seit dem letzten Zugriff auf das MBR transformiert wurde. Man bedenke, daß MBRs immer *achsenparallel* sind und somit natürlich nicht einfach mittransformiert (z.B. rotiert) werden können.

Zu den vom Modul angebotenen Funktionen gehören u.a. *geometrische Grundaufgaben* wie

- Berechnung von Länge und Orientierung einer Strecke (bzw. zweier Punkte).
- Berechnung des Winkels zweier Strecken.
- Berechnung der Fläche eines Polygones.
- Berechnung des Schnittpunktes zweier sich kreuzender Strecken.
- Berechnung der Determinanten.
- Berechnung des kleinsten Abstandes zweier beliebiger SDTs.
- Berechnung der topologischen Relationen „intersects", „disjoint", „inside / contains" und „equal" zwischen beliebigen SDTs.

Viele der hier verwendeten Algorithmen stammen aus [Sed92, SDK96] und konnten aus meiner Studienarbeit ([Wes96]) übernommen werden.

Die Dienste des Geometriemoduls werden direkt an der Schnittstelle der Schicht „Räumlicher Datenbestand“ verfügbar gemacht (s. Abb. 6.1 bzw. Abb. 6.2), da diese an verschiedenen Stellen in oberen Schichten des Systemes benötigt werden.

Geometrische Objekte mit expliziten topologischen Relationen: Dieses Modul stellt einen ersten Schritt in Richtung einer *räumlichen Indizierungsmethode* dar: Weil die Errechnung der oben erwähnten topologischen Relationen „inside“, „intersects“ etc. relativ aufwendig ist, bieten die hier implementierten Klassen die Möglichkeit, diese Relationen explizit zu speichern, so daß eine einmalige Berechnung ausreicht (vorausgesetzt, die Geometrie der Objekte bleibt konstant – dies ist für den hier betrachteten statischen Datenbestand sicherlich der Fall). Die Klassen erweitern die Basisklassen des Geometriemoduls durch Vererbung.

Desweiteren wird hier die *räumliche Selektion (Spatial Selection)* unterstützt. Ohne ein Durchsuchen des Datenbestandes können nun ausgehend von einem Objekt sämtliche Objekte erhalten werden, die in einer der gespeicherten Relationen zu diesem Objekt stehen sollen. Somit wird z.B. die Überprüfung der Relation „contains(A,B)“ trivial, vorausgesetzt, A und B sind beide im räumlichen Datenbestand vorhanden. Das Vorgehen funktioniert natürlich nicht mehr, wenn nur einer der beiden Operanden im Datenbestand vorliegt. Hierfür ist eine weitere Indizierungsmethode vorgesehen, s.u.

Räumlicher Index, räumlich indizierte geometrische Objekte: Einen zweiten Schritt in Richtung räumlich indizierte Objekte stellt das Modul „Räumlicher Index“ dar: hier wird sowohl ein (primitives, aber effizientes) räumliches Indizierungsverfahren in Form eines *regelmäßigen Gitters* (s. Kap. 3) implementiert, als auch eine Anzahl weiterer Klassen (die wiederum die Basis-SDTs des Geometriemoduls per Vererbung erweitern). Die Instanzen der hier implementierten Klassen sind zur *Eintragung in das Gitter* vorgesehen. Das Gitter stellt auch den Basismechanismus zur Erzeugung einer *topologisch strukturierten Datenmenge* dar:

Wird z.B. der Punktkonstruktor mit einem Koordinatenpaar aufgerufen, an dessen Stelle im Gitter bereits ein Punkt existiert, so wird statt eines neuen Punktes der *bereits indizierte Punkt* zurückgegeben. Somit wird automatisch sichergestellt, daß

- es keine kongruenten Punkte gibt, daß
- zwei an den Endpunkten aneinanderstoßende Linien tatsächlich über eine gemeinsame Punktkomponente verknüpft sind, daß
- die Seiten zweier sich berührende Polygone tatsächlich gemeinsame Seiten sind, etc.

Ebenso wird sichergestellt, daß es keine kongruenten Strecken gibt: wird der Konstruktor für die zweite (kongruente) Strecke aufgerufen, so wird die bereits indizierte Strecke zurückgegeben. Analog wird für bereits existente kongruente Ketten, Polygone und Aggregate verfahren.

Das Gitter ist als einfaches zweidimensionales Feld konstanter Auflösung und Größe implementiert. Eine Auflösung von 400 Zellen ist voreingestellt – jede Zelle des Feldes enthält nun einen sogenannten „Bucket“, wobei es sich um eine einfache Liste und ein die Geometrie der Zelle beschreibendes MBR handelt. Die Liste selbst enthält nun jedes Element des räumlichen Datenbestandes, welches die Zelle schneidet (also einen nichtleeren Schnitt mit dem MBR des Buckets hat). Ein Objekt kann somit in mehrere Buckets eingetragen werden.

Verschiedenste Selektionen können nur relativ effizient bearbeitet werden:

- Punktselektionen („Finde einen Punkt mit Koordinaten (x,y)“): Statt alle Objekte des Datenbestandes zu durchsuchen, muß so lediglich der entsp. Bucket ermittelt und alle Elemente

der Liste des Buckets untersucht werden (nimmt man eine konstante mittlere Füllung der Buckets bzw. Gleichverteilung an, so wäre der Geschwindigkeitsgewinn bei 400 Zellen eben jener Faktor, 400).

- Bereichsselektionen („Find alle Objekte, die im Bereich $((x1, y1) - (x2, y2))$ liegen“): Hier muß zunächst das Bereichsrechteck mit allen Buckets auf Überlappung untersucht werden (hierzu müssen 400 Paaren von MBRs auf Überlappung geprüft werden, wobei es sich um eine sehr billige Operation handelt). Die so gewonnene Menge von Buckets stellt die Menge der Kandidaten-Buckets dar: Alsdann wird jedes individuelle Element jedes Buckets auf Enthaltensein im Bereichsrechteck geprüft (hier reicht es aus, ausschließlich die MBRs der einzelnen Objekte zu betrachten). Offensichtlich ist diese Selektion um so effiziente, je kleiner der selektierte Bereich ist (natürlich kann *kein* Index *irgendeinen* Vorteil leisten, wenn der gesamte Bereich selektiert wird).

Auf ähnliche Art und Weise werden auch alle anderen unterstützten räumlichen Selektionen implementiert: stets handelt es sich um einen zweistufigen „Filtere und Verfeinere“-Prozeß (s. Kap. 3):

1. Bestimme die relevanten Kandidaten-Buckets anhand des MBRs eines *Referenzobjektes;* im Falle „enthält(A,B)“ ist A das Referenzobjekt.
2. Jedes der in den als relevant ermittelten Buckets gespeicherte Objekt wird individuell auf Gültigkeit der Selektionsbedingung untersucht. Hier spielen die MBRs der einzelnen Objekte eine wichtige Rolle zur Effizienzerhöhung (u.a. haben sogar die Segmente eines Polygones MBRs, natürlich aber auch das Polygon selbst). In der Regel liefern die MBRs *notwendige Bedingungen:* Bei Nichterfüllung der entsp. Selektionsbedingung zwischen den MBRs kann der (i. d. R.) sehr viel berechnungsteurere Feintest unterbleiben. Ansonsten muß der Feintest vorgenommen werden.

Beispielhaft soll die Selektion „liefere alle Objekte, die in Polygon A enthalten sind“ untersucht werden (also „enthält(A,*)“):

1. Das MBR von Polygon A wird mit allen Buckets auf Überlappung geprüft (Grobtest, Filter).
2. Die Elemente der so ermittelten Buckets werden individuell untersucht: Das MBR jedes Elementes jedes Buckets wird mit dem MBR von Polygon A verglichen. Ist es nicht vollständig im MBR von A enthalten, so kann der Kandidat bereits verworfen werden (notwendige Bedingung verletzt). Ansonsten muß jedoch die „enthält“-Bedingung anhand der Geometrie der beiden Objekte geprüft werden (Feintest).

Folgende Selektionen werden vom implementierten Index unterstützt:

- Liefere alle Objekte, die das Referenzobjekt schneiden,
- Liefere alle Objekte, die im Referenzobjekt enthalten sind,
- Liefere alle Objekte, die das Referenzobjekt enthalten,
- Liefere alle Objekte, deren Entfernung zum Referenzobjekt kleiner als r ist.

Diese Selektionen sind durch eine kleine Makrosprache an der Schnittstelle der Schicht „Räumlicher Datenbestand“ direkt aufrufbar. So würde z.B. das Konstrukt

```
(with-selected-objects (retrieved-object reference-object :contains)
  (princ retrieved-object)
  (terpri))
```

alle Objekte ausdrucken, die im Objekt `reference-object` enthalten sind. Während obiges Konstrukt sowohl Grob- als auch Feintest durchführt, würden mit

```
(with-selected-buckets (relevant-bucket reference-object :contains)
  (princ relevant-bucket)
  (terpri))
```

nur die relevanten Buckets ausgedruckt, also nur der Grobtest durchgeführt. Für eine „$\epsilon(r)$"-Selektion muß zusätzlich der Radius (in Metern) angegeben werden:

```
(with-selected-objects (retrieved-object reference-object :epsilon 100)
  (princ retrieved-object)
  (terpri))
```

Einige offensichtliche Punkte sind folgende:

- Die „Schneidet"-Selektion ist umso effizienter, je kleiner das Referenzobjekt ist.
- Die „Enthält"-Selektion ist umso effizienter, je kleiner das Referenzobjekt ist.
- Die „Enthalten in"-Selektion ist umso effizienter, je größer das Referenzobjekt ist.
- Die „$\epsilon(r)$"-Selektion ist umso effizienter, je kleiner r und das Referenzobjekt sind.

Derartige Regeln könnte z.B. der Optimierer bei der Auswahl der Selektionsreihenfolge im Rahmen der Plangenerierung berücksichtigen (s. dort). Natürlich spielt im Rahmen des Feintestes auch die Komplexität der Objekte eine Rolle (also die Anzahl der Komponentenobjekte).

Das Gitter ist nur dann sinnvoll einsetzbar, wenn eine ungefähre Gleichverteilung der räumlichen Objekte vorliegt. Dies ist im verwendeten Datenbestand näherungsweise der Fall (die Randbuckets sind etwas dünner besetzt).

Räumlich indizierte geometrische Objekte mit expliziten topologischen Relationen: Die hier implementierten Klassen werden per Mehrfachvererbung gebildet: so ist ein „räumlich indizierter Punkt mit expliziten Relationen" sowohl ein „Punkt mit expliziten topologischen Relationen" als auch ein „räumlich indizierter Punkt".

Zur Berechnung der expliziten Relationen kann jetzt bereits der räumliche Index benutzt werden: sobald ein neues Objekt eingeführt wird, kann der Index z.B. zur Bestimmung aller (bereits indizierter) Objekte verwendet werden, die im neuen Objekt enthalten sind, etc. Zudem werden von diesem Modul die Schnittpunkte von sich kreuzenden Strecken automatisch explizit gemacht, indiziert und ebenso Relationen für diese errechnet und gespeichert.

SQD-Konvertierer: Der SQD-Konvertierer liest eine SQD-Datei (hierbei handelt es sich um das vom Hamburger Vermessungsamt gelieferte Dateiformat des SiCAD-, Siemens-CAD-Systemes) und erzeugt zunächst eine interne Repräsentation dieser Objekte. Für jedes eingelesene SQD-Objekt wird alsdann eine Konvertierungsfunktion aufgerufen – diese Routinen müssen vom Klienten des SQD-Konvertierers implementiert werden. Da der SQD-Konvertierer als unabhängiges Programm entworfen wurde und der Konvertierer somit keine Annahmen über das benutzende Modul machen kann, ist es erforderlich, Kommunikation auf diese Weise zu implementieren (vgl. das „Erbauer"-Entwurfsmuster in [GHJV96, S. 103]).

So wird z.B. für einen SQD-Punkt die Konvertierungsfunktion `transform-sqd-pg` mit den entsp. Punktparametern aufgerufen. Das benutzende Modul (Klient) implementiert diese Funktion und erzeugt somit eine eigene Repräsentation dieser Objekte.

Objektschlüsselkatalog: Jedem SQD-Datum ist ein vierstelliger thematischer Identifizierer zugeordnet (Objektschlüssel, s. Kap. 3). So wird z.B. eine Parkfläche durch den Objektschlüssel „2224“ identifiziert. Zusätzlich zum verwendeten Objektschlüssel werden auch die möglichen räumlichen Datentypen der Objekte aufgelistet – so hat z.B. ein Objekt mit dem Schlüssel „2224“ stets den räumlichen Datentyp „Fläche“ bzw. Polygon. Der Objektschlüsselkatalog ist somit das Äquivalent zum Datenwörterbuch, da hier die Metadaten katalogisiert werden.

Der Objektschlüsselkatalog implementiert durch eine Hashtabelle eine Funktion, die anhand des Objektschlüssels eine natürlichsprachliche Beschreibung liefert (z.B. $f(2224) =$„Park“). Zusätzlich wird hier eine zweite Abbildung implementiert: Oftmals ist es wünschenswert, *spezielle Konvertierungen* während des Einlesens von SQD-Objekten in Abhängigkeit von deren Thematik vorzunehmen. Daher liefert die zweite hier implementierte Funktion anhand des Objektschlüssels eines eingelesenen SQD-Objektes ein Funktionsobjekt, welches auf das eingelesene SQD-Datum angewendet wird und weitere Konvertierungen vornimmt.

Thematische räumlich indizierte geometrische Objekte mit expliziten topologischen Relationen: Dieses Modul bildet die oberste Ebene der Schicht „Räumlicher Datenbestand“: zunächst werden die Klassen des Moduls „Räumlich indizierte SDTs mit expliziten topologischen Relationen“ (wieder durch Vererbung) erweitert. Die Klassen werden u.a. um einen Slot erweitert, der zur Aufnahme von Objektschlüsseln dient. Da in einer SQD-Datei u.U. kongruente Objekte mit verschiedenen Objektschlüsseln vorkommen können, handelt es sich um ein *listenwertigen* Slot: wie bereits diskutiert, werden kongruente Objekte in einem Objekt zusammengefaßt (topologisch strukturiert); dieses Objekt bekommt dann alle Themen der so zusammengefassten (kongruenten) Objekte. Anhand der Reihenfolge der Objektschlüssel kann die Thematik bzgl. des Elternobjektes wieder eindeutig ermittelt werden: Enthält z.B. eine SQD-Datei ein Segment einer Parkfläche, das die Thematik „Parkgrenze“ trägt und ein weiteres kongruentes Segment eines Teiches, welches die Thematik „Teichbegrenzung“ trägt, so werden diese beiden Segmente in einem Objekt mit beiden Bedeutungen zusammengefaßt. Die Reihenfolge der Objektschlüssel dieses Objektes bezieht sich dann auf die Reihenfolge der Elternobjekte im Slot „Komponente von“: bzgl. des ersten Elternobjektes hat das Segment die Thematik „Parkgrenze“, bzgl. des zweiten Elternobjektes jedoch die Thematik „Teichbegrenzung“.

Schließlich wird auch eine Klasse definiert, deren Instanzen ganze räumliche Datenbestände repräsentieren: eine solche Instanz verfügt u.a. über einen eigenen räumlichen Index. Verschiedene räumliche Datenbestände können gleichzeitig im LISP-Image gehalten werden; jeder wird in der Regel zu einer SQD-Datei korrespondieren. Die Schnittstelle sieht eine Funktionen vor, mit deren Hilfe eine SQD-Datei in einen Datenbestand konvertiert werden kann. Die räumlichen Objekte werden dann automatisch konvertiert, in den Index eingefügt, verschiedene Relationen explizit gemacht, etc. Das Einlesen und Konvertieren einer SQD-Datei dauert einige Zeit (ca. 15 Min. für die vorhandene SQD-Datei – hier müssen ja einige 10.000 topologische Relationen errechnet und der Index aufgebaut werden), so daß es sinnvoll ist, den Datenbestand in der konvertierten Form (als Datei) abzuspeichern. Später kann er dann recht schnell wieder eingelagert werden.

Nun hat man einen räumlich indizierten, topologisch strukturierten und mit expliziten Relationen versehenen räumlichen Datenbestand vorliegen, der von VISCO effizient genutzt werden kann.

Die diskutierte Implementation der Schicht „räumlicher Datenbestand“ erfordert insgesamt 163 kB LISP-Code.

6.2.2 Die Inspektionskomponente Map Viewer

Eine Inspektionskomponente wurde ebenfalls implementiert – hierbei handelt es sich um den sog. Map Viewer. Dieser ist als spezielle Anwendung innerhalb der Anwendungsschicht vorgesehen und liest die Daten des räumlichen Datenbestandes zwecks grafischer Darstellung, wozu eine Karten-

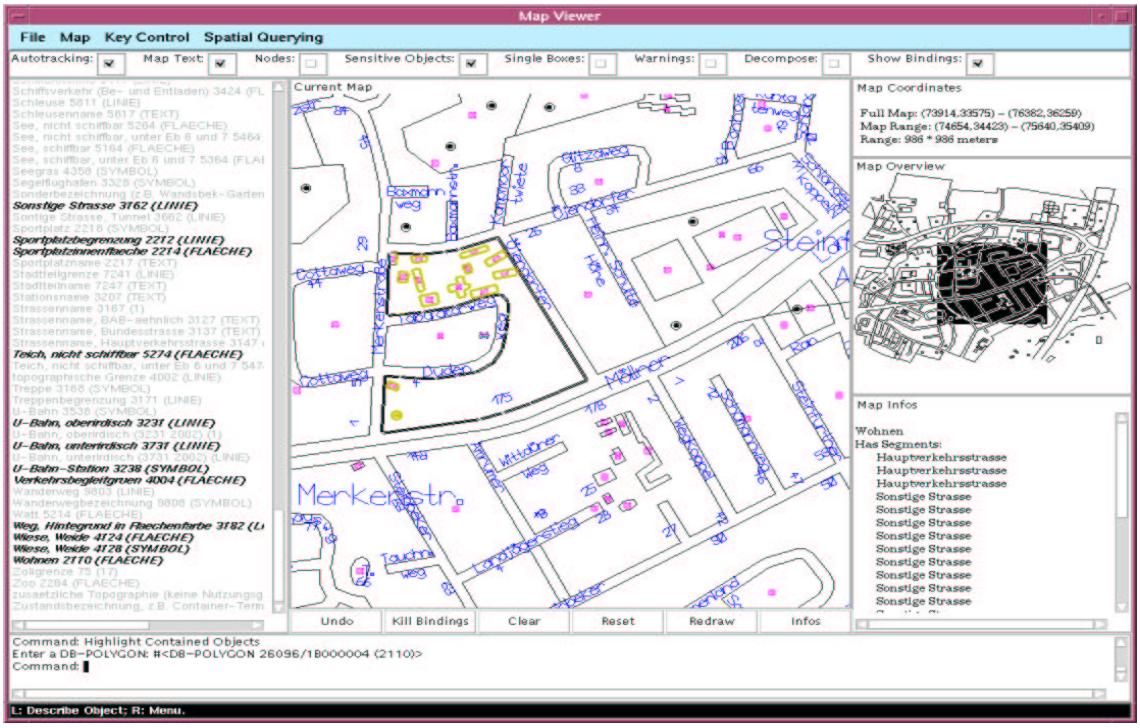


Abbildung 6.5: Inspektion des Datenbestandes mit dem Map Viewer

Metapher verwendet wird.[1]

In der Anfangsphase dieser Arbeit war der Map Viewer sehr wichtig, da nur mit seiner Hilfe eine Vorstellung über Art und Umfang des vom Hamburger Vermessungsamt zur Verfügung gestellten Datenmaterials ermittelt werden konnte. Während viele Standarddateiformate ja von best. Grafikprogrammen eingelesen werden können (z.B. DXF von AutoCAD), so ist dies für SQD-Dateien nicht der Fall, so daß der Map Viewer tatsächlich die einzige Inspektionsmöglichkeit darstellt.

Abb. 6.5 zeigt die mit Hilfe von CLIM implementierte Oberfäche des Map Viewer, wobei verschiedene Teilfenster zu erkennen sind:

- Links befindet sich der *Objektschlüsselselektor*. Hierbei handelt es sich um den Objektschlüsselkatalog des Hamburger Vermessungsamtes ([Fre96b]). Die in der aktuellen Karte vorhandenen Objektschlüssel können hierin selektiert und deselektiert werden, wodurch die entsprechenden Objekte in der Kartendarstellung dann ein- bzw. ausgeblendet werden. In der aktuellen Karte nicht vorhandene Objektschlüssel werden in „Geisterschrift“ dargestellt. Da sich jeder Objektschlüssel ein- und ausschalten läßt, kann man eine Anzahl von 2^n *Layern* herstellen (s. Kap. 3).
- In der Mitte wird die *aktuelle Karte* dargestellt: der Benutzer kann einzelne Objekte per Mausgeste inspizieren oder eine Beschreibung der Thematik im *Infofenster* (rechts unten) erhalten. Zudem kann die Graphstruktur der Objekte inspiziert werden (s. Abb. 6.6).
- Rechts oben wird der *aktuelle Ausschnitt* der aktuellen Karte in Gauss-Krüger Koordinaten (und Quadratmetern) angegeben.
- Rechts mittig wird der aktuelle Ausschnitt der aktuellen Karte als schwarzes Quadrat in der *Übersichtskarte* dargestellt: das Zentrum des Quadrates kann interaktiv verschoben und der Radius des Quadrates (bzw. des eingeschriebenen Kreises) angepaßt werden.

[1] Tatsächlich ist der Map Viewer die Benutzungsschnittstelle zur Schicht „Räumlicher Datenbestand“, denn mit seiner Hilfe wird ein Datenbestand geladen, die Konvertierung einer SQD-Datei angestoßen, etc.

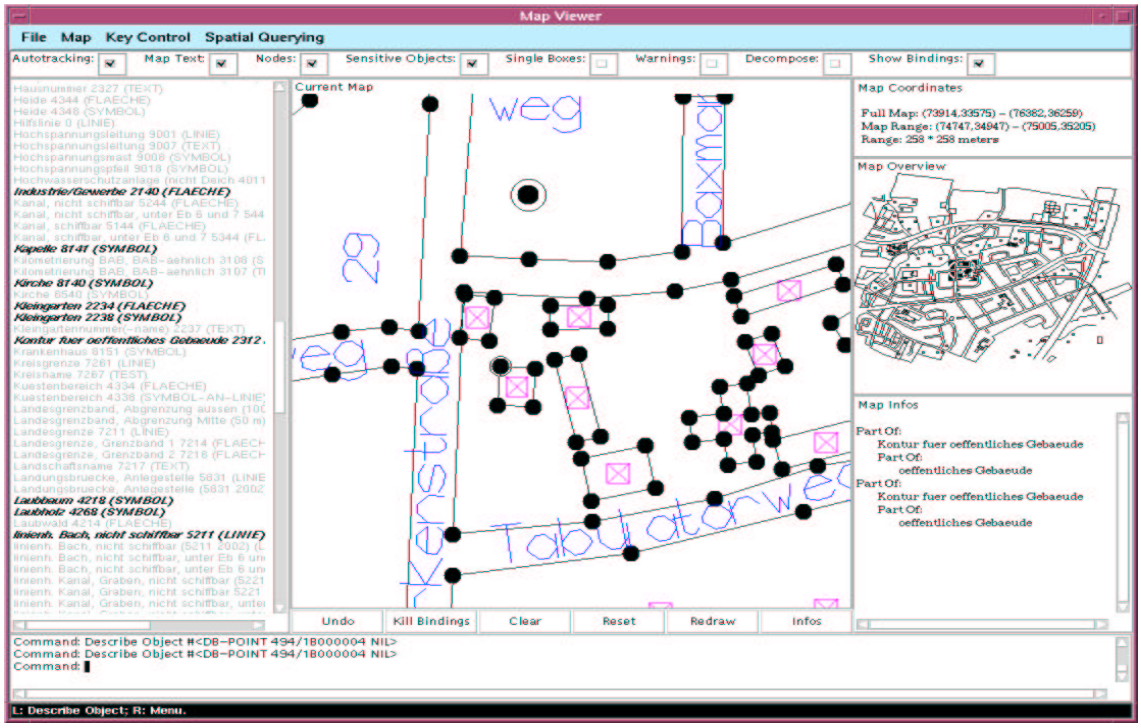


Abbildung 6.6: Inspektion der Graphenstruktur der DISK-Objekte

- Rechts unten befindet sich das *Infofenster:* hier werden textuelle Beschreibungen einzelner mit der Maus selektierter Objekte ausgegeben.
- Unten ist die von CLIM automatisch verwaltete *Kommandozeile* angebracht: hier können textuelle Kommandos eingetippt werden.
- Eine Reihe von *Schaltern (Check Boxes)* und *Tastern (Push Buttons)* steuern verschiedene grafische Darstellungsoptionen.

Für die grafische Darstellung der Karte kann bereits der räumliche Index gewinnbringend genutzt werden: der aktuelle Kartenausschnitt ist natürlich eine räumliche *Bereichsselektion (Range Selection)* in Form eines Quadrates (und mit der Bedingung „not outside“) auf dem aktuellen Datenbestand. Obwohl das „Clipping“ an den Fenstergrenzen natürlich vom Windowmanager geregelt wird und somit nicht im aktuellen Ausschnitt liegende Objekte auch nicht dargestellt werden, ist es dennoch viel effizienter, die zu zeichnenden Objekte durch die Bereichsselektion bestimmen zu lassen.

Folgende zusätzliche Funktionalität wird vom Map Viewer geboten:

- Ausblenden eines einzelnen, individuell in der Karte selektierten Objektes.
- Deselektion aller Objektschlüssel eines individuell in der Karte selektierten Objektes.
- Ausschließliche Selektion aller Objektschlüssel eines individuell in der Karte selektierten Objektes.
- Selektion und Deselektion aller Objektschlüssel.
- Drucken der aktuellen Karte.

- Inspektion der explizit gespeicherten räumlichen Relationen: hierzu wird z.B. der Menüpunkt „Show Contained Objects“ ausgewählt und dann ein Polygon in der Karte selektiert. Der Map Viewer wird dann alle im selektierten Objekt enthaltenen Objekte hervorheben.

- Einblenden der Knoten der Graphstrukturen (Punkte) – durch die Selektion von Komponentenobjekten kann die Graphstruktur („Komponente von“-Beziehung der DAGs) im Infofenster inspiziert werden.

Der Map Viewer konnte mit nur 46 kB implementiert werden. In der Kürze des Programmes zeigt sich die Leistungsfähigkeit von LISP, CLOS und CLIM als mächtige Prototyping-Umgebung. Dennoch funktioniert nicht immer alles so, wie man es sich wünschen würde: so mußten z.B. die skalier- und rotierbaren Beschriftungen selbst implementiert werden, da z. Zt. weder die X-Windows-Basis noch CLIM Textobjekte skalieren- und/oder rotieren können. So wurde für jeden Buchstabe mit einem Grafikeditor eine Vektordarstellung gezeichnet, die dann in eine CLIM-Zeichenroutine compiliert werden konnte. Die Kartendarstellungen könnten noch verbessert werden, wenn die bisher nicht genutzten Grafikattribute der SQD-Objekte entsp. berücksichtigt würden (Farbe, Schraffur, etc.).

6.2.3 Implementation von VISCO

Im folgenden wird ausführlich die Implementation von VISCO dargestellt, wobei besonderes Gewicht auf die Vorstellung des optimierenden Compilers und des Grafikeditors gelegt wird. Vorausgeschickt sei, daß die erwähnte Animationskomponente und auch der Normalisierer aufgrund offensichtlicher Schwierigkeiten nicht implementiert werden konnten.

Einen Eindruck von der bisher implementierten Funktionalität des Prototypen bekommt der Leser im folgenden Kapitel, wo die Verwendung der Prototypen anhand eines Beispieles dargestellt wird.

ASG

Alle fünf Komponenten müssen auf dem von der ASG-Schicht implementierten *Repository* arbeiten: hierbei handelt es sich um die abstrakte syntaktische Repräsentation der aktuellen VISCO-Definition. Das gesamte ASG-Modul konnte durch 64 kB LISP-Code implementiert werden: Die Kürze wird vor allem durch eine kompakte Operatorschreibweise erreicht (s.u.).

Definition (Abstrakte Syntax): Die *abstrakte Syntax* einer VISCO-Definition ist durch einen gerichteten Multi-Hypergraphen G gegeben, $G = (\mathsf{E}, \mathsf{K})$, wobei E die Menge der Ecken oder Knoten und $\mathsf{K} \subseteq \bigcup_{i \in \mathbb{N}, i \geq 1} \mathsf{Kantentypen} \times \mathsf{E}^i$ die Menge der gerichteten Hyperkanten ist. Die *Richtung* einer $kantentyp_n$-Hyperkante, $n \geq 2$ $(kantentyp, a_1, a_2, \ldots, a_n)$ geht von $a_2 \ldots a_n$ nach a_1. Eine Hyperkante mit $n = 1$ heißt *Eigenschaft,* eine Hyperkante mit $n = 2$ heißt *Kante.* Auf den Hyperkanten sind eine Reihe von Funktionen definiert: es handelt sich somit um attributierte Hyperkanten. So werden etwaige zur Beschreibung erforderliche Parameter repräsentiert (z.B. benötigt man ein Attribut bzw. eine Funktion wie $position^\star$ auf der Menge der Eigenschaften $hat\_position$, s.u.).

Die Menge Kantentypen ist ein Menge von Symbolen,

$$\begin{aligned}\mathsf{Kantentypen} = \{ & \mathit{enthält}, \mathit{enthalten\_in}, \mathit{schneidet}, \mathit{schneidet\_nicht},\\ & \mathit{komponente\_von}, \mathit{hat\_komponente},\\ & \mathit{mittelpunkt\_von}, \mathit{hat\_mittelpunkt},\\ & \mathit{erlaubte\_winkelabweichung\_höchstens},\\ & \vdots\\ & \mathit{ist\_schnittpunkt\_von},\\ & \vdots\\ & \mathit{hat\_folien\_breite}, \mathit{hat\_folien\_höhe},\\ & \mathit{hat\_folien\_sx\_min}, \mathit{hat\_folien\_sx\_max},\\ & \mathit{hat\_folien\_sy\_min}, \mathit{hat\_folien\_sy\_max},\\ & \mathit{hat\_position}, \mathit{hat\_orientierung}, \mathit{hat\_länge},\\ & \mathit{hat\_maximum}, \mathit{hat\_status}, \mathit{hat\_semantik}\}.\end{aligned}$$

Die Funktion *inverse* : Kantentypen → Kantentypen gibt das Symbol zurück, welches die inverse Kante bezeichnet ($\mathit{inverse}(\mathit{enthält}) = \mathit{enthalten\_in}$, etc.).

Eine spezielle Teilmenge ist die Menge der *Eigenschaften,* die mit „hat“ beginnen. Während es auf den ersten Blick sinnvoller erscheint, diese Eigenschaften als spezielle Funktionen bzw. „Methoden“ auf den entsp. Knotenmengen zu definieren, wird sich die einheitliche Behandlung von Kanten, Hyperkanten und Eigenschaften bei der Beschreibung des Compiler noch als nützlich herausstellen. Hierdurch sei die Reifikation der Eigenschaften und Relationen gerechtfertigt – der Compiler muß diese als *Daten* behandeln können.

Hyperkanten werden zur *Repräsentation mehrstelliger Operatoren* verwendet. Ein Beispiel ist die Kante $\mathit{ist\_schnittpunkt\_von}(p, s_1, s_2)$: so wird die Abhängigkeit repräsentiert, daß Punkt p der Schnittpunkt von s_1 und s_2.

Die Menge der Knoten E wird vollständig partitioniert durch die Menge der Punkte, Segmente, Polygone und Ketten, Folien und Gebiete: $\mathsf{E} = \mathsf{P} \uplus \mathsf{S} \uplus \mathsf{X} \uplus \mathsf{F} \uplus \mathsf{G}$.

Weiterhin gilt:

- $\mathsf{P} = \mathsf{Nagel} \uplus \mathsf{Murmel} \uplus \mathsf{Ursprung}$
- $\mathsf{S} = \mathsf{Atomares\_Segment} \uplus \mathsf{Gummiband}$
- $\mathsf{Atomares\_Segment} =$
  $\mathsf{Holzstab} \uplus \mathsf{Antenne} \uplus \leq \mathsf{\_Antenne} \uplus \geq \mathsf{\_Antenne}$
- $\mathsf{X} = \mathsf{Kette} \uplus \mathsf{Polygon}$
- $\mathsf{G} = \mathsf{Inneres\_Gebiet} \uplus \mathsf{Äußeres\_Gebiet} \uplus \mathsf{Epsilon\_Gebiet}$
- $\mathsf{Epsilon\_Gebiet} = \epsilon\mathsf{\_Gebiet} \uplus \epsilon^{\oplus}\mathsf{\_Gebiet} \uplus \epsilon^{\ominus}\mathsf{\_Gebiet}$

Im Anhang wird eine formale Sprachdefinition für VISCO vorgestellt: u.a. beinhalten die dort angegebenen Axiome umfangreiche syntaktische Konsistenzbedingungen. Eine VISCO-Definition wird dort als Modell einer Menge von Axiomen definiert. Nun wird dieses Modell als Graph G interpretiert: dabei werden die einzelnen Individuen des Grundbereiches des Modelles (für die *visco_object* gilt) als die Knoten V und die einzelnen Relationstupel und Attribute zwischen diesen als Kanten E des Graphen G betrachtet. So gilt z.B.

P = visco_point
Nagel = visco_nail
Murmel = visco_marble
Ursprung = visco_origin
S = visco_line
Atomares_Segment = visco_atomic_segment
Holzstab = visco_stick
Antenne = visco_antenna
$\leq$ _Antenne = visco_ $\leq$ _antenna
$\geq$ _Antenne = visco_ $\geq$ _antenna
Gummiband = visco_rubberand
etc.
$\vdots$

Auch die Kanten K lassen sich direkt überführen (*dpo* steht für „direct part of“, direkter Bestandteil von):

$$\begin{aligned}
&\forall a, b \ \in \mathsf{E} \\
&\quad (((komponente_von, a, b), (hat_komponente, b, a) \in \mathsf{K} \Leftarrow dpo(a, b)) \wedge \\
&\quad ((enthält, a, b), (enthalten_in, b, a) \in \mathsf{K} \Leftarrow visible_contains(a, b)) \wedge \\
&\quad ((schneidet, a, b), (schneidet, b, a) \in \mathsf{K} \Leftarrow visible_intersects(a, b)) \wedge \\
&\quad ((schneidet_nicht, a, b), (schneidet_nicht, b, a) \in \mathsf{K} \Leftarrow visible_disjoint(a, b)) \wedge \\
&\quad ((mittelpunkt_von, a, b), (hat_mittelpunkt, b, a) \in \mathsf{K} \Leftarrow b \in explicit_centroid_of(a)) \wedge \\
&\quad ((erlaubte_winkelabweichung_höchstens, a, b), \\
&\quad\ (erlaubte_winkelabweichung_höchstens, b, a) \in \mathsf{K} \Leftarrow \\
&\quad\quad \exists\ d\ (relative_orientation_constraint(a, b, d))) \wedge \\
&\quad (\forall p\ ((ist_schnittpunkt_von, p, a, b) \in \mathsf{K} \Leftarrow intersection_point_of(p, a, b))))
\end{aligned}$$

Einige Attribute bzw. Eigenschaften sind:

$$\begin{aligned}
&\forall p \in \text{visco_nail}\ ((hat_position, p) \in \mathsf{K}) \\
&\forall s \in \text{visco_atomic_segment}\ ((hat_länge, s) \in \mathsf{K}) \\
&\forall s \in \text{visco_atomic_segment}\ (orientation_constraint(s) \Rightarrow (hat_orientierung, s) \in \mathsf{K}) \\
&\forall o \in \text{visco_non_enclosure_object}\ ((hat_status, o) \in \mathsf{K}) \\
&\forall o \in \text{visco_db_object}\ ((hat_semantik, o) \in \mathsf{K}) \\
&\vdots
\end{aligned}$$

Zusätzlich ist es notwendig, eine Reihe von Funktionen auf der Menge der Kanten K zu definieren: auf diese Weise können eventuell notwendige Parameter von Beschränkungen als Funktionswerte dieser auf den Hyperkanten definierten Funktionen zurückgegeben werden (attributierte Kanten).

So wird z.B. auf der Menge der Eigenschaften

$$\{e \mid e \in \mathsf{K} \wedge \exists\ p \in \mathsf{P}\ (e = (hat\_position, p))\}$$

die Funktion $position^\star : \mathsf{K} \to \mathbb{R}^2$ definiert. Hier gilt:

$$\begin{aligned}
&\forall p\ (hat_position, p) \in \mathsf{K} \Rightarrow \\
&\qquad position^\star((hat_position, p)) = (x(p), y(p))
\end{aligned}$$

Im Anhang wird die Position eines Punktes durch die beiden Attribute bzw. Funktionen x und y repräsentiert. Analog gilt:

$länge^\star : \mathsf{K} \to \mathbb{R}$, so daß

$$\begin{aligned}
&\forall s\ (hat_länge, s) \in \mathsf{K} \Rightarrow \\
&\qquad länge^\star((hat_länge, s)) = length(s)
\end{aligned}$$

Im folgenden wird die $status^\star$-Funktion noch von Bedeutung sein:

$status^\star : \mathsf{K} \to \{DB, DB\_C, U\}$, so daß

$$
\begin{array}{l}
\forall o\ (hat\_status, o) \in \mathsf{K} \Rightarrow \\
\quad status^{\star}((hat\_status, o)) = \\
\quad\quad \textbf{if}\ db\_object(o) \\
\quad\quad\quad (\ \textbf{if} must\_match\_primary\_db\_object(o)\ DB\ \textbf{else}\ DB\_C) \\
\quad\quad \textbf{else} \\
\quad\quad\quad U
\end{array}
$$

Im Kap. 5 wurden U-Objekte als Hilfsobjekte bezeichnet (U steht für *Universum*), DB-Objekte als $D^{\star}$- und DB_C-Objekte als D-Objekte (der Prototyp verwendet – aus „historischen Gründen“ – die Kürzel U, DB und DB_C).

Während die formale Sprachdefinition auch die Semantik einer VISCO-Definition zum Ausdruck bringt (s. Anhang), sollen hier nur die syntaktischen Bedingungen betrachtet werden.

Die Betrachtung einer VISCO-Anfrage als wohlgeformten Graphen wird im weiteren Verlauf noch einige Vorteile bringen (u.a. bei der Beschreibung des Compilers), wodurch die hier vorgenommenen Transformationen gerechtfertigt sind. Jeder Knoten- und Kantentyp wird im ASG-Modul durch eine eigene CLOS-Klasse implementiert; die attributierten Kanten bzw. Funktionen für diese sind dann natürlich entsp. Methoden.

Eine Reihe von vom ASG-Modul angebotenen Operatoren wird nun verwendet, um eine VISCO-Definition im Repository zu konstruieren – die Operatoren werden durch den Grafikeditor angewendet und können nur so verwendet werden, daß stets syntaktisch korrekte VISCO-Definitionen, also wohlgeformte Graphen im ASG-Modul entstehen. Wird ein neues Objekt erzeugt, so werden u.a. automatisch die räumlichen Relationen zu bereits existierenden Objekten berechnet. Dies setzt natürlich die Kenntnis der Geometrie der einzelnen Elemente voraus. Die konkrete Syntax (also die Visualisierung der abstrakten Syntax) wird jedoch vom Grafikeditor bestimmt, denn er stellt die Oberfläche bzw. *die visuelle Sprache an sich* dar. Hier wird deutlich, daß eine enge Kopplung zwischen ASG-Modul und Grafikeditor vorliegen muß – tatsächlich muß also ein Teil der konkreten Syntax (nämlich die Geometrie) auch Teil der abstrakten Syntax sein. Dennoch spielt es für die abstrakte Syntax keine Rolle, wie Gummibänder visuell von Holzstäben unterschieden werden können. Es ist klar, daß die verwendeten Visualisierungen unterschiedlich sein müssen. Werden die verwendeten Visualisierungen jedoch komplexer, so muß achtgegeben werden, daß hierdurch keine zusätzlichen räumlichen Relationen in der konkreten Syntax auftauchen, die in der abstrakten Syntax keine Bedeutung haben – so hat z.B. die Visualierung einer Murmel (die konkrete Syntax ist ein kleiner Kreis) andere räumliche Eigenschaften als ein punktförmiges Objekt (abstrakte Syntax einer Murmel). Hierdurch eventuell entstehende Inkonsistenzen zwischen konkreter und abstrakter Syntax können jedoch vermieden werden, wenn der Benutzer entsprechend eingeschränkt wird (z.B. durch Gitter).

Das Ableiten von z.B. räumlichen Einschränkungen beim Erzeugen neuer Objekte anhand der Geometrie läßt sich als schrittweiser Inferenzprozeß deuten ([Lin95]), ähnlich einem Kalkül, der aus Grundtermen mit Hilfe deduktiver Axiome neue Theoreme ableitet. Es wird deutlich, daß hierbei das Problem der *Nichtmonotonie* auftritt: im Rahmen eines interaktiven Konstruierens mit dem Grafikeditor ist es notwendig, Objekte nicht nur zu erzeugen, sondern auch wieder zu löschen oder z.B. zu verschieben. Natürlich müssen dann eventuell deduzierte Informationen wieder zurückgenommen werden, andere kommen vielleicht hinzu. Dies setzt in der Regel ein kompliziertes Abhängigkeitsverwaltungssystem (Truth Maintenance System, TMS) voraus. Man kann also *zwei Arten von Operatoren* unterscheiden:

- *Kritische* oder *nichtmonotone Operatoren* führen dazu, daß bereits gemachte Ableitungen zurückgenommen werden müssen, während dies bei
- *unkritischen Operatoren* nicht der Fall ist.

Eine Möglichkeit zur Behandlung der Nichtmonotonie wäre, die *Konstruktionssequenz von Operatoren zu memorieren* und den Graphen anhand dieser nach einem kritischen Operator *komplett neu zu rekonstruieren* (hier würden also alle bisher gemachten Inferenzen verworfen und vollständig

neu gemacht). Nichtmonotone Operatoren *verändern zuvor die Konstruktionshistorie* – soll z.B. ein Objekt gelöscht werden, so wird aus der Historie der entsp. Konstruktionsschritt gelöscht. Alsdann wird das Repository gelöscht und versucht, den Graphen anhand der nun veränderten Historie komplett neu zu rekonstruieren. Schlägt dies fehl, so darf die nichtmonotone Operation nicht zugelassen werden und der alte Zustand muß widerhergestellt werden.

Die unkritischen Operatoren lassen sich hingegen leicht behandeln: sie erzeugen einfach neue Einträge in der Konstruktionshistorie. Soll z.B. ein neues Objekt erzeugt werden, so müssen hier keine Inferenzen zurückgenommen werden (es gibt keine „Default-Schlüsse“ o.ä., die die Situation verkomplizieren könnten; bei „Default-Schlüssen“ können sog. Standardannahmen [Defaults] anhand konkreterer neuer Informationen invalidiert werden). Daher wird der neu in die Historie aufgenommene Operator einfach ausgeführt, ohne daß eine komplette Rekonstruktion notwendig wird.

Da alle nichtmonotonen Operatoren auf oben beschriebene Art (durch Manipulation der aktuellen Konstruktionshistorie) implementiert werden können (hierzu gehören in erster Linie das Löschen und Verschieben von Objekten), sind diese Operatoren tatsächlich nicht im ASG-Modul vorhanden. Stattdessen wird die Konstruktionshistorie vom Grafikeditor verwaltet.

Nun soll noch die verwendete Makrosprache zur Definition von Operatoren vorgestellt werden:

```
(defoperator create-marble ((transparency transparency) (status symbol)
                            (x number) (y number) operator-result
                            &rest initargs)
  (:stored-operator nil)
  (:precondition ((declare (ignore initargs))
                  (and (check-point x y status 'marble
                                    operator-result transparency)
                       (inside-any-enclosure-p* x y (query transparency)))))
  (:code ((apply #'make-visco-marble transparency status x y initargs))))

(defun check-point (x y status type operator-result transparency)
  (and (not (get-present-point-at (query transparency) x y))
       (status-of-point-object-ok-p status type operator-result)
       (inside-p* x y transparency)
       (every #'(lambda (obj)
                  (=> (typep obj 'point)
                      (> (distance-between* x y (x obj) (y obj))
                         +intersects-threshold+)))
              (visco-objects (query transparency)))))
```

Hier wird also der Operator `create-marble` definiert: das Makro definiert lediglich eine Reihe von CLOS-Methoden und Klassen. So wird die `:code`-Sequenz des Operators mittels des Methodenaufrufs `(apply-create-marble ...)` ausgeführt. Die Anwendbarkeit des Operators wird intern zuvor automatisch geprüft (`:precondition`); sie läßt sich auch unabhängig hiervon mittels `(create-marble-applicable ...)` prüfen. Ist der Operator nicht anwendbar, so wird das Symbol `not-applicable` zurückgegeben, ansonsten jedoch die neu konstruierte Murmel. Per `(apply #'make-visco-marble ...)` wird die neue Murmel erzeugt. Hier werden dann automatisch eine Reihe von „Inferenzen“ obiger Art durchgeführt (für sie finden sich direkte Entsprechungen in der formalen Sprachdefinition im Anhang – dort wird z.B. definiert, unter welchen Bedinungen welche räumlichen Relationen expliziert werden).

Es wird zwischen *memorierten* und *nichtmemorierten* Operatoren unterschieden (`:stored-operator ...`). Ein memorierter Operator erzeugt bei seiner Anwendung automatisch ein CLOS-Operatorobjekt und verknüpft Argument(e) und das per Operatoranwendung gewonnene Resultat über dieses miteinander – so wird z.B. die räumliche Relation „Mittelpunkt von“ per Operatoranwendung explizit gemacht. Das Operatorobjekt repräsentiert also die Kanten *mittelpunkt_von* bzw. *hat_mittelpunkt* (s.o.). Das Resultat (der Mittelpunkt) wird dabei entweder vom Operator

neu kreiert oder aber ein bereits an dieser Position vorhandener Punkt wird zurückgegeben. Ähnliches geschieht für alle vom Benutzer explizit zu machenden räumlichen Relationen (alle anderen werden ja automatisch anhand der Geometrie errechnet), so z.B. für die Winkeleinschränker und Schnittpunktberechnung.

Syntaxgesteuerter Grafikeditor

Die eigentliche Oberfläche für VISCO wird nun durch den syntaxgesteuerten Grafikeditor dargestellt. Wie bereit diskutiert, wird durch seine Benuztzung im ASG-Repository die abstrakte Syntax der aktuellen VISCO-Definition in Form eines gerichteten Hypergraphen erzeugt. Bei der Gestaltung der Oberfläche wurde versucht, die oben dargestellten Design-Kriterien (wie Konsistenz, Einfachheit, etc.) nach Foley et al. umzusetzen ([FDFH96]).

Der Grafikeditor stellt mit ca. 222 kB LISP-Code die aufwendigste Komponente des Prototypen dar.

Benutzungsoberfläche: Zunächst soll die implementierte Oberfläche vorgestellt werden: in Abb. 6.7 sind zwei große Hauptfenster zu erkennen, die mit „VISCO“ (li.) und „VISCO Buttons“ (re.) bezeichnet sind. Eigentlich sollten sie in ein Fenster integriert werden, was aufgrund technischer Probleme nicht möglich war. Bei ihnen handelt es sich um zwei miteinander kommunizierende LISP-Prozesse; jeder ist eine eigene CLIM-Anwendung.

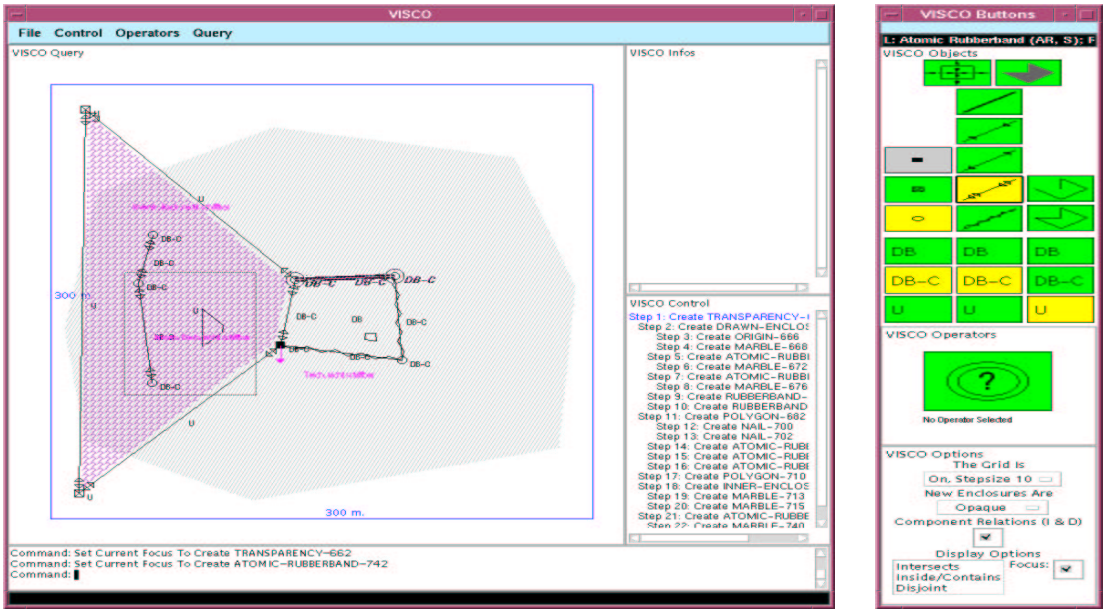


Abbildung 6.7: Der syntaxgesteuerte Grafikeditor

Das **„VISCO“-Fenster** selbst besteht wiederum aus vier Subfenstern:

- Das („VISCO Query“) *Hauptfenster* zur Konstruktion von Anfragen befindet sich in der Mitte,
- rechts oben liegt das („VISCO Infos“) *Infofenster,* der
- *Stöberer* („VISCO Control“) ist rechts unten angebracht, und schließlich
- findet sich unten die bereits bekannte *CLIM-Kommandozeile.*

Sinn und Zweck der ersten drei Fenster sollen nun etwas ausführlicher diskutiert werden.

Hauptfenster „VISCO Query“: Das große mittlere Hauptfenster stellt die Arbeitsfläche dar – hier wird interaktiv vom Benutzer konstruiert. Einige typische Operationen sind

- Erzeugen von Objekten,
- Löschen von Objekten,
- Manipulieren (z.B. Verschieben) von Objekten,
- Inspizieren von Objekten,
- Rekonstruktion von Objekten (z.B. kann ein Murmel in einen Nagel verwandelt werden),
- Anwenden eines bereits ausgewählten Operators auf ein (oder zwei) zu selektierende(s) Objekt(e) (Präfixoperatoranwendung),
- Erzeugen einer neuen Strecke durch Aggregation bereits vorhandener Endpunkte, eines Polygones durch Aggregation bereits vorhandener Strecken, etc.
- Setzen des aktuellen Fokus,
- Setzen der aktuellen Folie.

Generell gilt, daß es keine kongruenten Punkte und Strecken gibt. Zwei Komplexobjekte mit gemeinsamen Komponenten haben zumindest stets eine nicht-gemeinsame Komponente. Die Implementierungen der *Objekt-Erzeugungssequenzen* sind relativ komplex, da der Benutzer z.B. zur Erzeugung einer neuen Strecke per „Rubberbanding“

1. sowohl einen neuen Punkt per (linkem) Mausklick erzeugen will, als auch
2. bereits vorhandene Punkte verwenden will (durch Aggregation).

Wird die Erzeugung der Strecke abgebrochen (Cancel, rechte Maustaste), so muß ein für diese Strecke *neu erzeugter Punkt* – Punkte sind vollwertige Objekte! – wie auch die Strecke selbst gelöscht werden; ein im Rahmen der Streckenerzeugung *aggregierter Punkt* muß jedoch erhalten bleiben.

Beim Erzeugen eines neuen Polygones sollte der Benutzer nun

1. sowohl neue Segmente erzeugen können (die noch nicht vorhanden sind), und zwar *nach obiger Methode,*
2. als auch bereits vorhandene Segmente per Aggregierung in das neue Polygon integrieren dürfen.

Der Fall eines Abbruches ist analog zu behandeln.

Es wird deutlich, daß das Erzeugen von zusammengesetzten (komplexen) Objekten eine mehrstufige Handlungssequenz erfordert: auf jeden Fall muß es die Möglichkeit geben, *während* dieser Handlungssequenz z.B. die Typen der neu zu erzeugenden Punkte bzw. Komponenten zu variieren. So soll der nächste Punkt eines zu erzeugenden Polygones vielleicht eine Murmel, der übernächste jedoch wieder ein Nagel sein – analoges gilt natürlich für die Segmente des Polygones.

Der *Konstruktionskontext* bzw. *-zustand* oder *Modus* des Grafikeditors kann im „VISCO Buttons“-Fenster vom Benutzer verändert werden, *während* er eine zusammengesetzte Handlungssequenz im Hauptfenster ausführt (daher wurden auch *zwei* konkurrente parallele CLIM-Anwendungen

notwendig). Dies ist für *atomare* Handlungen natürlich nicht der Fall, da hier zwischendurch keine Teilhandlungen stattfinden.

Da Objekte beim Konstruieren *aggregiert* werden können, muß eine Möglichkeit vorgesehen werden, Objekte zu *referenzieren*. Dies wird durch aktives Feedback in Form von dynamischen Hervorhebungen (Highlighting) unterstützt: beim Konstruieren mit der Maus werden dabei die im gerade aktuellen Konstruktionskontext (s. „VISCO Buttons"-Fenster) verwendbaren Knoten bzw. Objekte hervorgehoben, während nicht verwendbare Teile inaktiv bzgl. Zeigegesten mit der Maus sind. Ein bereits vorhandenes Polygon kann z.B. nicht Teil eines gerade zu konstruierenden Rechteckes werden, wohl aber eines seiner Segmente. Das Referenzieren von Objekten durch verschieden genaues Zeigen wurde bereits in Kap. 2 diskutiert und wurde hier genauso implementiert (s. Abb. 6.8).

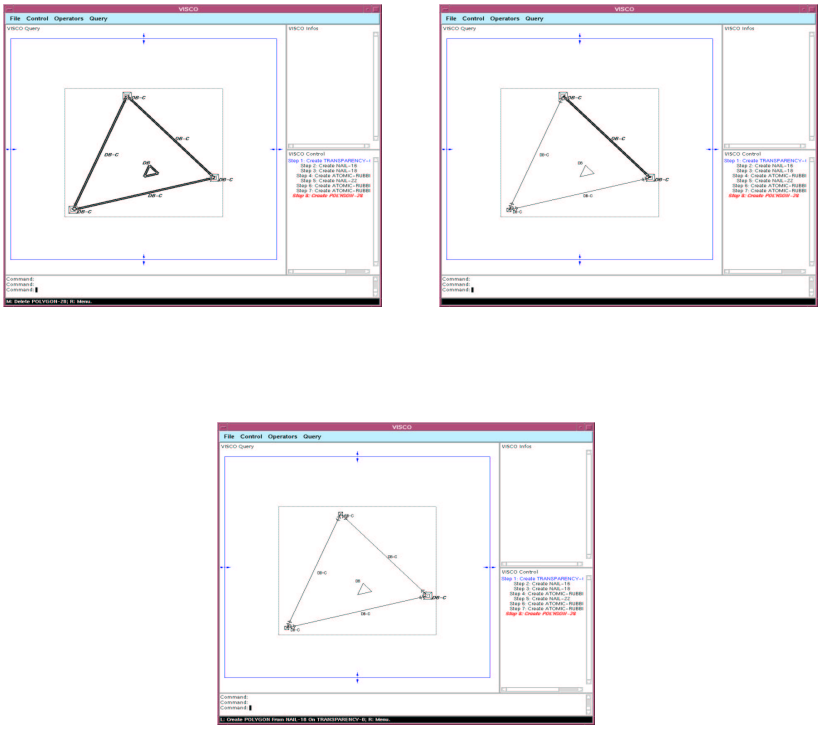


Abbildung 6.8: Referenzieren von Objekten

Wird nun der Operator „Erzeuge Mittelpunkt" auf ein Objekt angewendet, an dessen Mittelpunktsposition bereits ein Punktobjekt existiert, so wird für dieses so referenzierte Objekt die zusätzliche Einschränkung „Mittelpunkt von" eingetragen – ansonsten wird ein neues Punktobjekt anhand des sichtbaren Grafikeditor-Modus (s. „VISCO Buttons"-Fenster) erzeugt (s. obige Diskussion des Operators im ASG-Modul). Mehrere Objekte können einen gemeinsamen Mittelpunkt haben: jede erwünschte „Mittelpunkt von"-Einschränkung für ein Objekt muß wie oben beschrieben vom Benutzer explizit gemacht werden. Es gibt keine kongruenten Punkte. Analog wird für den Operator „Erzeuge Schnittpunkt" verfahren - auch hier ist daran zu denken, daß mehrere Strecken sich in einem Punkt schneiden können. Der Operator ist jedoch lediglich zweistellig, so daß mehrfache Operatoranwendungen nötig werden.

Bei den durch Operatoren berechneten bzw. referenzierten Objekten handelt es sich alsdann um *abgeleitete Objekte*: hierzu gehören sowohl Mittel- als auch Schnittpunkt, ebenso aber innere, äußere und ϵ-Gebiete. Abgeleitete Punkte werden mit einem „D" (für „derived") versehen, und abgeleitete Gebiete können visuell von den konstanten Gebieten unterschieden werden. Wird das Argument

des entsp. Operators nun verändert – z.B. die Form eines Polygones, welches Argument eines ϵ-Gebietes ist –, so wird das abgeleitete Objekt automatisch angepaßt bzw. neuberechnet. Da die Operatoren meist keine bijektiven Funktionen implementieren (so ist zwar der Mittelpunkt einer Strecke eindeutig, zu einem Mittelpunkt gibt es jedoch beliebig viele Strecken), dürfen abgeleitete Objekte nicht verschoben werden (Lösungsmöglichkeiten werden in [JF93, Bor81] dargestellt).

Infofenster „VISCO Infos“: Das rechts oben angebrachte Infofenster dient einfach zur textuellen Darstellung verschiedenster Informationen und Meldungen.

Stöberer „VISCO Control“: Rechts unten ist der sehr wichtige Stöberer (Browser) angebracht: jeder in dieser Liste auftauchende Eintrag stellt einen Konstruktionsschritt des Benutzers und somit die Konstruktionshistorie dar. Hierzu muß bemerkt werden, daß die im Stöberer dargestellte Konstruktionsgeschichte nicht vollständig identisch mit der intern verwalteten Konstruktionsgeschichte ist – einige Einträge in der internen Historie dienen lediglich Verwaltungszwecken: sie sind daher für den Benutzer irrelevant und würden ihn lediglich unnötig belasten.

Der *aktuelle Fokus* liegt stets auf einem (gerade „aktuellen“) Konstruktionsschritt und wird im Stöberer *kursiv und rot* dargestellt. Wird ein neues VISCO-Objekt im Arbeitsfenster erzeugt, so liegt der aktuelle Fokus zunächst auf diesem. Mit Hilfe des Stöberers kann der Benutzer die Konstruktionsgeschichte der aktuellen VISCO-Definition schrittweise nachvollziehen, indem er die einzelnen Einträge sukzessive selektiert. Durch Selektion eines Eintrages wird der aktuelle Fokus auf diesen Konstruktionsschritt gesetzt. Das zu jedem so bestimmten Konstruktionszeitpunkt gehörende Diagramm wird dann im Hauptfenster eingeblendet (s. Kap. 5: eine VISCO-Definition ist eine Sequenz von Diagrammen). Das aktuelle Fokus-Objekt wird im Hauptfenster mit einem strichlierten Rahmen gekennzeichnet.

Der Grafikeditor ist so gestaltet, daß stets nur Objekte auf der *aktuellen Folie* konstruiert werden können: die aktuelle Folie wird blau dargestellt, und auch der zur gerade aktuellen Folie gehörende Konstruktionsschritt im Stöberer.

Folgende Operationen werden auf den Einträgen des Stöberers angeboten:

- Setzen des aktuellen Fokus,
- Setzen der aktuellen Folie (falls es sich um einen Folienkonstruktionsschritt handelt),
- Löschen des Eintrages (hierbei handelt es sich um eine kritische Operation, die bei Nichtgelingen – s.o. – automatisch zurückgenomen wird).

Das **„VISCO Buttons“-Fenster** selbst besteht wiederum aus drei Subfenstern (s. Abb. 6.9, von oben nach unten):

- Das „VISCO Objects“-Fenster beinhaltet eine Reihe von Schaltern zum Festlegen des aktuellen Konstruktionskontextes (Modus).
- Das Fenster „VISCO Operators“ dient zur Darstellung eines gerade ausgewählten Operators (in Form eines „Operator Icons“ lt. Chang, s. Kap. 2).
- Das Fenster „VISCO Options“ regelt verschiedene Darstellungsattribute und -optionen.

Prinzipiell gilt folgende konsistent verwendete Farbgebung für Schalter:

Grün: der Schalter ist deselektiert, aber prinzipiell selektierbar.

Grau: der Schalter ist deselektiert und nicht selektierbar.

Gelb: der Schalter ist selektiert.

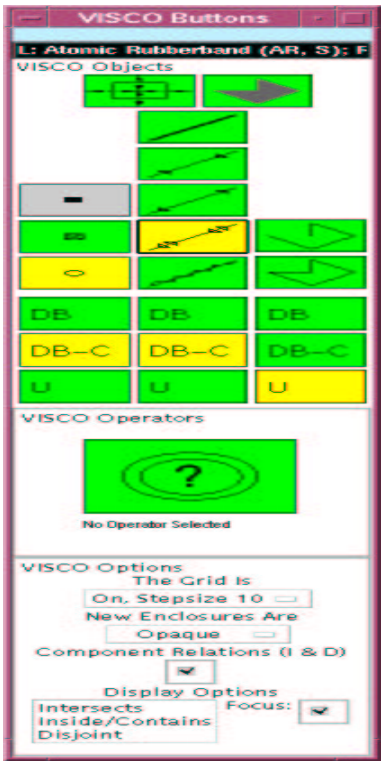


Abbildung 6.9: Das „VISCO Buttons“-Fenster

Hellgrün: der Schalter ist deselektiert und prinzipiell selektierbar, aber momentan nicht.

Hellgelb: der Schalter ist selektiert, aber momentan nicht deselektierbar.

Das „momentan“ bezieht sich meist auf einen temporären Zustand, z.B. während der Objekterzeugung: erzeugt der Benutzer z.B. gerade ein „skizziertes Gebiet“, so leuchtet der entsp. Schalter hellgelb, denn er ist zwar an, darf aber momentan nicht deselektiert werden, da der Benutzer ja gerade ein Gebiet erzeugt.

Wiederum soll nun jedes Teilfenster diskutiert werden:

„VISCO Objects“: Bis auf die obersten beiden Schalter (die für Overheadprojektorfolie und skizziertes Gebiet stehen) sind alle Schalter zu Spalten zu jeweils drei Schaltern zusammengefaßt: die Spalten heißen Punktschalter, Segmentschalter, Ketten- und Polygonschalter. Pro Spalte kann nur ein Schalter selektiert werden. Auch schließen sich die Schalter für Gebiet und Folie sowie die der drei Spalten gegenseitig aus.

Leicht abgesetzt darunter finden sich ebenfalls drei Spalten zu drei Schaltern mit den Aufschriften „DB DB-C U“: jede Spalte bezieht sich auf die über ihr liegende Spalte. Es handelt sich hierbei um die sog. *Statusschalter;* die Statusschalter einer Spalte schließen sich gegenseitig aus. Die *erste Spalte* bezieht sich somit auf den *Status von Punkten,* die *zweite* auf den *Status von Segmenten,* und die *dritte* auf den *Status von Polygonen und Ketten.* Die entsp. Objektschalter können nur selektiert werden, wenn für die entsp. Spalte ein Statusschalter an ist. Ein Statusschalter kann per linkem Mausknopf (mehrfach) selektiert, per mittlerem Mausknopf deselektiert werden (s.u.).

Objekte mit DB-Status müssen stets gegen primäre Objektes des Datenbestandes abgeglichen werden, während dies für DB-C-Objekte nicht der Fall ist (sie müssen lediglich gegen Objekte des Datenbestandes abgeglichen werden). Objekte mit Status „U“ (Universe) oder auch Hilfsob-

jekte (s. Kap. 5) müssen nicht im Datenbestand vorliegen – der Prototyp verlangt jedoch, daß diese Objekte *konstruierbar* sind, s. u. Nicht jede Kombination von Statusschaltern ist erlaubt: eine Kombination *(Punkt,Strecke)=(U,DB)* ist verboten (s. Kap. 5), denn das Vorhandensein der Strecke im Datenbestand (DB-C) impliziert natürlich das Vorhandensein ihrer Endpunkte im Datenbestand (DB-C). Da hier teilweise komplizierte Kombinationen vorliegen, aber stets nur legale Kombinationen eingegeben werden dürfen, muß der Benutzer bei der Auswahl der Statusattribute unterstützt werden: per Mehrfachselektion (linke Maustaste) ein und desselben Statusschalters wird automatisch die nächste legale Kombination eingestellt, in der der Schalter ebenfalls an ist. Der Schalter kann per rechter Maustaste deselektiert werden. Selektiert der Benutzer einen bel. Statusschalter, so wird als nächste Kombination diejenige legale Kombination eingestellt, die minimal von der vorherigen abweicht – hierzu werden andere Schalter evtl. ein- oder ausgeschaltet.

Wird nun ein *neues* Objekt im Arbeitsfenster konstruiert, so bilden alle aktivierten Schalter den *Konstruktionskontext* oder *Modus* des Grafikeditors: sie bestimmen Typ und Status noch zu erzeugender Objekte.

Wird eine neue Strecke erzeugt, so können nur dann *neue* Endpunkte erzeugt werden, wenn zusätzlich ein Punktschalter selektiert ist. Ist kein Punktschalter selektiert (und kann somit Typ und Status von neuen Punkten nicht bestimmt werden), so besteht im Rahmen der Streckenerzeugung immer noch die Möglichkeit, bereits vorhandene Punkte als eigene Endpunkte zu aggregieren. Eine Aggregation ändert bei Bedarf *automatisch den Status der so aggregierten Objekte:* z.B. werden aggregierte DB-Komponentenobjekte automatisch zu DB-C-Objekten (vorausgesetzt, es wird nicht gerade ein U-Elternobjekt erzeugt). U-Objekte hingegen können nur dann als Komponenten aggregiert werden, wenn das aggregierende Elternobjekt selbst U-Status hat.

Für ein zu aggregierendes Objekt muß folgende Bedingung gelten: Ein Objekt kann aggregiert werden, wenn es entweder

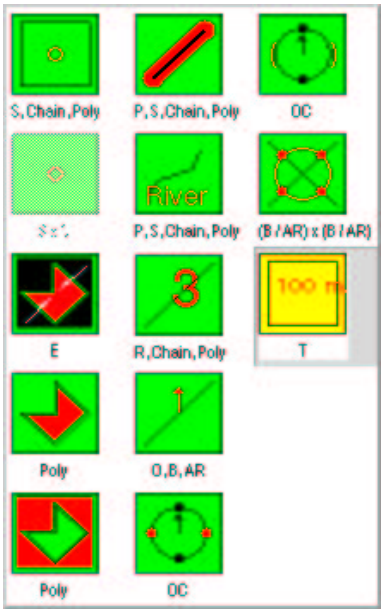
1. einen Status hat, der *konform* mit dem Status des neu zu konstruierenden Elternobjektes ist,
2. oder aber der Status des zu aggregierenden Objektes so *verändert werden kann,* daß er
   (a) *konform* mit dem neu zu konstruierenden Elternobjekt *wird,* und zusätzlich
   (b) *konform* mit allen bereits vorhandenen Elternobjekten *bleibt.* Der Status bereits vorhandener Elternobjekte wird jedoch auf keinen Fall verändert.

Konform bedeutet, daß die Teilmengenrelation korrekt aufrechterhalten werden muß: $\mathsf{DB} \subset \mathsf{DB\_C} \subset \mathsf{U}$ (s. Kap. 5). Natürlich gilt, daß die Komponenten eines Komplexobjektes nicht in einer „größeren“ Menge als das Elternobjekt selbst sein dürfen. Der Status eines Objektes wird automatisch als entsp. Beschriftung am Objekt visualisiert. Beschriftungen können bei Bedarf interaktiv verschoben werden.

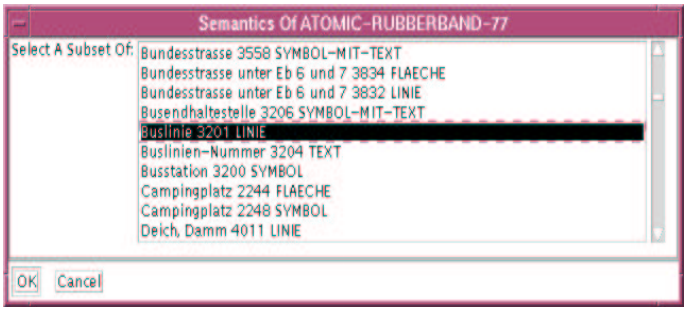
„VISCO Operators“: Hier wird der gerade aktuelle Operator (in Form eines Ikones) dargestellt: über eine spezielle Geste kann der aktuelle Operator auf einem Objekt (oder auch auf ein Objektpaar im Falle eines binären Operators) im Arbeitsfenster angewendet werden – da hier zunächst der Operator und dann die bzw. das Argument(e) ausgewählt werden, spricht man von *Präfixoperatoranwendung.*

Ein Selektion des aktuellen Operatorikones erzeugt ein Pop-Up-Menü, welches zur Auswahl eines neuen aktuellen Operators aus der Liste der momentan *prinzipiell anwendbaren Operatoren (s.u.)* auffordert (Abb. 6.10(a), ein Operator ist momentan nicht anwendbar, ein anderer wird gerade ausgewählt). Das Ikon des so gewählten Operators ersetzt dann das Ikon des vorherigen aktuellen Operators. Operatoren können aber auch über ein textuelles Menü in der Menüleiste ausgewählt werden.

Folgende Operatoren sind vorgesehen (sie korrespondieren mehr oder weniger direkt zu Operatoren im ASG-Modul):



(a) Ikonen für Operatoren



(b) Festlegung der Thematik

Abbildung 6.10: Operatoren

- Erzeuge Mittelpunkt
- Erzeuge Schnittpunkt
- Erzeuge äußeres Gebiet
- Erzeuge inneres Gebiet
- Erzeuge negiertes skizziertes Gebiet
- Erzeuge ϵ-Gebiet (der Radius wird interaktiv bestimmt)
- Setze Semantik (Thematik)
- Setze Maximumsbedingung
- Setze Orientierungseinschränkung (Zeiger)
- Erzeuge Skalenmarke für Skala (wenn noch keine Skala vorhanden ist, wird automatisch eine erzeugt)
- Erzeuge Skalenintervall (wenn noch keine Skala vorhanden ist, wird automatisch eine erzeugt)
- Setze Folieneigenschaften (Breite, Länge, Skalierungsintervalle, etc.),
- Erzeuge Winkeleinschränker

Die einzigen (bisher implementierten) binären Operatoren sind „Erzeuge Winkeleinschränker“ und „Erzeuge Schnittpunkt“. Werden durch Operatoren nun z.B. neue Punkte erzeugt, so werden Status und Typ dieser wiederum anhand des aktuellen Konstruktionskontextes ermittelt – sind dort z.B. „DB-Murmeln“ für Punkte eingestellt und wird ein neuer Mittelpunkt per Operatoranwendung erzeugt, so wird dieser Operator eine DB-Murmel im Arbeitsfenster erzeugen. Die verwendeten Operatorikonen sind übrigens teilweise dynamisch und verändern sich in Abhängigkeit vom eingestellten Konstruktionskontext.

Hier wird deutlich, daß die Operatoren u.U. recht komplizierte *Bedingungen für ihre Anwendbarkeit prüfen müssen:* so ist der Operator „Erzeuge Mittelpunkt“ nur dann (für einen Konstruktionskontext Punkttyp = Murmel) anwendbar, wenn die neue Murmel tatsächlich in (mindestens) einem die Murmel enthaltenem Gebiet zum Liegen kommt.

Die *Ermittlung der prinzipiell anwendbaren Operatoren* ist daher ein aufwendiger Prozeß (er wird jedesmal durchgeführt, wenn der Benutzer einen Operator aus der Menüleiste oder das korrespondierende Ikon auswählt):

- Ein *unärer Operator ist prinzipiell anwendbar,* wenn es mindestens *ein Objekt* in der aktuellen VISCO-Definition gibt, welches die Anwendbarkeitsbedingungen des Operators erfüllt.
- Ein *binärer Operator ist prinzipiell anwendbar,* wenn es mindestens *ein Objektpaar* in der aktuellen VISCO-Definition gibt, welches die Anwendbarkeitsbedingungen des Operators erfüllt.

Ist ein Operator prinzipiell anwendbar, so wird sein Ikon grün hinterlegt, bzw. kann aus der Menüleiste textuell ausgewählt werden (ansonsten wird er in „Geisterschrift“ dargestellt, bzw. sein Ikon grau hinterlegt, s.o.).

Teilweise können die Operatoren direkt auf Operatoren im ASG-Modul abgebildet werden – teilweise müssen aber zuvor noch weitere Parameter interaktiv vom Benutzer bestimmt werden. So ist z.B. für die Erzeugung eines ϵ-Gebietes die Kenntnis des Radius r nötig: nachdem der Benutzer zunächst den Operator „Erzeuge ϵ-Gebiet“ und dann das Argumentobjekt für den Operator selektiert hat, kann er interaktiv bzw. direkt-manipulativ mit der Maus den Radius der Umgebung aufziehen.

Ähnlich interaktiv werden Skalenmarken und -intervalle sowie die erlaubten Winkelabweichungen der Winkeleinschränker bestimmt. Die Thematik eines Objektes wird durch Mehrfachselektion von Einträgen aus dem Objektschlüsselkatalog festgelegt (s. Abb. 6.10(b)). Eine Maximumsbeschränkung wird einfach als Zahl eingetippt.

Auf den so erzeugten Metaobjekten (Zeiger, Skala etc.) können später direkt-manipulative Operationen ausgeführt werden: so kann man z.B. interaktiv Zeiger drehen und in der Länge sowie Startposition verändern, Skalenmarken löschen, die Winkelabweichung eines Winkeleinschränkers vergrößern oder verkleinern, die Beschriftungen eines Objektes verschieben, etc. Nahezu alle durch Operatoren erzeugte Metaobjekte lassen sich per Mausgeste direkt löschen (Zeiger, Skala, Winkeleinschränker, Thematik, Maximumsbeschränkung).

„VISCO Options“: Hier finden sich nun diverse Schalter zur Einstellung von Darstellungsoptionen etc.:

- Ein *aktives Gitter* zur Unterstützung des Konstruktionsprozesses im Arbeitsfenster läßt sich ein- und ausschalten und in der Feinheit variieren (das Gitter ist unsichtbar).
- Neue Gebiete werden entweder *opak (undurchsichtig) oder transparent (durchscheinend)* dargestellt.
- Werden Objekte auf der Arbeitsfläche aggregiert (z.B. im Rahmen einer Streckenbildung), so kann hier eingestellt werden, *daß die topologischen Relationen „Schneidet“ und „Schneidet nicht“ für diese Objekte durch die Aggregierung gelöscht werden sollen (s. Kap. 5)* Ist also der Schalter „Component Relations“ aus, so werden die „Schneidet“ und „Schneidet nicht“ Relationen zu und von den so aggregierten Objekten verworfen – hierbei handelt es sich um die in Kap. 5 beschriebene Relaxierungsmöglichkeit. Diese bzgl. der Relationen „Schneidet“ und „Schneidet nicht“ identitätslosen Komponentenobjekte werden im Gegensatz zu den anderen Objekten *hellgrau* dargestellt.
- Die errechneten topologischen Relationen können für das aktuelle Fokusobjekt *visualisiert* werden. Hierfür wird jedes Objekt (natürlich mit Ausnahme des Fokusobjektes selbst) in

einer speziellen Farbe dargestellt, welche die errechnete räumliche Beschränkung *zum Fokusobjekt* kodiert:

Rot: ein rotes Objekt soll das Fokusobjekt „schneiden“.

Blau: ein blaues Objekt „schneidet nicht“ das Fokusobjekt.

Grün: das Fokusobjekt muß in einem grünen Objekt (Gebiet) „enthalten“ sein. Ist das Fokusobjekt selbst ein transparentes Gebiet ist, so werden auch die Objekte grün dargestellt, die das Gebiet „enthält“.

Alle Relationen können natürlich gleichzeitig visualisiert werden, da sie sich gegenseitig ausschließen. Die Visualisierung der errechneten topologischen Relationen bzw. Beschränkungen (Constraints) beziehen sich stets auf das Objekt, auf dem der *aktuelle Fokus* liegt (Fokusobjekt). Mit Hilfe des Stöberers kann schrittweise jede explizierte topologische Beschränkung für die aktuelle VISCO-Definition auf diese Weise inspiziert werden.

- Schließlich gibt es noch einen Schalter, mit dessen Hilfe das aktuelle Fokusobjekt im Arbeitsfenster mit einem *markierenden Rechteckt (zur visuellen Orientierung)* versehen werden kann. Das aktuelle Fokusobjekt bzw. der korrespondierenden Konstruktionseintrag in der Historie wird ja auch stets im Stöberer rot und kursiv dargestellt.

Diskussion einiger interner Aspekte des Grafikeditors Die Klassen des Grafikeditors erweitern die Klassen der ASG-Schicht durch Vererbung – zusätzliche Eigenschaften kommen hinzu, um die konkrete Syntax einer VISCO-Definition darstellen zu können, wie Farbe, Füllmuster, etc.

Hier ergibt sich ein typisches und in der Literatur zur objektorientierten Programmierung ausgiebig diskutiertes Problem ([GHJV96, Kap. 3, „Erzeugungsmuster-Katalog“]): Im bereits erstellten ASG-Modul werden Instanzen der ASG-Basisklassen erzeugt – für den Grafikeditor ist es jedoch notwendig, daß nicht Instanzen der ASG-Klassen, sondern Instanzen der Grafikeditorklassen erzeugt werden. Intern werden im ASG-Modul die Instanzen durch Aufruf spezieller Konstruktoren erzeugt: so wurde im oben (S. 153) dargestellten Operator `create-marble` durch Aufruf des Konstruktors `make-visco-marble` innerhalb von

```
(:code ((apply #'make-visco-marble transparency status x y initargs)))
```

das neue Murmel-Objekt vom Typ `marble` konstruiert – tatsächlich wird jedoch eine `gui-marble` benötigt (also eine Murmel für den Grafikeditor). Daher wird der Konstruktor `make-visco-marble` nun entsprechend überladen (es handelt sich bereits um eine generische Funktion), und zwar für die spezielle `transparency`-Unterklasse `gui-transparency`:

```
(defmethod make-visco-marble ((transparency gui-transparency) (status symbol)
                              (x number) (y number) &rest initargs)
  (let ((obj (call-next-method)))
    (apply #'change-class obj 'gui-marble initargs)))
```

Praktischerweise gibt es in Common LISP die generische Funktion `change-class` – eine speziell überladene `update-instance-for-different-class :after`-Methode nimmt dann die noch verbleibenden Initialisierungen der so aufgewerteten Instanz vor: z.B. werden hier die grafischen Attribute der neuen Instanz bestimmt.

Im folgenden soll erläutert werden, wie interaktive Handlungen des Benutzers (Löschen, Verschieben, Erzeugen von Objekten etc.) an das ASG-Modul kommuniziert werden: Oben wurde erwähnt, daß nach jeder Operation der ASG entsprechend umkonstruiert oder ergänzt werden muß. Hierzu wird bei allen Operationen prinzipiell folgendes dreistufiges Verfahren angewendet:

1. *Sichern* der aktuellen *internen Konstruktionshistorie.* Die interne Konstruktionshistorie ist die Sequenz *aller* vom Benutzer bisher durchgeführten Handlungen, während die *externe*

Konstruktionshistorie eine spezielle Teilmenge dieser ist, die auch im Stöberer dargestellt wird. Ein *Eintrag* in der Historie repräsentiert eine speziellen Handlung, Operation, o.ä. Im folgenden beziehe ich mich auf die interne Historie.

2. *Manipulieren* der aktuellen Konstruktionshistorie – dies umfaßt
   - *Löschen* eines Eintrages,
   - *Hinzufügen* eines Eintrages,
   - *Austauschen* eines Eintrages.
3. *Vollständiges Rekonstruieren* des ASG anhand der so veränderten Historie.
4. Beim ersten Nichterfolg *Widerherstellen* der gesicherten Historie – die Benutzerhandlung ist dann nicht zulässig.

Bei komplexen VISCO-Definitionen können durchaus einige Dutzend Einträge in der Historie vorliegen – die Rekonstruktion geschieht jedoch hinreichend schnell und somit für den Benutzer nahezu sofort. Dennoch ist diese Vorgehen nicht effizient genug, um z.B. *während* des interaktiven Verschiebens eines Objektes den ASG bei jeder Positionsveränderung der Maus neu zu rekonstruieren – der ASG wird erst dann rekonstruiert, wenn der Benutzer sich endgültig für eine neue Position des Objektes entschieden hat. Da der ASG dann eventuell nicht rekonstruierbar ist, merkt der Benutzer dies leider erst, nachdem er die Handlung bereits vorgenommen hat. Natürlich macht das System diese Handlung dann automatisch rückgängig (s. obiges Vorgehen). Es wäre dennoch nützlich, dem Benutzer schon *während der Durchführung der Handlung* mitzuteilen, ob der gerade ersichtliche Zustand syntaktisch korrekt ist. Während der interaktiven Handlungsdurchführung werden daher nur einige sehr wesentliche, immer einzuhaltenen syntaktischen Bedinungen geprüft – so darf z.B. eine Murmel nicht so verschoben werden, daß sie in keinem Gebiet mehr liegt. Diese Bedingung wird auch während der Handlungsdurchführung geprüft – alle komplexeren Bedingungen sind jedoch nur durch Rekonstruktion prüfbar (obiges Vorgehen).

Durch oben beschriebenes Vorgehen wird auch die Implementierung einer allgemeinen (unbeschränkten) UNDO- und REDO-Funktionalität einfach möglich: vor jeder Operation wird die aktuelle Konstruktionsgeschichte kopiert und als oberstes Element eines Stapelspeichers (Stack) abgelegt. Ein UNDO erfordert dann einfach eine POP-Operation, die Installation der gepoppten Historie als aktuelle Historie und eine komplette Rekonstruktion eben dieser. Ein REDO-Mechanismus wird durch einen zweiten Stapel implementiert, auf den vor jeder UNDO-Operation die aktuellen Historie abgelegt wird. Wesentlich ist hierfür, daß ein Operator einen die Operation selbst vollständig charakterisierenden Eintrag für die Historie erzeugen kann – alle Effekte im Zustandsraum müssen dabei explizit gemacht werden (u.a. alle Operatorargumente, etc.).

Diskussion einiger (momentaner) Unzulänglichkeiten des Grafikeditors Während die *Geometrie* einer VISCO-Definition interaktiv vom Benutzer selbst erzeugt werden muß, sollte die *Ausgestaltung der konkreten Syntax* in Form von Farben für Objekte, Texturen für Gebiete etc. dem System überlassen werden. Hierfür ist eine interne *Präsentationskomponente* im Grafikeditor vorgesehen. Die Präsentationskomponente erzeugt somit nicht die Geometrie (wie dies z.B. beim „CIGALES“-System der Fall ist, s. Kap. 4), aber die konkreten Farbgebungen, Texturen und andere grafische Attribute, die in der abstakten Syntax (und somit im ASG-Modul) keine Relevanz haben. Überlappende Gebiete sollten z.B. so dargestellt werden, daß sie *visuell voneinander unterschieden* werden können. Die implementierte Präsentationskomponente ist jedoch noch sehr rudimentär und erzeugt teilweise schlechte Grafikattribute (s. Abb. 6.11, 6.12, 6.13).

Da es sich um einen *syntaxgesteuerten Grafikeditor* handelt und somit achtgegeben werden muß, daß stets syntaktisch wohlgeformte Graphen im ASG-Modul entstehen, sind die *Kontextforderungen* für spezielle Konstruktionsschritte teilweise sehr stark, was den Benutzer in den momentan durchführbaren Operationen *relativ stark einschränkt.* In der Literatur findet man daher oft die Forderung, daß ein syntaxgesteuerter Editor über *zwei Modi* verfügen sollte:

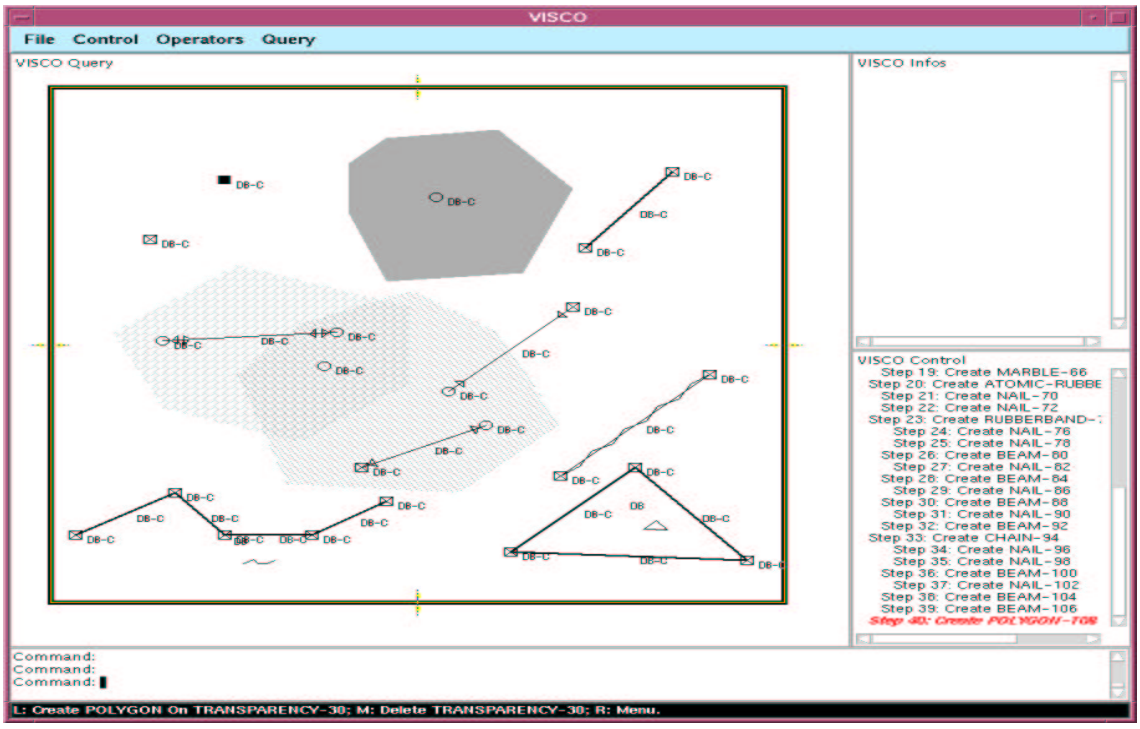


Abbildung 6.11: Präsentation von Objekten

- Einen Modus, indem jeder Schritt nur syntaktisch korrekte Nachfolgezustände erzeugt, und
- einen Modus, in dem *temporäre Inkonsistenzen in Kauf genommen werden.* Vorausgesetzt wird dabei, daß die Inkonsistenzen am Ende dieser Phase aufgelöst bzw. beseitigt werden können, so daß wieder ein korrekter Zustand entsteht.

Es ist klar, daß die syntaktischen Kontextanforderungen den Benutzer u.U. zu komplizierten Handlungssequenzen zwingen, die durch das Zulassen von temporär inkonsistenten Zuständen nicht erfolderlich wären (hierbei handelt es sich sozusagen um „nicht erlaubte Abkürzungen“ in einem komplizierten Zustandsgraphen). Für den VISCO-Editor wurde nur der erste Modus implementiert; wohlwissend, daß der zweite Modus wesentlich benutzerfreundlicher, aber auch sehr viel schwerer zu implementieren ist. Im Gegensatz zu GENED (s. Kap. 2) ist der Grafikeditor für VISCO kein Freiformeditor, sondern eine Oberfläche zum systematischen Konstruieren von VISCO-Definitionen.

Optimierender Compiler

Der optimierende Compiler für VISCO erzeugt anhand des abstrakten Syntaxgraphen (ASG) ein Programm, welches den räumlichen Datenbestand nach passenden Konstellationen durchsucht – es sollen also *Bindungen* von VISCO-Objekten zu Objekten im räumlichen Datenbestand hergestellt werden (in der formalen Sprachdefinition im Anhang werden diese Bindungen durch die Funktion *bound_to* repräsentiert). Der Compiler konnte durch 101 kB Sourcecode realisiert werden.

Betrachtet man sowohl den Datenbestand als auch die VISCO-Anfrage (ASG) als Graphen, so wird deutlich, daß hier ein spezielles *Teilgraphenisomorphismus-Problem* zu lösen ist: es ist also eine Abbildung gesucht, die bijektiv die Knoten der VISCO-Definition auf einen Teil der Knoten des Datenbestandes abbildet.

Definition (Isomorphe Graphen): Zwei Graphen $G_1 = (\mathsf{E}_1, \mathsf{K}_1)$ und $G_2 = (\mathsf{E}_2, \mathsf{K}_2)$ heißen *isomorph,* falls es eine Bijektion $\phi : \mathsf{E}_1 \rightarrow \mathsf{E}_2$ gibt, so daß $(u, v) \in \mathsf{K}_1 \Leftrightarrow (\phi(u), \phi(v)) \in \mathsf{K}_2$ gilt.

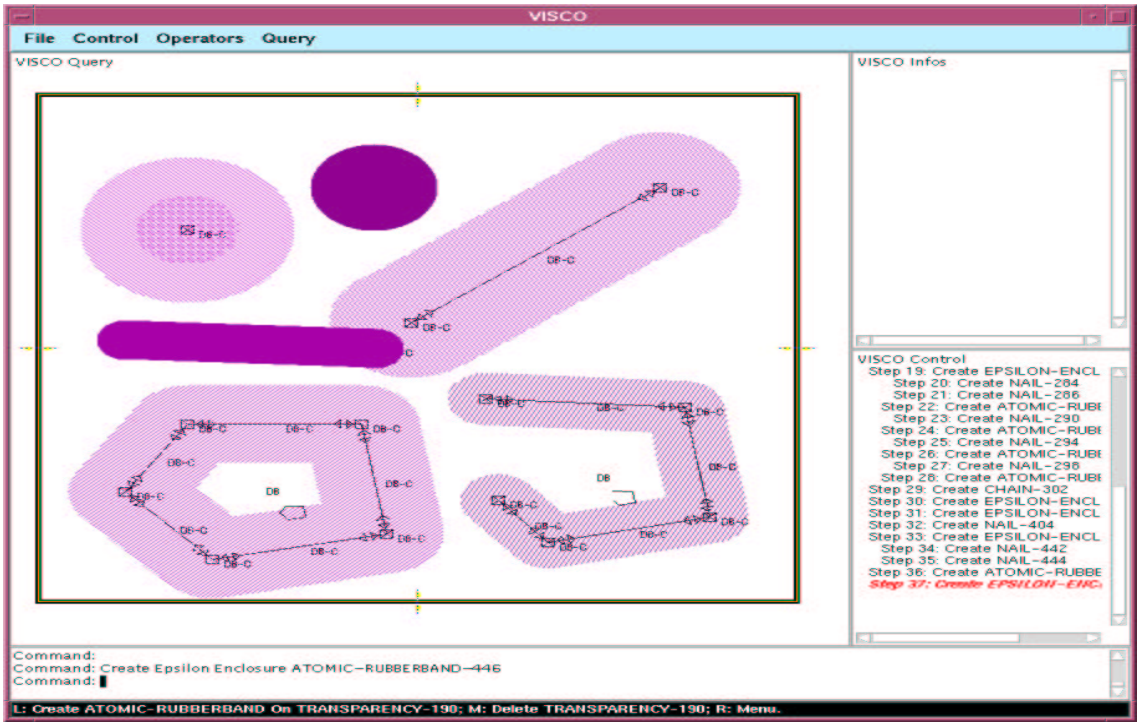


Abbildung 6.12: Präsentation von ϵ-Gebieten

Graph Q sei nun der ASG, und Graph DB' sei ein Teilgraph des Graphen $DB = (\mathsf{E}_{DB}, \mathsf{K}_{DB})$ des räumlichen Datenbestandes. Für einen solchen Teilgraphen DB' gilt: $\mathsf{E}_{DB'} \subseteq \mathsf{E}_{DB}$ und $\mathsf{K}_{DB'} \subseteq \mathsf{K}_{DB}$. Sind Q und DB' isomorph, so heißt ϕ *Teilgraphenisomorphismus* zwischen Q und DB.[2]

Es ist bekannt, daß das Teilgraphenisomorphismusproblem für allgemeine Graphen NP-vollständig ist ([FB96, S. 593]) – dies muß jedoch nicht für spezielle Graphen gelten. Im Gegensatz hierzu ist die NP-Vollständigkeit des Graphisomorphismusproblemes noch nicht bewiesen (und wird auch bezweifelt).

Für den hier betrachteten speziellen Teilgraphenisomorphismus müssen an ϕ noch weitere Bedingungen gestellt werden: So dürfen lediglich Punktknoten gegen Punkknoten abgeglichen werden, und auch mehrere Kantentypen müssen ja berücksichtig werden. Die Kanten des Graphen bzw. Tupel haben ja die Form $k \in \bigcup_{i \in \mathbb{N}, i \geq 1} \mathsf{Kantentypen} \times \mathsf{E}^i$, so daß obige Isomorphismusbedingung umgeschrieben werden müsste. Zusätzlich sind spezielle Funktionen bzw. Attribute auf den Kanten definiert (wie z.B. die Funktion *position** auf der *hat_position*-Eigenschaft, s.o.): prinzipiell könnte man für jedes mögliche Attribut einen eigenen Kantentyp einführen und somit auf Funktionen wie *position** verzichten. Man hätte dann eine Reihe von *hat_position_*(x, y)-Kanten – da hier auch relative Positionen, Orientierungen etc. eine Rolle spielen, bräuchte man sehr viele Kanten: hier soll nur demonstriert werden, daß es sich *prinzipiell* um ein spezielles Teilgraphenisomorphismusproblem handelt. Die Isomorphismusforderung würde dann $(relation, u_1, u_2, \ldots, u_n) \in \mathsf{K}_Q \Leftrightarrow (relation, \phi(u_1), \phi(u_2), \ldots, \phi(u_n)) \in \mathsf{K}_{DB'} \wedge bindbar(\phi(u_1), u_1) \wedge bindbar(\phi(u_2), u_2) \wedge \ldots \wedge bindbar(\phi(u_n), u_n)$ lauten. *bindbar* soll genau dann wahr werden, wenn Punkte gegen Punkte, Linien gegen Linien, etc. abgeglichen werden.

Da hier keine echte räumliche Datenbank verwendet wird, erzeugt der Compiler anhand des ASG einfach ein LISP-Programm, welches dann vom LISP-Compiler übersetzt und schließlich ausgeführt wird. Ein solches LISP-Programm stellt also letzlich Anfragen an die räumliche Datenbank

[2] Da es auch Knoten im ASG mit U-Status geben kann (also Knoten, die nicht notwendigerweise in K_{DB} enthalten sind), müsste man eigenlich nicht alle Teilgraphen von DB, sondern des Universums aller geometrischen Objekte betrachten. Die folgende Darstellung dient jedoch nur der Illustration.

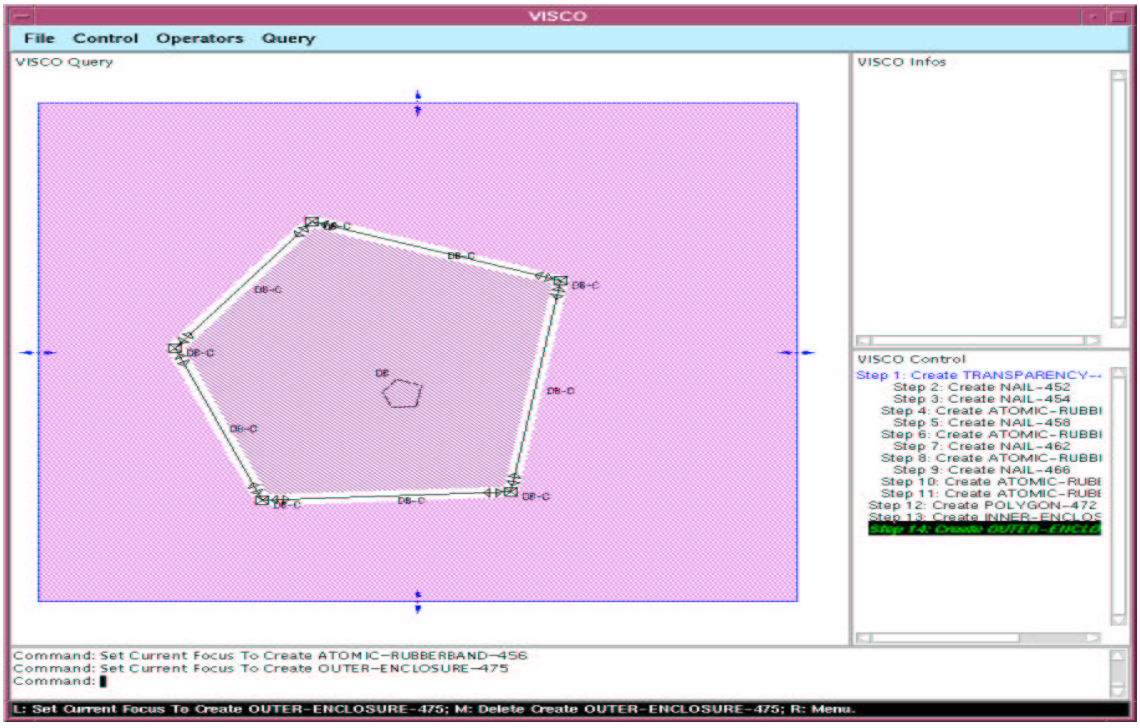


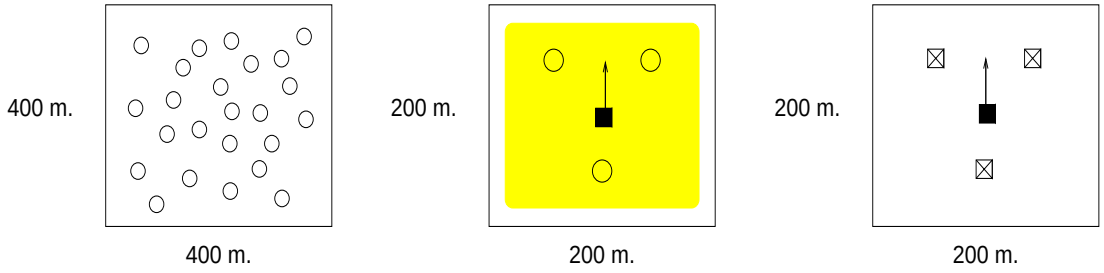
Abbildung 6.13: Präsentation von inneren und äußeren Gebieten

unter Verwendung von Instruktionen der Anfragesprache der verwendeten räumlichen Datenbank. Dies wird in vielen Fällen ein erweiterter SQL-Dialekt sein (s. Kap. 3). Hier werden jedoch die Schnittstellenfunktionen und -makros der Schicht „Räumlicher Datenbestand“ benutzt.

Die *Optimierbarkeit deklarativer Sprachen wie SQL* wird in der Literatur oft als großer Vorteil dargestellt (s. [Dat95, S. 501]): da z.B. das relationale Modell (und somit SQL) semantisch auf einer hohen Ebene anzusiedeln ist, gibt es *viele Freiheiten* in der konkreten Bearbeitung einer Anfrage. In der Regel gibt es eine große Anzahl von *möglichen Bearbeitungsplänen,* von denen ein *automatischer Optimierer* den besten auswählen kann. Aufgrund der Deklarativität repräsentiert eine einzelne SQL-Anfrage eine ganze Klasse von möglichen Ausführungsplänen: Dies ist z.B. für navigierende Anfragesprachen von hierarchischen Datenbanken nicht der Fall, da hier nur ein Plan beschrieben wird. Der *beste Plan* bzw. ein möglichst effizienter muß somit vom Anfragesteller selbst ermittelt werden. Ein Optimierer hingegen kann *hundertausende von Plänen generieren und automatisch bewerten, wozu ihm Wissen z.B. in Form von Datenbankstatistiken zur Verfügung steht, das einem menschlichen Anfrageformulierer fehlt.* Die Bearbeitungszeit eines guten und eines schlechten Planes für SQL differiert oftmals um einen Faktor von 10.000 und mehr (s. [Dat95, S. 504]).

Während die Optimierung in der Regel von einer speziellen Komponente (dem Optimierer) der Datenbank bzw. des Datenbankmanagementsystems (DBMS) als Teilaufgabe der Anfragebearbeitung durchgeführt wird, mußte für den VISCO-Prototypen ein externer Optimierer im Compiler vorgesehen werden, da keine echte räumliche Datenbank mit eigener Anfragebearbeitungskomponente verwendet wird.

Da ein spezielles Teilgraphenisomorphismusproblem gelöst werden muß (welches ja in der allgemeinen Form NP-vollständig ist), besteht Grund zu der Annahme, daß ein Optimierer sogar *unerläßlich* ist, da andererseits halbwegs komplexe Anfragen (in vernünftigen Zeitspannen) überhaupt nicht beantwortet werden können. Der hier implementierte Abgleichalgorithmus benötigt im schlechtesten Fall mindestens exponentielle Zeit.



(a) Datenbestand (b) Anfrage 1 (c) Anfrage 2

Abbildung 6.14: Komplexität zweier Anfragen

So erfordert z.B. die vollständige Bearbeitung der VISCO-Anfrage in Abb. 6.14(b) die Zeit $n(n-1)(n-2)(n-3)(n-4)$ (wenn n die Anzahl der Punkte des Datenbestandes in Abb. 6.14(a) ist), bzw. bei k Murmeln die (in k) exponentielle Zeit $\frac{n!}{(n-k+1)!}$. Im Kontrast hierzu steht die Anfrage in Abb. 6.14(c), deren komplette Bearbeitung (schlechtestenfalls) die Zeit $4n$ benötigt (wenn man den Zugriff auf den räumlichen Index mit $O(1)$ bewertet) – aber auch bei k Nägeln beträgt die Zeitkomplexität noch $O(k * n) = O(k)$, und ist somit linear in k.

Dies liegt natürlich daran, daß die Eigenschaft *hat_position* unter Verwendung des räumlichen Index sofort geprüft werden kann, sobald der Ursprung und somit das Koordinatensystem instantiiert sind – es ist dann keine weitere Suche mehr notwendig. In der KI unterscheidet man auch zwischen *informierten* und *uninformierten* Verfahren bzw. *starken* und *schwachen* Methoden. Die Rolle von *Wissen* wird u.a. darin gesehen, einen entscheidenden Beitrag zu Eingrenzung des Suchraumes beim Problemlösen zu leisten. Es stellt sich also die Frage, inwiefern z.B. Heuristiken zur effizienten bzw. möglichst *informierten Suche* und somit VISCO-Anfragebearbeitung genutzt werden können. Derartiges Wissen muß im Optimierer kodiert werden.

Hier kann beobachtet werden, daß *geometrische und metrische Einschränkungen* in der Regel sehr starke Bedingungen sind, da es sich um *quantitative Einschränkungen* handelt: Position, Winkel, Orientierung und Form diskriminieren stärker als z.B. die qualitativen topologischen Relationen (man denke z.B. an „schneidet nicht“) – diese beschreiben in der Regel sehr große Klassen von Konstellationen und sind deshalb als „weniger informiert“ einzustufen. Die Bevorzugung geometrischer und metrischer Eigenschaften und Relationen wird daher eine wesentliche Heuristik für den Optimierer sein.

Ein Modell des VISCO-Compilers: Wie oben dargestellt muß es Ziel des Compilers sein, ein Suchprogramm zu erzeugen, welches auf möglichst informierte und somit effiziente Weise den Datenbestand durchsucht, um sukzessive Bindungen bzw. Teile der Funktion ϕ herzustellen. *Informiert* bedeutet, daß die Suche so fortschreiten sollte, daß jeweils so wenig Alternativen wie möglich betrachtet werden müssen, so daß der Verzweigungsfaktor möglichst früh minimiert wird.

Der für VISCO verwendete *Suchalgorithmus* ist trivial: es handelt sich um eine einfache *Tiefensuche* (auch bekannt als das Rückziehungsverfahren). Tiefensuche bietet sich aus mehreren Gründen an:

- Das Verfahren ist einfach zu implementieren.
- Der Verwaltungsaufwand ist gering.
- Da es keine unendlichen Pfade im Datenbestand gibt, entstehen keine der bekannten Probleme, die gegen eine Tiefensuche sprechen würden.
- Der Datenbestand ist in der Regel sehr groß, so daß es unmöglich ist, z.B. in einer Breitensuche parallel alle möglichen Bindungen aufzubewahren.

- Die Umsetzung des ASG in ein entsp. Suchprogramm ist trivial (s.u.).

Zur Übersetzung einer VISCO-Anfrage geht der optimierende Compiler nun prinzipiell in *fünf Schritten* vor:

1. **Trage in den ASG eine Reihe von spez. Compilerkanten ein.**
   Diese Compilerkanten werden anhand der bereits vorhandenen Kanten des ASG ermittelt – ausschließlich Compilerkanten haben einen Einfluß auf den Übersetzungsvorgang. Alle semantikrelevanten Eigenschaften der Knoten müssen durch die Compilerkanten explizit gemacht werden, da jeder ASG-Knoten identisch übersetzt wird. Dies erfordert eine Vielzahl weiterer Kanten, die bisher nicht erforderlich waren. *Die einzelnen Compilerkanten repräsentieren die einzelnen Programmfragmente des aus ihnen zusammenzusetzenden Suchprogrammes.* Eine Kante *für* einen Knoten e ist eine Kante k, für die $second(k) = e$ gilt ($first(k)$ ist der *Kantentyp*).[3] Wird von der *Implementierung* einer Compilerkante geredet, so ist damit das dieser Kante zugeordnete Programmfragment gemeint.

2. **Erzeuge alle erlaubten Pläne des ASG.**
   Ein Plan ist eine Permutation der Knotenmenge E des ASG. Eine Permutation von E ist eine bijektive Abbildung $\pi : 1 \ldots |\mathsf{E}| \rightarrow \mathsf{E}$.

3. **Wähle den besten erlaubten Plan und nenne ihn π.**
   Dies ist die Aufgabe des Optimierers.

4. **Erzeuge für jeden Knoten des ASG die sog. Bindungsfunktion.**
   Für i von 1 bis $|\mathsf{E}|$ und $e = \pi(i)$, führe aus:

   (a) Erzeuge die sog. Bindungsfunktion ϕ_e für e. Diese geht anhand des "Erzeuge und Prüfe"-Verfahrens (Generate & Test) vor. Die Bindungsfunktion ϕ_e hat folgende Struktur:

   `i. Solange es noch Kandidaten` e' `für` e `gibt, führe aus:`

   Generator: `Liefere einen erfolgversprechenden Kandidaten` e' `für` e`, so daß Hoffnung besteht, daß` $\phi(e) = e'$ `werden könnte.`
   Hierbei handelt es sich um den "Erzeuge"-Schritt – dieser Schritt wird durch die Implementierung einer speziellen Compilerkante für e – der sog. Generatorkante – implementiert.

   Tester: `Der ''Prüfe''-Schritt prüft nun alle` ***zu diesem Zeitpunkt*** `prüfbaren Bedingungen, also Eigenschaften und Relationen zu anderen Objekten (s.u.).`
   Dieser Test wird durch die Konjunktion der Tester-Implementierungen der einzelnen Compilerkanten für e implementiert.
   `Fällt der (ausgeführte) Test positiv aus, so`

   - prüfe, ob $i = |\mathsf{E}|$ ist:
     Ja: `melde ''Erfolg!''`
     Nein: `rufe die Bindungsfunktion des nächsten zu überprüfenden Knotens` $\pi(i+1)$ `auf,` $\phi_{\pi(i+1)}$

   `ii. An dieser Stelle gibt es keine Kandidaten` e' `für` e `mehr: die Bindungsfunktion gibt daher die Kontrolle an die aufrufende Funktion zurück.`

5. **Entferne die in Schritt 1 eingetragenen Compilerkanten.**

Durch die beiden Schriftarten kann in obigem Algorithmus zwischen der `Struktur des erzeugten Suchprogrammes` und der Struktur des Compilers selbst unterschieden werden.

[3] $first((x_1, x_2, x_3, \ldots, x_n)) = x_1$, $second((x_1, x_2, x_3, \ldots, x_n)) = x_2$, ..., $nth((x_1, x_2, x_3, \ldots, x_n)) = x_n$

Zur Ausführung des Suchprogrammes muß dann lediglich die Bindungsfunktion des Knotens $\pi(1)$ aufgerufen werden, $\phi_{\pi(1)}$. Liefert jeder Generator höchstens endlich viele Kandidaten und ist die Eckenmenge E endlich, so terminiert daß Verfahren natürlich – derartige Anfragen werden auch „sicher“ genannt. Die in Kap. 5 vorgestellte Sprache VISCO wurde hier so eingeschränkt, daß Anfragen stets sicher sind und somit terminieren.

Laut Winston ([Win93, S. 49]) sollte ein „guter“ Generator folgende drei Eigenschaften erfüllen:

- **Vollständigkeit:** alle möglichen Lösungen sollten gefunden werden.
- **Redundanzfreiheit:** keine Lösung sollte mehr als einmal gefunden werden.
- **Informiertheit:** je informierter eine Generator ist, desto stärker wird der Suchraum eingeschränkt und desto erfolgversprechender sind die Kandidaten, die er liefert.

Während die ersten beiden Punkte für alle hier vorgestellten Generatoren gelten, wird der letzte Punkt – also der Grad der Informiertheit eines Generators – ein wesentliches Merkmal für die Heuristiken des Optimierers sein: hierin unterscheiden sich die einzelnen Generatoren beträchtlich. Offensichtlich ist z.B. ein *hat_endpunkt*-Generator um ein vielfaches informierter als ein *schneidet_nicht*-Generator.

Ein *Plan* π repräsentiert nun also die *Reihenfolge, in der die Bindungen für die einzelnen Objekte gesucht werden.* Liegt zwischen zwei Objekten a und b eine Kante (typ, a, b) vor (die Kante zeigt also auf a, womit es sich um a's Kante handelt), so kann diese a-Kante in der $\phi(a)$-Bindungsfunktion nur dann geprüft werden, wenn b bereits vorliegt: käme nun jedoch a vor b im Plan π vor, so könnte diese Kante in der Bindungsfunktion von a ($\phi(a)$) nicht geprüft werden – wäre jedoch auch die *inverse Kante* $(inverse(typ), b, a)$ im ASG vorhanden, so könnte eine der beiden Kanten *ignoriert* werden, wodurch sowohl der Plan a, b als auch der Plan b, a möglich würden. Ein *Generator* für die Kante (typ, a, b) für Knoten a geht dann so vor, daß er anhand der Bindung von b ($\phi(b)$) einen Kandidaten für a ermittelte. Auch ein Tester für diese Kante könnte nur dann ausgewertet werden, wenn b bereits gebunden ist. Schließlich muß aber auch an die *Hyperkanten* gedacht werden: die Kante $(erzeuge\_schnittpunkt, p, s_1, s_2)$ hat keine Inverse, und somit wären nur die Reihenfolgen s_1, s_2, p und s_2, s_1, p möglich.

In diesem Sinne wird auch von *aktivierbaren Kanten* gesprochen: im oben skizzierten Algorithmus müssen die im Generator und im Tester verwendeten Kanten stets aktivierbar sein. Eine Kante $(typ, a_1, a_2, \ldots, a_n)$ ist also aktivierbar, wenn im Plan π $a_2, \ldots, a_n$ (in beliebiger Reihenfolge) vor a_1 auftauchen. Die Kanten repräsentieren also komplexe Abhängigkeiten. Eine *Eigenschaft* $(n = 1)$ ist immer aktivierbar.

Für jeden Knoten muß mindestens ein Generator (in Form einer aktivierbaren Generatorkante) vorhanden sein. Eine Ausnahme gibt es lediglich für Gebiete, da ein Gebiet

- entweder ein *konstantes Gebiet* ist und somit für sich selbst steht, oder aber
- durch sein *Argument bestimmt wird* (also z.B. das Polygon, dessen Inneres es denotiert) und somit der Generator des Argumentes bereits ausreicht. Hier besteht also eine Abhängigkeit vom Argument. Ein Gebiet kann jedoch zusätzliche Bedingungen auferlegen, so daß der „Tester“-Schritt notwendig ist.

Für Gebiete entfällt somit der „Erzeuge“-Schritt – für alle anderen Objekte wird der „Erzeuge“-Schritt jedoch ein Objekt des räumlichen Datenbestandes zurückliefern, sofern das VISCO-Objekt den Status *DB* oder *DB_C* hat. Der Generator wird nach und nach einen endlichen Strom von Kandidaten liefern, weil der Datenbestand endlich ist. Für *DB* und *DB_C* Objekte gibt es stets mindestens einen *immer* aktivierbaren Generator, nämlich den *maximal uninformierten Generator,* der den gesamten räumlichen Datenbestand aufzählt – nach Möglichkeit sollte dieser daher vermieden werden.

Handelt es sich hingegen um ein *U*-Objekt (also ein Hilfsobjekt), so gibt es in der Regel zunächst eine unendliche (und sogar überabzählbare) Anzahl von Kandidaten im Universum geometrischer Objekte (s. Kap. 5). Ein solcher nicht-abreißender Strom von Kandidaten würde dazu führen, daß das Verfahren nicht terminiert. Daher wird an *U*-Objekte nun die zusätzliche Bedingung gestellt, daß sie *direkt instantiierbar* sein müssen: dies bedeutet, daß ein solches Objekt entweder „Bottom-Up“ unter Verwendung seiner Komponentenobjekte *konstruiert* oder aber z.B. durch eine Operatoranwendung *unmittelbar berechnet werden kann.* Im Gegensatz zu den *DB*- und *DB_C*-Objekten liefert der Generator hier somit *höchstens* einen Kandidaten (pro Aufruf der entsp. ϕ-Funktion). Daher ist klar, daß das Verfahren wieder terminieren muß – somit ist die hier implementierte Teilmenge von VISCO entscheidbar. In der formalen Sprachbeschreibung im Anhang sind diese zusätzlichen Einschränkungen explizit ausgewiesen: hier findet man im Axiom *visco_universe_object* unter „Limitations“ die Forderung *direct_instantiable*.

Diese Einschränkung impliziert natürlich, daß *bestimmte Anfragen nicht mehr gestellt werden können:* z.B. ist eine *U*-Murmel nicht zulässig (und auch nicht sinnvoll, s. Kap. 5), solange sie nicht durch einen Operator berechnet und somit *direkt instantiiert* werden kann. Ein *U*-Nagel ist hingegen *stets* direkt instantiierbar, da ja seine Koordinaten (in Bezug auf die Folie) bekannt sind. Er kann daher konstruiert werden: offensichtlich muß jedoch das Koordinatensystem der Folie zuvor instantiiert worden sein, so daß hier eine weitere komplexe Abhängigkeit besteht. Im Gegensatz zu einer *U*-Murmel kann ein *U*-Nagel sinnvoll zum Konstruieren weiterer Einschränkungen verwendet werden (s. auch Bsp. 1 in Kap. 7).

Nun soll jeder der fünf oben vorgestellten Schritte des Compilers detaillierter dargestellt werden:

Schritt 1: Trage in den ASG eine Reihe von speziellen Compilerkanten ein. Da die Bindungsfunktion ϕ eines Knotens bzw. VISCO-Objektes ausschließlich anhand der Compilerkanten bestimmt wird, ist es notwendig, daß *alle semantikrelevanten Aspekte durch diese explizit gemacht werden.*

Im folgenden wird die spezielle Teilmenge der Compilerkanten der Kantenmenge K des ASG als CK bezeichnet ($\mathsf{CK} \subset \mathsf{K}$).

Wesentlich ist, daß mit jeder Compilerkante eine Reihe von Funktionen assoziiert sind:

- *ignorierbar* : $\mathsf{CK} \times \lambda \rightarrow \mathbb{B}$ – gibt diese Funktion $\top$ (wahr) zurück, so kann die entsp. Kante ignoriert werden (der zweite Parameter dient zur Übergabe von π).
- *generator_implementierung* : $\mathsf{CK} \rightarrow \lambda$ - diese Funktion liefert ein Funktionsobjekt, welches den Generator selbst implementiert. Die leere Funktion wird gleich $\bot$ (falsch) gesetzt.
- *tester_implementierung* : $\mathsf{CK} \rightarrow \lambda$ - diese Funktion liefert ein Funktionsobjekt, welches den Tester selbst implementiert. Die leere Funktion wird gleich $\bot$ (falsch) gesetzt.

Während die Funktionen *generator_implementierung* und *tester_implementierung* für alle Kanten des selben Typs die selbe Struktur haben, kann die Funktion *ignorierbar* von Kanteninstanz zu Kanteninstanz variieren. Folgende Punkte müssen beobachtet werden:

- Für einen Compilerkantentyp kann es sowohl eine Tester- als auch eine Generatorimplementierung geben.
- Eine Compilerkante, für die keine Generatorimplementierung existiert, kann nicht als Generatorkante verwendet werden, sondern lediglich als Testerkante.
- Eine Compilerkante, für die keine Testerimplementierung existiert, kann nicht als Testerkante verwendet werden, sondern lediglich als Generatorkante.
- Jede Compilerkante muß zumindest eine Generator- oder Testerimplementierung anbieten.

- Eine Kante, die sowohl Generator- als auch Testerimplementierung anbietet, kann natürlich breiter verwendet werden als eine Kante, die dies nicht tut.

Im Sinne eines *datengesteuerten Vorgehens* wird der Compiler zur Übersetzung des Knotens e nun alle *aktivierbaren Kanten (s.u.)* diese Knotens ermitteln, eine *Generatorkante* unter diesen auszeichnen (die restlichen ermittelten Kanten sind dann *Testerkanten*) und schließlich ein Programm erzeugen (die *Bindungsfunktion* ϕ_e), welches anhand der *Generatorimplementierung der ausgewählten Generatorkante* einen Kandidaten e' betrachtet und diesen durch eine *Konjunktion der Testerimplementierungen der Testerkanten* überprüft. Besteht der Kandidat e' den Test, so ist er *tatsächlich* Kandidat für e, und eine Bindung $\phi(e) = e'$ wurde gefunden. Alsdann wird die Bindungsfunktion des nächsten Knotens aufgerufen, sofern es sich nicht um den letzten Knoten handelt – dann wäre nämlich ein *Anfrageergebnis* zu melden.

Die einheitliche datengesteuerte Behandlung von Eigenschaften, Relationen und Operatoren begünstigt eine vollständige Optimierbarkeit aller Aspekte – u.a. ist diese Gleichbehandlung Voraussetzung dafür, daß verschiedene Kosten (von Objekte, Plänen, etc.) miteinander verglichen werden können.

Die Menge der Compilerkantentypen $\mathsf{C\_Kantentypen}$ ist ein Menge von Symbolen (die mit „e“ beginnenden Compilerkanten sind Eigenschaften):

$$
\begin{aligned}
&\mathsf{C\_Kantentypen} = \\
&\quad \{\, c\_e\_ist\_db\_object,\ c\_e\_ist\_primärobjekt,\ c\_e\_hat\_typ, \\
&\quad\ \ c\_e\_hat\_relationen,\ c\_e\_hat\_semantik, \\
&\quad\ \ c\_e\_hat\_höchstens\_n\_segmente,\ c\_e\_hat\_mindestens\_n\_segmente, \\
&\quad\ \ \vdots \\
&\quad\ \ c\_e\_hat\_länge,\ c\_e\_hat\_länge^{-1}, \\
&\quad\ \ c\_hat\_position,\ c\_hat\_position^{-1}, \\
&\quad\ \ c\_hat\_orientierung,\ c\_hat\_orientierung^{-1}, \\
&\quad\ \ \vdots \\
&\quad\ \ c\_hat\_endpunkt,\ c\_endpunkt\_von, \\
&\quad\ \ c\_hat\_segment,\ c\_segment\_von, \\
&\quad\ \ \vdots \\
&\quad\ \ c\_enthält,\ c\_enthalten\_in, \\
&\quad\ \ c\_schneidet,\ c\_schneidet\_nicht, \\
&\quad\ \ c\_erlaubte\_winkelabweichung\_ist, \\
&\quad\ \ c\_mittelpunkt\_von,\ c\_hat\_mittelpunkt\}
\end{aligned}
$$

Es gilt: $\mathsf{CK} = \{k \mid k \in \mathsf{K} \wedge \mathit{first}(k) \in \mathsf{C\_Kantentypen}\}$.

Compilerkanten werden in drei Gruppen nach Stelligkeit eingeordnet:

- **Unäre Kanten oder Eigenschaften (Properties),**
- **Binäre Kanten oder Abhängigkeiten (Dependencies),**
- **Hyperkanten oder Mehrfachabhängigkeiten (Multidependencies).**

Wiederum sind etwaige Parameter dieser Kanten als Funktionen auf den entsp. n-Tupeln implementiert (s.o.). Jeder C-Kantentyp ist als eigene CLOS-Klasse implementiert – die Instanzen dieser Klassen repräsentieren einzelne Kanten. Zur Definition von C-Kanten wurde eine eigene Schreibweise eingeführt, die in einem Schritt die Definition der Klasse, der entsp. Attribute und der Methoden *tester_implementierung* und *generator_implementierung* durchführt. Die Funktion *ignorierbar* muß hingegen (bei Bedarf) individuell für einzelne Kanteninstanzen als (anonymes) Funktionsobjekt definiert werden (als Wert eines Slots).

An dieser Stelle sollen nur drei Beispiele aus der großen Anzahl von C-Kantentypen vorgestellt werden:

```
(defproperty at-least-has-segments-is
  (:additional-slots at-least)
  (:tester `( (=> (typep candidate 'geom-chain-or-polygon)
                  (>= (length (segments candidate))
                      ,at-least))
              ,@(funcall continuation))))

(defdependency segment-of
  (:generator `( (dolist (candidate (segments (bound-to ,present)))
                   (with-binding (,candidate candidate)
                     ,@(funcall continuation)))))
  (:tester `( (member candidate (segments (bound-to ,present)))
              ,@(funcall continuation))))

(defdependency position-is
  (:generator-T `( (with-matrix (,(matrix present))
                     (let ((candidate
                             (get-point-from-spatial-index*
                               (x ,candidate) (y ,candidate))))
                       (with-binding (,candidate candidate)
                         ,@(funcall continuation))))))
  (:generator-NIL `( (with-matrix (,(matrix present))
                       (let ((candidate (p (x ,candidate) (y ,candidate)
                                           :affected-by-matrix-p nil)))
                         (with-binding (,candidate candidate)
                           ,@(funcall continuation))))))
  (:tester `( (with-matrix (,(matrix present))
                (and (=-eps (slot-value candidate 'x)
                            (x ,candidate))
                     (=-eps (slot-value candidate 'y)
                            (y ,candidate))))
              ,@(funcall continuation))))
```

Eine solche C-Kantentypdefinition enthält in der Regel eine Reihe von *Klauseln:* für eine *Abhängigkeit* (`defdependency`) sind folgende Klauseln vorgesehen:

- `:generator`
- `:tester`
- `:generator-T`, `:generator-NIL`
- `:generator-T->T`, `:generator-T->NIL`,
  `:generator-NIL->T`, `:generator-NIL->NIL`
- `:tester-T`, `:tester-NIL`
- `:tester-T->T`, `:tester-T->NIL`,
  `:tester-NIL->T`, `:tester-NIL->NIL`

Eine Abhängigkeit ist eine binäre Kante: sie verläuft stets zwischen den (lokal benannten) Objekten `candidate` und `present`. Die vorhandenen Bindungen dieser Objekte können durch die Funktion `bound-to` (hierbei handelt es sich also um die Implementierung von ϕ) ermittelt und durch die Umgebung `with-binding` etabliert werden.

Ein Tupel (*kantentyp*, *candidate*, *present*) wird in der obigen Schreibweise in der Form `candidate->present` notiert (die „Pfeilrichtung“ ist hier aus „historischen Gründen“ genau entgegengesetzt dargestellt, `->`). Anhand von `present` können nun Kandidaten für `candidate` geliefert (Generator) bzw. verifiziert werden (Tester). Die Bezeichnungen `T` und `NIL` in den Namen der Klauseln geben dabei jeweils an, ob `candidate` und `present` explizit im räumlichen Datenbestand vorliegen soll oder nicht (die einstelligen Klauselbezeichner erlauben für `present` beides – zusätzlich gilt dieses bei den nullstelligen Klauseln auch für `candidate`).

In Abhängigkeit dieser Wahrheitswerte müssen in der Regel unterschiedliche Implementierungen vorgesehen werden – die Funktionen *tester_implementierung* und *generator_implementierung* sind also so gestaltet, daß sie jeweils unterschiedliche Implementierungen zurückgeben. Während z.B. ein Kandidat für b bzgl. der Abhängigkeit $(enthält, a, b)$ für den Fall, daß sowohl a als auch b im Datenbestand explizit vorhanden sind (`T->T`-Kante) einfach durch die explizit im Datenbestand gespeicherte Kante zwischen a und b aufgefunden werden kann (s.o.), so erfordert der Fall, daß a und/oder b nicht im Datenbestand vorliegen, die Benutzung der *räumlichen Selektionsmakros,* `‘(with-selected-objects (candidate (bound-to ,present) :inside) ...)`.

Für *Eigenschaften und Mehrfachabhängigkeiten* sind lediglich folgende Klauseln vorgesehen:

- `:generator`
- `:tester`
- `:generator-T, :generator-NIL`
- `:tester-T, :tester-NIL`

Doch zurück zu den obigen Beispielen: Die Eigenschaft `at-least-has-segments-is` korrespondiert zur Eigenschaft *c_e_hat_mindestens_n_segmente* und wird bei Bedarf für Polygone, Ketten und Gummibänder angelegt. Diese C-Kante kann nicht als Generator (da hierfür kein Index vorgesehen ist), sondern lediglich als Tester verwendet werden.

Die Abhängigkeit `segment-of` korrespondiert zur *c_segment_von*-Compilerkante: sie wird für die Streckenkomponenten eines Polygones oder einer Kette erzeugt. Die Kante kann auch als Generator verwendet werden: in diesem Fall liefert sie sukzessive alle Segmente des bereits gebundenen Elternobjektes `present` als Kandidaten.

Als letztes Beispiel sei die Kante `position-is` (*c_hat_position*) diskutiert: hier werden *zwei* Generatoren implementiert. Wenn der entsp. Nagel im räumlichen Datenbestand vorhanden sein muß, wird ein Zugriff auf den räumlichen Datenbestand notwendig (`get-point-from-spatial-index*`). Ansonsten ist der Punkt nicht im Datenbestand zu finden – die `:generator-NIL`-Implementierung muß daher ein neues Punktobjekt *konstruieren*, was durch Anwendung des `(p x y)`-Konstruktors geschieht. Der Tester überprüft lediglich, ob die Koordinaten des gerade zu validierenden Punktes `candidate` auch mit den geforderten Koordinaten übereinstimmen.

Schritt 2: Erzeuge alle erlaubten Pläne des ASG Zunächst müssen die Begriffe „Plan“ und „erlaubter Plan“ definiert werden:

Definition (Plan und erlaubter Plan eines ASG $G = (\mathsf{E}, \mathsf{K})$): Ein *Plan* ist eine Permutation der Knotenmenge E des ASG. Eine Permutation von E ist eine bijektive Abbildung $\pi : 1 \ldots |\mathsf{E}| \rightarrow \mathsf{E}$. Die Inverse wird mit π^{-1} bezeichnet. Ein *erlaubter Plan* ist ein Plan, für den das Prädikat *erlaubter_plan* wahr wird: Die Menge der erlaubten Pläne des ASG $G = (\mathsf{E}, \mathsf{K})$ ist somit $\{\pi \mid \pi \in \lambda \wedge erlaubter\_plan(\pi, \mathsf{E}, \mathsf{K})\}$ (λ bezeichnet die Menge der Funktionen).

$$
\begin{array}{l}
\mathit{erlaubter\_plan}(\pi, \mathsf{E}, \mathsf{K}) \Leftrightarrow \\
\quad \mathit{bijektive\_funktion}(\pi) \wedge \\
\quad \mathit{domain}(\pi) = \{i \mid 1 \leq i \leq |\mathsf{E}|\} \wedge \\
\quad \mathit{range}(\pi) = \mathsf{E} \wedge \\
\quad \forall k \in \mathsf{CK} \\
\qquad (\mathit{aktivierbar}(k, \pi) \vee \mathit{verz\ddot{o}gerbar}(k, \pi) \vee \mathit{ignorierbar}(k, \pi)) \wedge \\
\quad \forall e \in \mathsf{E}\ (\mathit{aktivierbar}(e, \pi) \wedge \\
\qquad\quad ((e \in \mathsf{Gebiete} \wedge \\
\qquad\qquad \forall k \in \mathsf{CK}\ (\mathit{second}(k) = e \wedge \mathit{aktivierbar}(k, \pi) \Rightarrow \mathit{tester\_implementierung}(k)) \\
\qquad\qquad \vee \\
\qquad\qquad \exists\, k_1 \in \mathsf{CK} \\
\qquad\qquad\quad (\mathit{second}(k_1) = e \wedge \mathit{generator\_implementierung}(k_1) \wedge \mathit{aktivierbar}(k_1, \pi) \wedge \\
\qquad\qquad\quad \forall k_2 \in \mathsf{CK}, k_1 \neq k_2 \\
\qquad\qquad\quad (\mathit{second}(k_2) = e \wedge \mathit{aktivierbar}(k_2, \pi) \Rightarrow \mathit{tester\_implementierung}(k_2))))))
\end{array}
$$

Gibt die Funktion $\mathit{aktivierbar} : \mathsf{E} \times \lambda \rightarrow \mathbb{B}\ \top$ zurück, so ist der entsp. Knoten (im Plan π) aktivierbar. Hier müssen teilweise recht komplizierte zusätzliche Bedingungen gestellt werden: so kann z.B. ein Knoten einer Folie, die sowohl rotier- als auch in beide Richtungen beliebig skalierbar ist, nur dann aktiviert werden, wenn zuvor in π drei nicht-kollineare Nägel vorkommen (anderseits kann die Transformationsmatrix und somit das Koordintensystem der Folie nicht instantiiert werden).

Allgemein gilt, daß die Gültigkeit einer Kante $(\mathit{typ}, a_1, a_2, a_3, \ldots, a_n)$ für a_1 geprüft werden kann, wenn $a_2, a_3, \ldots, a_n$ bereits vorliegen: dies ist der Fall,wenn $a_2, a_3, \ldots, a_n$ vor a_1 in der Permutation π auftauchen, also früher gebunden werden. Eine solche Kante ist *aktivierbar*. Ausschließlich aktivierbare Kante können als Generator- oder Testerkanten verwendet werden (s.o.).

$$
\begin{array}{l}
\mathit{verz\ddot{o}gerbar}(k, \pi) \Leftrightarrow \\
\quad (\lambda(\mathit{kantentyp}, a_1, a_2, a_3, \ldots, a_n) \bullet \\
\qquad n = 2 \wedge \\
\qquad (\mathit{inverse}(\mathit{kantentyp}), a_2, a_1) \in \mathsf{CK} \wedge \\
\qquad \mathit{aktivierbar}((\mathit{inverse}(\mathit{kantentyp}), a_2, a_1), \pi) \\
\quad (\mathit{first}(k), \mathit{second}(k), \mathit{third}(k), \ldots, (n+1)\mathit{th}(k)))
\end{array}
$$

$$
\begin{array}{l}
\mathit{aktiverbar}(k, \pi) \Leftrightarrow \\
\quad (\lambda(\mathit{kantentyp}, a_1, a_2, a_3, \ldots, a_n) \bullet \\
\qquad \neg \mathit{ignorierbar}(k, \pi) \wedge \\
\qquad \pi^{-1}(a_2) < \pi^{-1}(a_1) \wedge \\
\qquad \pi^{-1}(a_3) < \pi^{-1}(a_1) \wedge \\
\qquad \vdots \\
\qquad \pi^{-1}(a_n) < \pi^{-1}(a_1) \\
\quad (\mathit{first}(k), \mathit{second}(k), \mathit{third}(k), \ldots, (n+1)\mathit{th}(k)))
\end{array}
$$

Binäre Kanten können *verzögert* werden, wenn es eine entsp. inverse Kante gibt: existieren Kanten (typ, a, b) und $(\mathit{inverse}(\mathit{typ}), b, a)$, so ist es egal, ob a vor b oder b vor a in π vorkommt, da die entsp. Bedingung in jedem Fall geprüft werden kann – die Bedingung ist ja sozusagen „doppelt“ repräsentiert. Hier wird deutlich, daß solche Doppelkanten bzw. Kanten und ihre Inversen eine wichtige Rolle spielen: erst durch die von ihnen repräsentierten „Wege“ werden mehrer Pläne möglich. Kanten und ihre Inversen unterscheiden sich zudem u.U. beträchtlich bzgl. ihrer Informiertheit (man denke z.B. an $(c\_\mathit{enth\ddot{a}lt}, a, b)$ und $(c\_\mathit{enthalten\_in}, b, a)$, wobei a ein sehr kleines Objekt sein soll).

Schritt 3: Wähle den besten erlaubten Plan und nenne ihn π. Nun ist noch zu klären, wie der Compiler in Schritt 3 den *besten erlaubten Plan* ermittelt – dies setzt natürlich die Ermittlung eines Bewertungsmaßes voraus, anhand dessen konkurrierende Pläne miteinander verglichen werden können. Zur folgenden Darstellung ist anzumerken, daß es sich hier ausschließlich um *empirisch ermittelte Heuristiken* handelt, die sich in der Praxis als wirkungsvoll erwiesen haben.

Als Basis der Bewertungsfunktionen dienen die Bewertungen der Informiertheit der einzelnen Compilerkanten. Dieses Wissen wird wiederum durch auf den Compilerkanten definierten Funktionen kodiert:

- $tester\_bewertung : \mathsf{CK} \rightarrow \mathbb{N}$
- $generator\_bewertung : \mathsf{CK} \rightarrow \mathbb{N}$

Je höher die von diesen Funktionen zurückgelieferten Werte sind, desto stärker schränkt die durch die Compilerkante repräsentierte Bedingung den Suchraum ein: so ist eine Kante *c_hat_position* besser zu bewerten als eine Kante *c_enthält*.

Definition (Bewertung eines Knotens e in einem Plan π): Die *Bewertung eines Knoten* e an der Stelle $\pi^{-1}(e)$ im Plan π ($knoten\_bewertung(e, \pi)$) ergibt sich folgendermaßen:

- Ermittle die Menge der aktivierbaren Testerkanten von e:
  $\mathsf{tester} = \{k \mid k \in \mathsf{CK} \wedge second(k) = e \wedge$
  $aktivierbar(k, \pi) \wedge tester\_implementierung(k)\}$
- Ist e kein Gebiet, so
  - Ermittle die Menge der aktivierbaren Generatorkanten von e:
    $\mathsf{generatoren} = \{k \mid k \in \mathsf{CK} \wedge second(k) = e \wedge$
    $aktivierbar(k, \pi) \wedge generator\_implementierung(k)\}$
  - Sortiere die Menge der aktiven Generatorkanten nach absteigender Güte:
    $\pi' : 1 \ldots |\mathsf{generatoren}| \rightarrow \mathsf{generatoren}$, so daß:
    $\forall i, j \in 1 \ldots |generatoren|$
    $(i < j \Rightarrow generator\_bewertung(\pi'(i)) \geq generator\_bewertung(\pi'(j)))$
  - Sei π' eine beliebige Bijektion, so daß
    $\pi' : 1 \ldots |\mathsf{tester} \setminus \{\pi'(1)\}| \rightarrow \mathsf{tester} \setminus \{\pi'(1)\}$
  - $knoten\_bewertung(e, \pi) =$
    $generator\_bewertung(\pi'(1)) \sum_{i=1}^{|\mathsf{tester} \setminus \{\pi'(1)\}|} tester\_bewertung(\pi'(i))$
- Ist e ein Gebiet, so
  - Sei π' eine beliebige Bijektion, so daß $\pi' : 1 \ldots |\mathsf{tester}| \rightarrow \mathsf{tester}$
  - $knoten\_bewertung(e, \pi) = \sum_{i=1}^{|\mathsf{tester}|} tester\_bewertung(\pi'(i))$

Es leuchtet ein, daß die Kante mit der besten *generator_bewertung* als Generatorkante ausgezeichnet und die einzelnen Testerkanten nach absteigender Güte geprüft werden sollten, so daß die Konjunktion so früh wie möglich scheitert. An dieser Stelle sollte man aber nicht nur die Stärke der durch die Kante repräsentierten Bedingung, sondern auch den Berechnungsaufwand zur Überprüfung der Bedingung berücksichtigen: die Bewertungsfunktionen stellen einen Kompromiß zwischen diesen beiden Aspekten da. Für jeden vom Generator gelieferten Kandidaten muß der gleiche Aufwand getrieben werden (in Form der Überprüfung der Testerkanten), und je mehr Kandidaten ein Generator liefert, desto höher ist dieser: so erklärt sich die besondere Betonung der Generatorkante in Form des Produktes zwischen der Generatorbewertung und der Summe der Testerbewertungen.

Die im Prototyp verwendeten Bewertungsmaße (also die Funktionen *generator_bewertung* und *tester_bewertung*) werden aufgrund eines einfachen Schemas ermittelt. In einer Art „Turnier“ werden alle Kanten paarweise miteinander verglichen – die *bessere Kante* bzw. der „Sieger der Begegnung“ bekommt jeweils einen „Punkt“:

Für jede Kante $k_1 \in \mathsf{CK}$ führe aus:

1. $generator\_bewertung(k_1) := 0$,
   $tester\_bewertung(k_1) := 0$.

2. Wenn $generator\_implementierung(k_1)$ gilt, dann

   - Für jede Kante $k_2 \in \mathsf{CK}, k_2 \neq k_1$ führe aus:

     (a) Wenn $generator\_implementierung(k_2) \wedge generator\_ist\_besser\_als(k_1, k_2)$ gilt, dann
     - $generator\_bewertung(k_1) := generator\_bewertung(k_1) + 1$

3. Wenn $tester\_implementierung(k_1)$ gilt, dann

   - Für jede Kante $k_2 \in \mathsf{CK}, k_2 \neq k_1$ führe aus:

     (a) Wenn $tester\_implementierung(k_2) \wedge tester\_ist\_besser\_als(k_1, k_2)$ gilt, dann
     - $tester\_bewertung(k_1) := tester\_bewertung(k_1) + 1$

Die Prädikate *tester_ist_besser_als* und *generator_ist_besser_als* werden durch einfaches Nachschlagen in einer Rangtabelle ermittelt:

- $tester\_ist\_besser\_als(k_1, k_2) \Leftrightarrow k_1$ erscheint vor k_2 in der Tabelle *tester_rang*
- $generator\_ist\_besser\_als(k_1, k_2) \Leftrightarrow k_1$ erscheint vor k_2 in der Tabelle *generator_rang*

Die beiden Tabellen bzw. Rangordnungen stellen relativ grobe (aber wirkungsvolle) Heuristiken dar und sollen hier nicht wiedergegeben werden.

Definition (Bewertung eines Planes π): Die *Bewertung eines Planes* π ergibt sich durch negativ-exponentielle Gewichtung der einzelnen Knotenbewertungen:

$$\sum_{i=1}^{|\mathsf{E}|} \frac{knoten\_bewertung(\pi(i), \pi)}{e^{(i-1)}}.$$

Die negativ-exponentielle Gewichtung bringt zum Ausdruck, daß auch die Knoten in einer Reihenfolge gebunden werden sollten, die den Verzweigungsfaktor der Suche möglichst schnell minimiert: je höher die Bewertung eines Knotens ist, desto früher sollte er im Plan erscheinen.

Schritt 4: Erzeuge für jeden Knoten des ASG die sog. Bindungsfunktion. Jetzt lassen die die beiden Schritte „Erzeuge“ und „Prüfe“ im obigen Algorithmus präzisieren: wiederum fehlt für Gebiete der Generator.

Erzeuge die sog. Bindungsfunktion ϕ_e für e. Diese geht anhand des “Erzeuge und Prüfe”-Verfahrens (Generate & Test) vor. Die Bindungsfunktion ϕ_e hat folgende Struktur:

1. Solange es noch Kandidaten e' für e gibt, führe aus:

   Generator: Der “Erzeuge”-Schritt hat folgende Struktur:

   (a) Ermittle die Menge der aktivierbaren Generatorkanten von e:
   $\mathsf{generatoren} = \{k \mid k \in \mathsf{CK} \wedge second(k) = e \wedge$
   $aktivierbar(k, \pi) \wedge generator\_implementierung(k)\}$

   (b) Sortiere die Menge der aktiven Generatorkanten nach absteigender Güte:
   $\pi' : 1 \ldots |\mathsf{generatoren}| \rightarrow \mathsf{generatoren}$, so daß:
   $\forall i, j \in 1 \ldots |generatoren|$
   $(i < j \Rightarrow generator\_bewertung(\pi'(i)) \geq generator\_bewertung(\pi'(j)))$

   (c) Sei $g = \pi'(1)$

(d) `Liefere unter Verwendung von` *generator_implementierung*(g) `einen (noch nicht gelieferten, also neuen) erfolgversprechenden Kandidaten` e' `für` e.

Tester: Der "Prüfe"-Schritt hat folgende Struktur:

(a) Ermittle die Menge der aktivierbaren Kanten von e ohne g (g wurde ja bereits verwendet):
$\mathsf{tester} = \{k \mid k \in \mathsf{CK} \wedge \mathit{second}(k) = e \wedge$
$\mathit{aktivierbar}(k, \pi) \wedge \mathit{tester\_implementierung}(k)\} \setminus \{g\}$

(b) Sortiere die Menge der aktiven Testerkanten nach absteigender Güte:
$\pi' : 1 \ldots |\mathsf{tester}| \rightarrow \mathsf{tester}$, so daß:
$\forall i, j \in 1 \ldots |\mathit{tester}|$
$(i < j \Rightarrow \mathit{tester\_bewertung}(\pi'(i)) \geq \mathit{tester\_bewertung}(\pi'(j)))$

(c) `Werte in dieser Reihenfolge für jede Kante die` *tester_implementierung* `aus, also` $\bigwedge_{i \in 1 \ldots |\mathit{tester}|} \mathit{tester\_implementierung}(\pi'(i))$
`Fällt der Test positiv aus, so`

- prüfe, ob $i = |\mathsf{E}|$ ist:
  Ja: `melde ''Erfolg!''`
  Nein: `rufe die Bindungsfunktion des nächsten zu überprüfenden Knotens` $\pi(i+1)$ `auf,` $\phi_{\pi(i+1)}$

2. `An dieser Stelle gibt es keine Kandidaten` e' `für` e `mehr: die Bindungsfunktion gibt daher die Kontrolle an die aufrufende Funktion zurück.`

Sowohl die Ermittlung der anwendbaren Kanten als auch die Sortierung dieser unter Verwendung der Bewertungsfunktionen sind Aktivitäten, die einmalig zur Übersetzungszeit der Bindungsfunktion anfallen. Die Struktur der Bindungsfunktionen der einzelnen Knoten ist somit fest – die Suchstrategie wird also ausschließlich zur Übersetzungszeit bestimmt und kann nicht aufgrund von sich zur Laufzeit ergebenden vorteilhaften Situationen dynamisch adaptiert werden.

Schritt 5: Entferne die in Schritt 1 eingetragenen Compilerkanten. Dieser Schritt bedarf keiner weiteren Erläuterungen.

Anmerkungen zur Implementierung des VISCO-Compilers

Es wurde deutlich, daß selbst die Übersetzung einer VISCO-Definition exponentiellen Aufwand erfordert: ein ASG mit $n = |\mathsf{E}|$ vorhandenen Knoten erlaubt im Extremfall $n!$ mögliche Pläne. Jeder dieser Pläne muß bewertet werden: in der Praxis ist die hier beschriebene Optimierung also nicht durchführbar, da eine VISCO-Definition sehr schnell 20 Objekte und mehr beinhaltet.

Tatsächlich ermittelt der implementierte Prototyp also nicht *alle möglichen Pläne,* sondern versucht, lediglich *einen* (den erstbesten) Plan zu konstruieren – zur Konstruktion dieses Planes wird jedoch die *knoten_bewertung* verwendet. Die Konstruktion eines legalen Planes gelingt in der Regel recht schnell, denn u.a. ist ja auch die Konstruktionsgeschichte (durch leichte Veränderung) ein legaler Plan.

In einer Art *Bestensuche (Best First)* wird dieser Plan nun sukzessive durch Hinzunahme eines neuen aktiven Nachfolgeknotens konstruiert: dabei wird unter den möglichen Nachfolgeknoten stets derjenige ausgewählt, der die beste Bewertung hat. Das Verfahren ist jedoch nicht vollständig (wie die Bestensuche auch), sondern bricht nach dem ersten gefundenen Plan ab. Dieser muß daher nicht optimal sein.

Um die Qualität der konstruierten Pläne zu erhöhen, wurde eine *weitere Heuristik* eingebaut, die Folien und Gebiete auch dann bevorzugt, wenn ihre *knoten_bewertungen* nicht den höchsten Wert unter allen aktiven (möglichen Nachfolge-) Knoten hat. Die Begründung hierfür ist, daß Folien

sobald als möglich im Plan erscheinen sollten, da dann die repräsentierte Geometrie einer VISCO-Anfrage geprüft werden kann (Folien haben ja ein Koordinatensystem). Gebiete werden deshalb bevorzugt, weil die *c_enthält*-Kante in der Regel eine sehr gute Generatorkante abgibt.

Nun sollen noch ein paar Bemerkungen zu den implementierten Folien gemacht werden: prinzipiell gibt es keine *DB*-Folien, da im räumlichen Datenbestand keine Aggregate vorliegen. Der Editor läßt nur die Erzeugung von *U*-Folien zu – also ist automatisch jede erzeugte Folie eine *U*-Folie (so erklärt sich auch das Fehlen eines entsp. Status-Schalters für Folien). Eine *U*-Folie wird somit durch Aggregation ihrer Komponentenobjekte instantiiert. Verschiedene Freiheiten der Transformationsparameter derartiger Folien (wie t_x, t_y, s_x, s_y, rot) erfordern nun einige *weitere Abhängigkeiten,* die durch die Funktion *aktivierbar* erfaßt werden müssen. Allgemein wird gefordert, daß *alle freien Transformationsparameter der Folientransformations-Matrizen eindeutig durch Punkte mit bekannter Position (Nägel) bestimmbar sein müssen.* Je mehr Freiheiten eine Folie für ihre Parameter zuläßt, desto mehr Nägel müssen vorhanden sein, damit die Transformationsmatrix anhand der bereits instantiierten Nägel eindeutig berechnet werden kann.

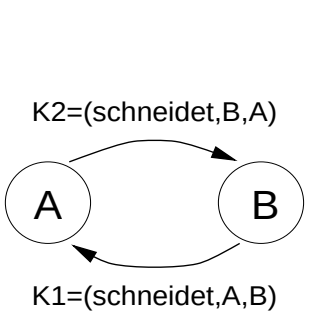
Diese zusätzlichen Abhängigkeiten der Folien von ihren Nägeln könnten durch entsp. *Hyperkanten (Multiabhängigkeiten)* repräsentiert werden, von denen jeweils eine aktivierbar sein müßte: dies würde jedoch einen wenig sinnvollen Aufwand an zusätzlichen Kanten erfordern, nämlich z.B. $(n\ 3)$ („n über 3“) 3er-Hyperkanten pro Folie, wenn n die Anzahl der Nägel dieser Folie wäre und drei Nägel zur Bestimmung der Transformationsmatrix benötigt würden, was im Fall einer beliebig skalier- und rotierbaren Folie zuträfe. Daher erscheint es sinnvoller, diese Abhängigkeiten durch die Funktion *aktivierbar* zu kodieren: wird nun versucht, *folie* als Nachfolgeknoten vorzusehen, so wird diese Funktion (bzw. der Aufruf *aktivierbar*(*folie*)) den bereits bis dahin konstruierten Plan π untersuchen und entscheiden, ob die bis dahin im Plan π vorhandenen bzw. zur Suchzeit instantiierten Nägel zur Bestimmung der Transformationsparameter von *folie* ausreichen. Ist dies nicht der Fall, so kann *folie* (noch) nicht an dieser Stelle im Plan erscheinen.

Es wird daher für jeden zur Berechnung der Transformationsparameter einer Folie mitverwendeten Nagel zusätzlicher spezieller Code in seiner Bindungsfunktion ϕ erzeugt. Per Seiteneffekt wird dann durch die Ausführung der Bindungsfunktion eines so verwendeten Nagels der entsp. Wert einem Folientransformationsparameter (wie z.B. t_x, t_y) im Folienknoten zugewiesen. Wird schließlich die Folienbindungsfunktion aufgerufen, so ist die Matrix der Folie bereits durch die vorgenommenen Zuweisungen per „Seiteneffekt“ (durch die Ausführungen der Bindungsfunktionen der verwendeten Nägel) eindeutig bestimmt.

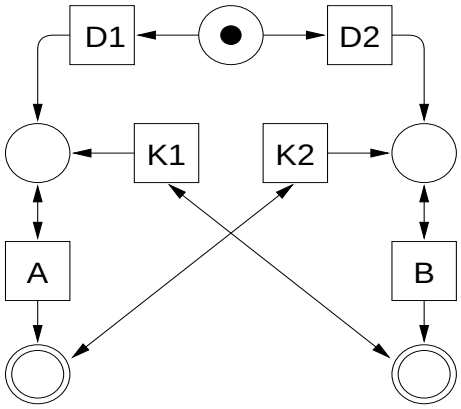
In der formalen Sprachbeschreibung im Anhang sind diese Beschränkungen des Prototypen im Axiom *visco_transparency* (und den Unterklassen) explizit ausgewiesen.

Die hier geführte Diskussion konnte nicht jedes Detail darstellen: z.B. konnte nicht jede Compilerkante (derer es recht viele gibt) ausführlich dargestellt werden. Auch die sog. *Effizienzkanten* konnten nicht diskutiert werden. Liegen in einem Plan π z.B. bereits die beiden Endpunkte einer im Datenbestand zu findenen Strecke s vor, so ist es sinnvoll, diese Strecke *direkt zu instantiieren.* Hierzu müssten lediglich die gemeinsamen Elternobjekte der Endpunkte ermittelt werden, was sehr einfach ist, da die Punkte ja bereits vorliegen (man denke an die Graphstruktur der Objekte). Somit könnten die beiden s-Kanten $(hat\_endpunkt, s, p_1)$ und $(hat\_endpunkt, s, p_2)$ ignoriert werden, (was durch die Prädikate $ignorierbar((hat\_endpunkt, s, p_1), \pi)$ und $ignorierbar((hat\_endpunkt, s, p_2), \pi)$ festgestellt würde), wenn man eine s-Multiabhängigkeit ($instantiiere\_strecke\_anhand\_endpunkte$, s, p_1, p_2) vorsehen würde. Es ist nicht sinnvoll, stets auf die *hat_endpunkt* Kanten zugunsten einer *instantiiere_strecke_anhand_endpunkte*-Kante zu verzichten, da dem Compiler ansonsten viele mögliche Pläne verloren gingen. Es gilt:

$$ignorierbar((hat\_endpunkt, s, p), \pi) \Leftrightarrow \\ \pi^{-1}(p_1(s)) < \pi^{-1}(s) \wedge \\ \pi^{-1}(p_2(s)) < \pi^{-1}(s)$$

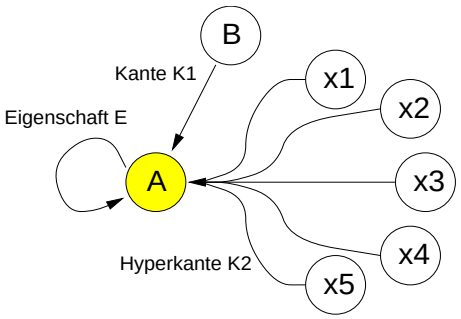


(a) ASG



(b) B/E-Petrinetz

Abbildung 6.15: ASG und Petrinetz



Funktionen auf Kanten / Hyperkanten:
- ignorierbar
- verzögerbar

Funktionen auf Knoten:
- aktivierbar

Abbildung 6.16: ASG für Knoten A

$$ignorierbar((instantiiere\_strecke\_anhand\_endpunkte, s, p_1, p_2), \pi) \Leftrightarrow \\ \neg(\pi^{-1}(p_1) < \pi^{-1}(s) \wedge \\ \pi^{-1}(p_2) < \pi^{-1}(s))$$

Schließlich soll noch erwähnt werden, daß ein erlaubter Plan eines ASG auch als *Prozeß eines B/E (Bedingungs/Ereignis)-Petrinetzes* dargestellt werden könnte (s. [JV87, S. 38], alle Stellen und Kanten haben eine Kapazität von 1). Die erlaubten Pläne des ASG aus Abb. 6.15(a) werden durch die Prozesse des Netzes ([JV87, S. 40]) aus Abb. 6.15(b) erzeugt. Die Transitionen *K1* und *K2* repräsentieren die entsp. Kanten des ASG: schalten sie, so erzeugen sie entsp. Programmcode für die Bindungsfunktion von A bzw. B, nämlich die *tester_implementierung* oder *generator_implementierung* (in [JV87, S. 53] werden derartig annotierte S/T-Netze *Netzprogramme* genannt). Da im dargestellten ASG ein Zyklus vorliegt, muß zunächst eine Kante *verzögert* werden: dies wird durch die entsp. Stelle und die beiden Transitionen *D1* und *D2* dargestellt („D“ steht für „defer“). Die beiden möglichen Prozesse sind nun $D1-A-K2-B$ und $D2-B-K1-A$. Nachdem A geschaltet hat, ist die Bindungsfunktion für Knoten A komplett. Die *terminalen Markierungen* sind dabei alle Markierungen, in denen die doppelt umrandeten Ausgangsstellen von A und B belegt sind.

Die durch die Funktionen *ignorierbar*, *aktivierbar* etc. implementierten komplexen Abhängigkeiten müßten durch Prädikate in den Transitionen des Netzes dargestellt werden – eine solche mit einem

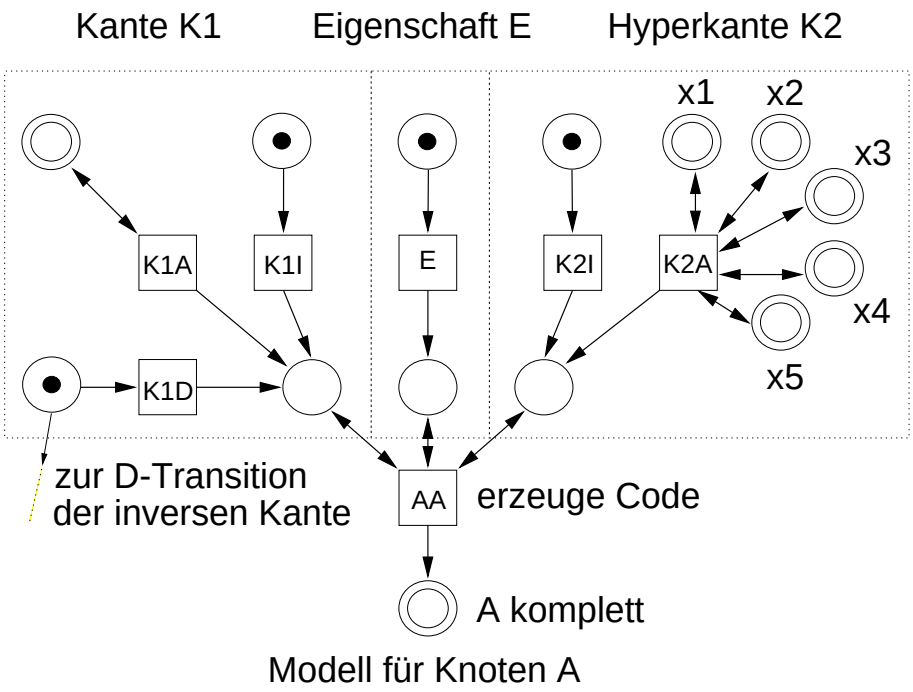


Abbildung 6.17: B/E-Petrinetz für Knoten A in Abb. 6.16

Prädikat x annotierte Transition x kann nur schalten, wenn zusätzlich zur verwendeten Schaltregel ([JV87, S. 39]) das Prädikat x wahr wird.

Auch Hyperkanten und Eigenschaften können behandelt werden: in Abb. 6.17 bezeichnet *AA* die Funktion *aktivierbar*(a), *K1I* steht für *ignorierbar*(k_1), *K1A* für *aktivierbar*(k_1), *K1D* für *verzögerbar*(k_1), etc. Offensichtlich kann jeder ASG systematisch in ein entsp. Netz transformiert werden. Die terminalen Markierungen sind all die Markierungen des Netzes, in denen alle doppelt umrandeten Ausgangsstellen der entsp. Transitionen belegt sind.

Ausführung & Ergebnispräsentation

Die Ausführung der aktuellen VISCO-Anfrage wird durch den Menüpunkt „Execute Query“ (im Grafikeditor) veranlaßt. Der Compiler wird gestartet und das erzeugte LISP-Suchprogramm ausgeführt. Der Benutzer holt nun die Ergebnispräsentationskomponente in den Vordergrund, um die aufgefundenen Konstellationen zu inspizieren, die Suche abbrechen zu können, etc.

Die implementierte Ergebnispräsentationskomponente ist in Abb. 6.18 dargestellt.[4] Wiederum sind drei große Teilfenster zu erkennen:

- Links oben findet sich das „VISCO Query Results“ Fenster zur *mosaikartigen Darstellung aller bisher aufgefundener Konstellationen.* Der Benutzer kann einzelne Kacheln selektieren (die zuletzt selektierte Kachel wird im „VISCO Inspect Query Result“-Fenster dargestellt), markieren (markierte Kacheln werden in Gelb dargestellt), markierte Kacheln löschen, Seiten umblättern, ganze Seiten löschen, etc. Hierfür sind eine Reihe von Tastern (Push Buttons) vorgesehen – u.a. findet sich dort auch ein Taster zum Abbrechen der momentanen Suche. Der *Fortschrittsanzeiger (Progress Bar)* in Form eines roten Balkens über dem „Abort Search“-

[4] Zur Darstellung ist zu bemerken, daß natürlich auch *Permutationen* der Komponentenbelegungen erkannt werden – somit erklärt sich das vielfache Auftreten von „ein und derselben“ Konstellation, denn die einzelnen Komponenten haben jeweils *unterschiedliche Bindungen.* Man könnte einen Filter einbauen, der Permutationen bereits aufgefundener Konstellationen aus dem Strom entfernt.

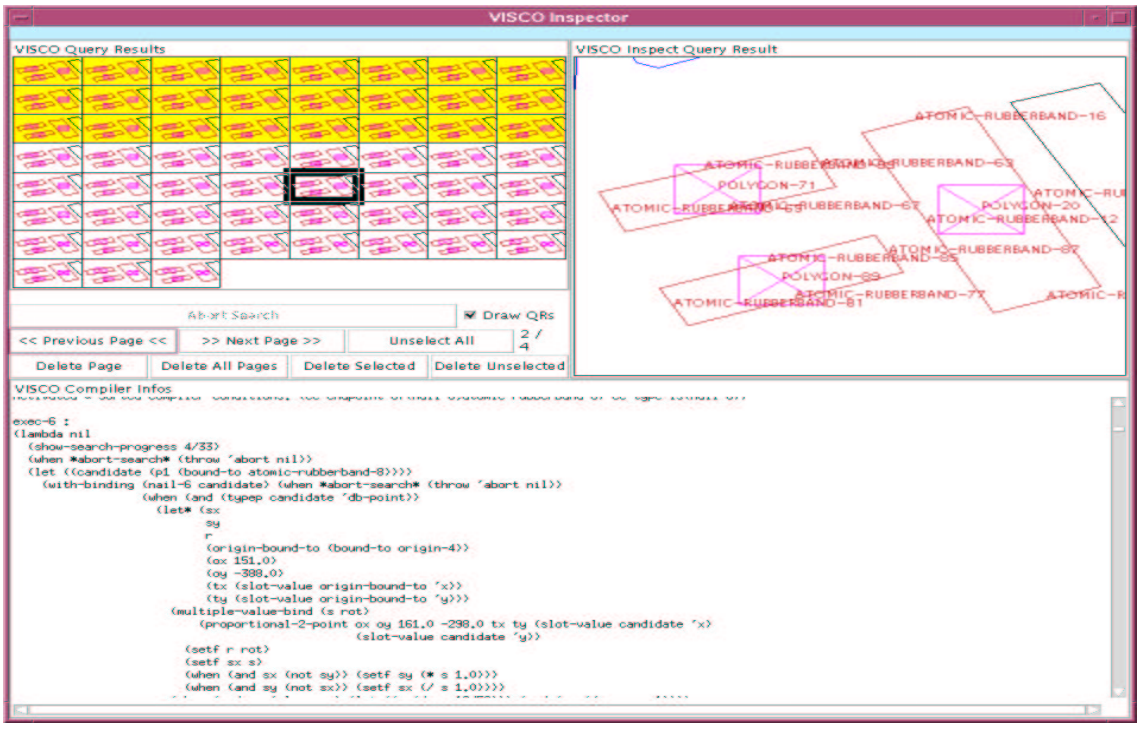


Abbildung 6.18: Inspektion der Anfrageergebnisse

Taster zeigt (im Viertelsekundentakt) die gerade aktive Bindungsfunktion an: Befindet sich die Kontrolle in der Bindungsfunktion von Knoten i von insgesamt n zu bindenen Knoten, so hat der Balken $\frac{i}{n}$ seiner vollen Länge.

- Das „VISCO Inspect Query Result“-Fenster (rechts oben) dient der *vergrößerten Darstellung der zuletzt selektierten Kachel*. Diese Kachel kann auch im Map Viewer betrachtet werden – holt der Benutzer den Map Viewer in den Vordergrund, so kann die volle Funktionalität des Map Viewer zur Inspektion genutzt werden. U.a. kann der Benutzer dann die Umgebung der Konstellation inspizieren, einzelne Objektschlüssel ein- und ausblenden, etc. (s.o.).

- Schließlich dient das „VISCO Compiler Infos“-Fenster zur Inspektion des gerade aktuellen LISP-Suchprogrammes. Hier wird vom Compiler der Programmtext eingeblendet.

Die oben angesprochenen Verifikations- und Übernahmemöglichkeiten selektierter Kacheln bzw. Konstellationen in den Grafikeditor konnte noch nicht realisiert werden.

Zur Implementation ist zu sagen, daß jedesmal, wenn eine komplette Konstellation gefunden wurde (wenn also im oben skizzierten Algorithmus der Bindungsfunktion `melde „Erfolg!\` auftaucht), die gerade aktuellen Bindungen der Knoten gesichert werden müssen. Zu diesem Zweck wird einfach eine CLOS-Instanz der speziell zu diesem Zweck entworfenen Klasse `query-result` erzeugt. Die entsp. Instanzen korrespondieren direkt zu den einzelnen Kacheln.

Die Ergebnispräsentationskomponente ist mit 15 kb LISP-Code die kleinste Komponente des Prototypen.

Kapitel 7

Benutzung des VISCO-Prototypen

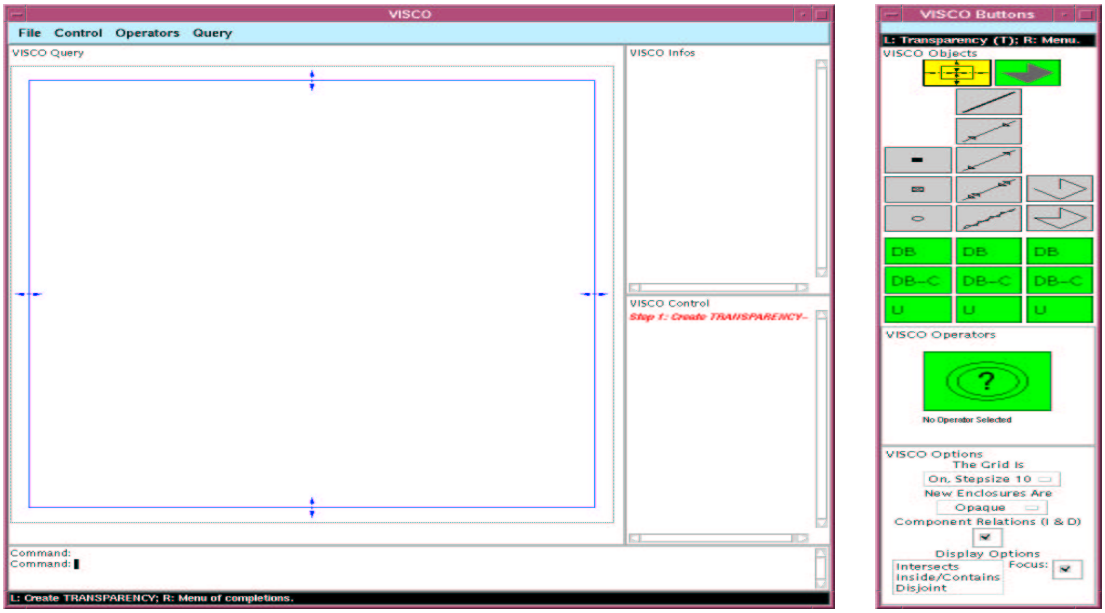
Im folgenden soll schrittweise anhand eines einfachen Beispieles die Interaktion mit dem VISCO-System unter Verwendung der einzelnen Komponenten Grafikeditor, Map Viewer und Ergebnispräsentationskomponente demonstriert werden. Nur die wesentlichsten Schritte der Interaktion werden hier wiedergegeben – dennoch kann dem Leser so ein Eindruck vermittelt werden.

Zu den Bearbeitungszeiten (Suchzeiten) der hier gestellten Anfragen ist zu sagen, daß sie alle unter zwei Minuten (auf einer Sparc Ultra I) liegen – dies ist jedoch meist nur die Zeit, die bis zum Auffinden der ersten Konstellationen vergeht. Oftmals wurde die Suche alsdann mit dem „Abort“-Taster abgebrochen.

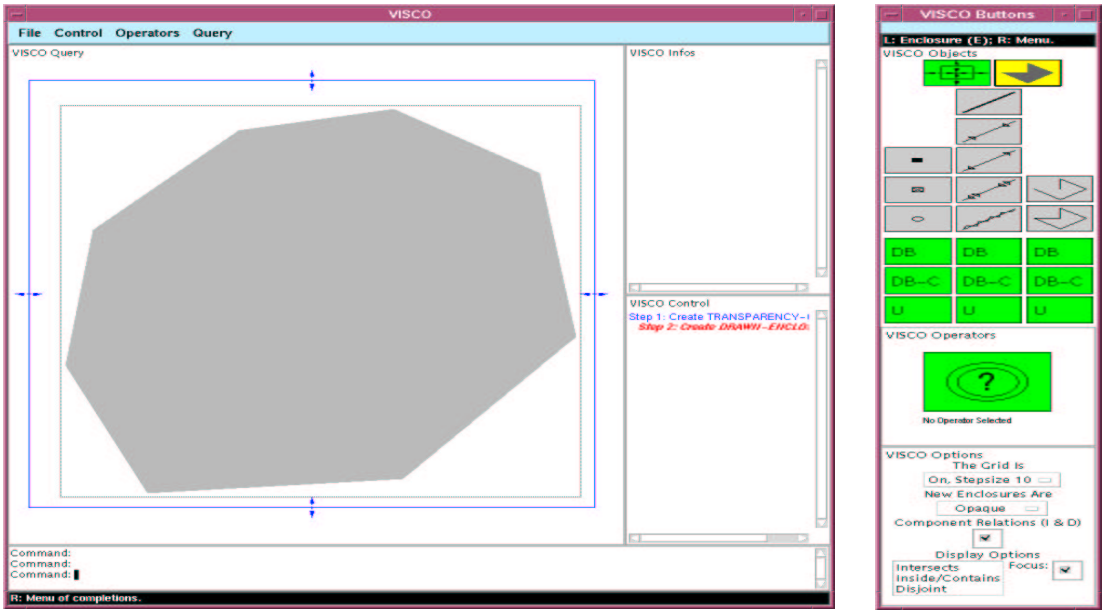
Desweiteren ist anzumerken, daß der Prototyp wesentlich mehr Sprachelemente und Operatoren implementiert, als in den folgenden Beispielen verwendet werden (s. Kap. 6). Berechnete ϵ-Gebiete, Schnitt- und Mittelpunkte können ebenfalls verwendet werden. Wird ein Objekt interaktiv verschoben, so werden die abhängigen Objekte automatisch neuberechnet (z.B. eine ϵ-Gebiet der veränderten Form eines Polygones angepaßt, ein Schnitt- oder Mittelpunkt neu berechnet, etc.). Der zugrundeliegende Mechanismus wurde ja in Kap. 6 erläutert. Die entsp. Abhängigkeiten werden im ASG-Repository verwaltet – einen Eindruck vom Mechanismus bekommt der Leser anhand des zweiten Beispieles.

Da dies kein Handbuch für die Benutzung des Prototypen ist, können die einzelnen Menüpunkte, Tastenbelegungen, Operatoren etc. hier nicht ausführlich dargestellt werden.

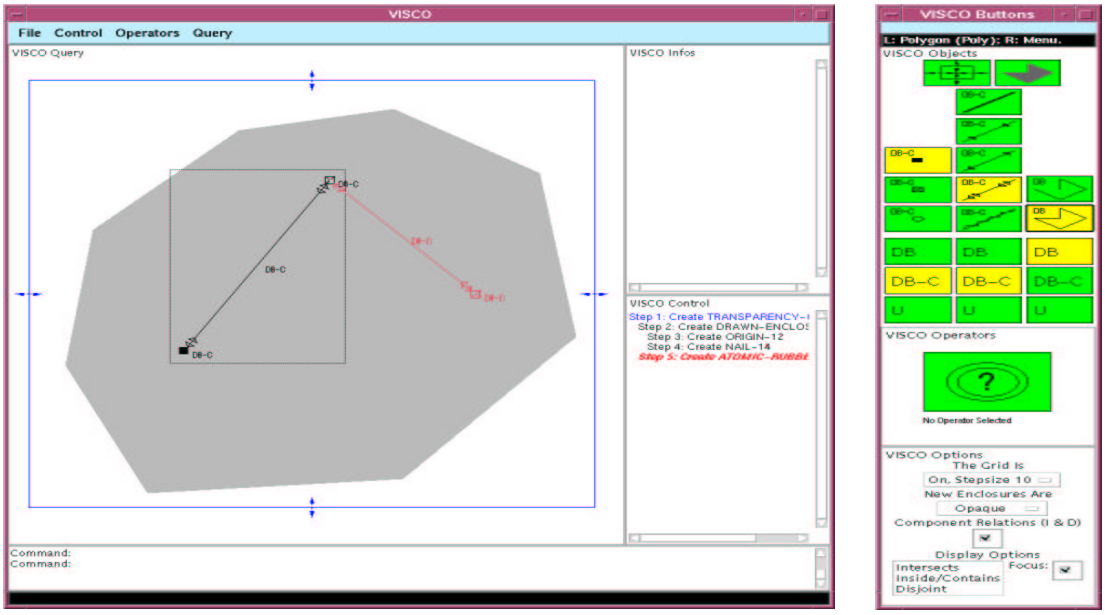
7.1 Beispiel 1: Visuelle räumliche Anfragen an die DISK



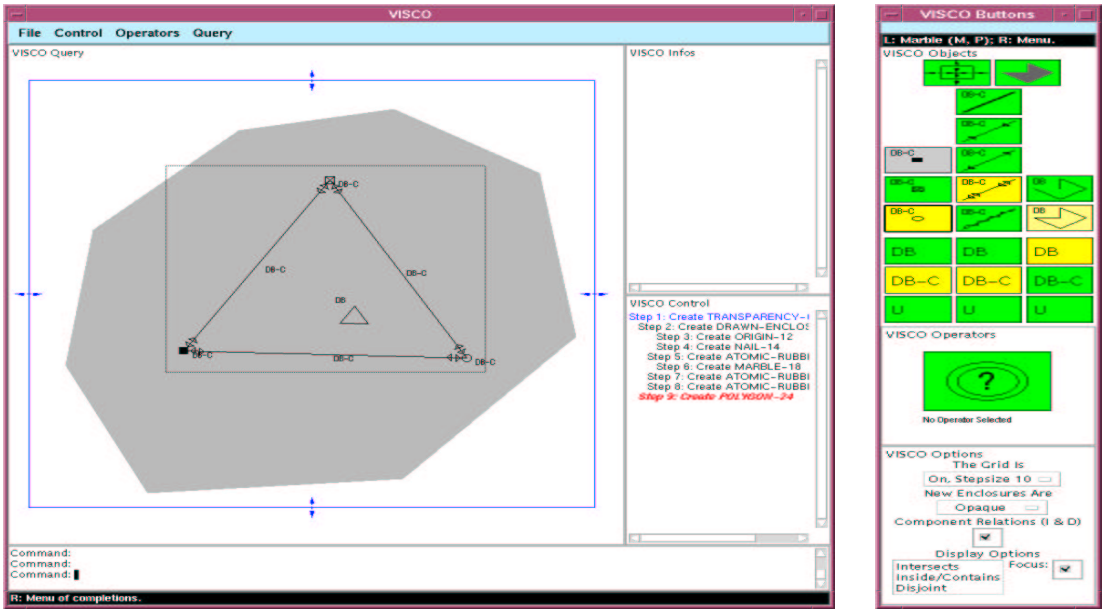
Schritt 1: Der Benutzer selektiert im Schalterfenster den Schalter für „Overheadprojektorfolie“ (er leuchtet nun gelb) und erzeugt interaktiv mit der Maus die Folie. Diese ist zunächst beliebig rotier-, translatier- und skalierbar (was durch die Pfeile an den Kanten visualisiert wird). Die Folie ist eine Hilfsfolie (s. Kap. 5), da der Prototyp keine D-Folien implementiert. Da der „Fokus“-Schalter an ist, wird das zuletzt erzeugte Objekt stets mit einem gestrichelten Rahmen versehen.



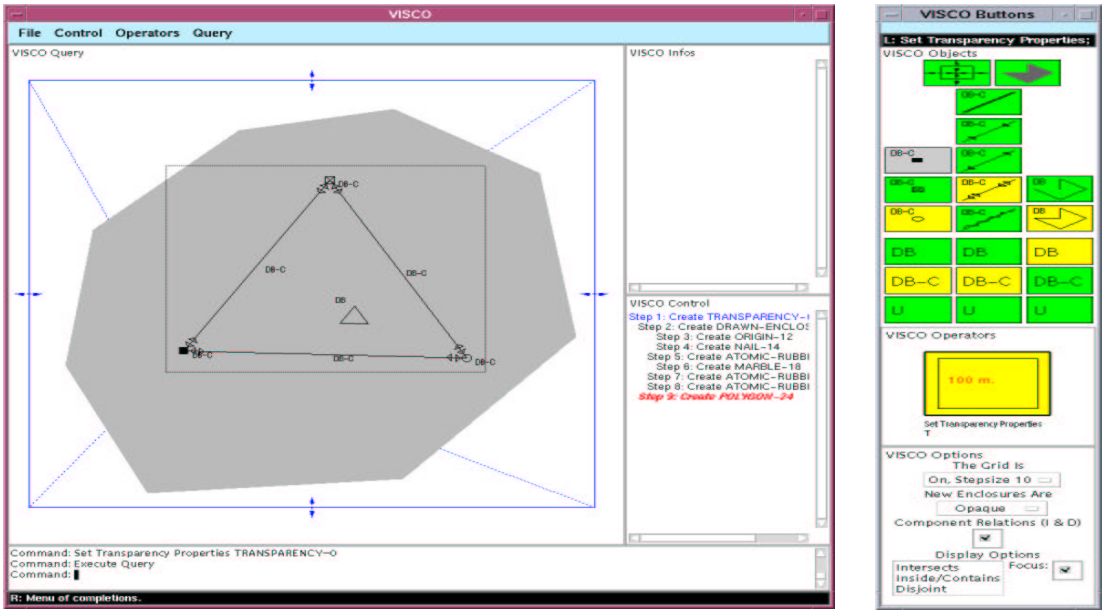
Schritt 2: Der Benutzer selektiert den Schalter für „Skizziertes (Konstantes) Gebiet“ und erzeugt interaktiv ein solches. Nun wird das Gebiet mit dem Fokus-Rechteck umrandet.



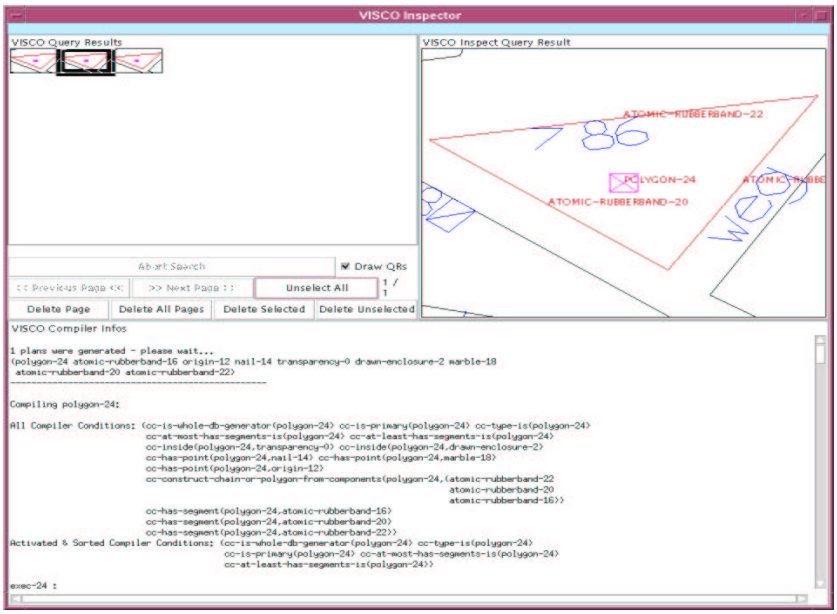
Schritt 3: Der Benutzer selektiert im Schalterfenster die Objekte Ursprung, Teleskopantenne und Polygon. Er bestimmt den Status dieser zu erzeugenden Objekte mit „DB-C DB-C DB“ – das Polygon soll somit im Datenbestand explizit vorhanden sein. Nachdem der erste Punkt des neues Polygones erzeugt ist, wird automatisch der Punkttyp im Schalterfenster von „Ursprung“ in „Nagel“ geändert, da eine Folie nur einen Ursprung haben kann. Ohne Änderungen des Zustandes im Schalterfenster wird der zweite erzeugte Punkt daher ein Nagel sein.



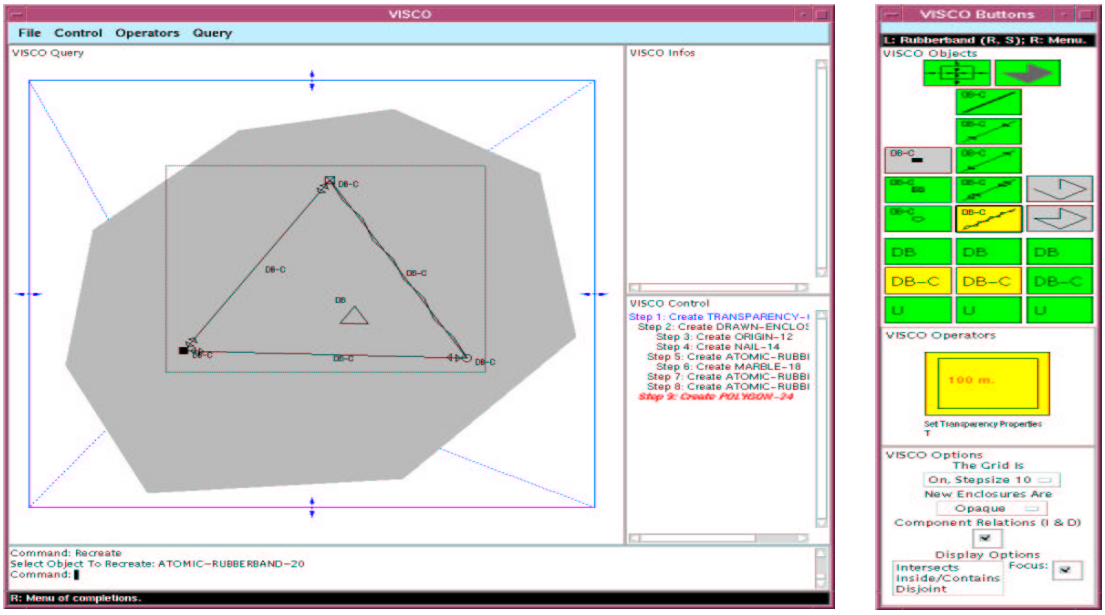
Schritt 4: Während der Benutzer noch das Polygon erzeugt, ändert er den Punkttyp im Schalterfenster für den dritten zu erzeugenden Punkt in „Murmel“. Nachdem die Murmel erzeugt ist, schließt der Benutzer das Polygon durch Selektion des ersten Punktes (des Ursprunges). Durch „Backspace“ hätte er bereits erzeugt Punkte löschen und durch „Cancel“ (Rechte Maustaste) die Polygonerzeugung abbrechen können.



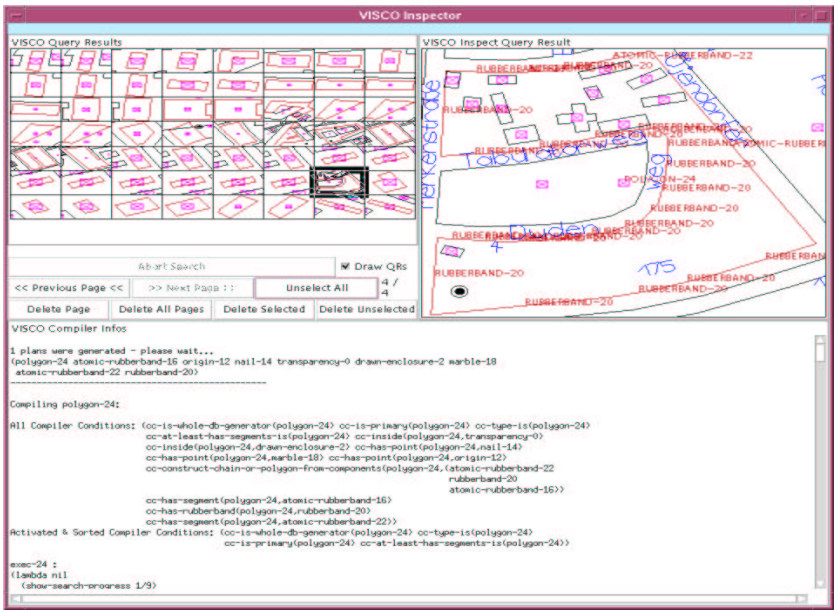
Schritt 5: Der Benutzer selektiert das momentan dargestellte Ikon, woraufhin die ikonische Operatorbibliothek erscheint (s. Abb. 6.10(a)). Aus dieser wählt er das „Bestimme Folieneigenschaften“-Ikon aus. Durch Selektion der Folie (eine best. Tastenkombination muß gedrückt werden) wird der so ausgewählte Operator nun auf die Folie angewendet (Präfixoperatoranwendung). Ein Menü erscheint, in dem die Eigenschaften der Folie textuell verändert werden können. Der Benutzer bestimmt, daß die Folie ausschließlich proportional skalierbar sein soll – dies wird durch die Führungslinien visualisiert.



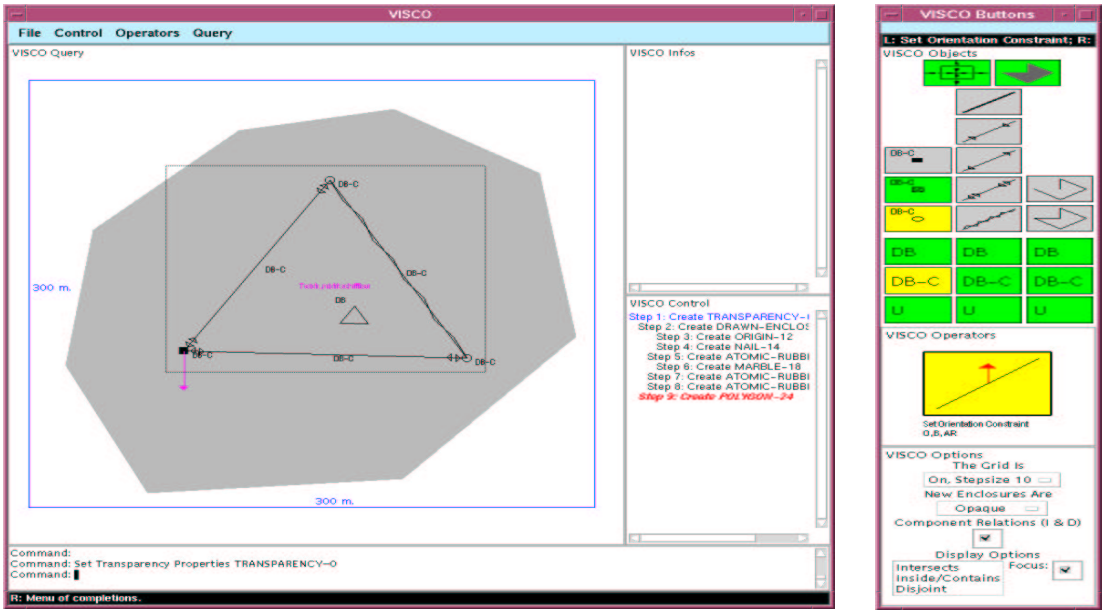
Schritt 6: Alsdann wird mit dem Menüpunkt „Query – Execute“ die Ausführung der so erzeugten Anfrage angestoßen – hier wird also nach expliziten Dreiecken im Datenbestand gesucht. Nach einer kurzen Wartezeit erscheinen drei Konstellationen im „Query Results“-Fenster, die alle das gleiche Dreieck darstellen (es gibt nur eines). Da es für jede Seite diese Dreieckes drei mögliche Belegungen gibt, erscheint das Dreieck dreifach.



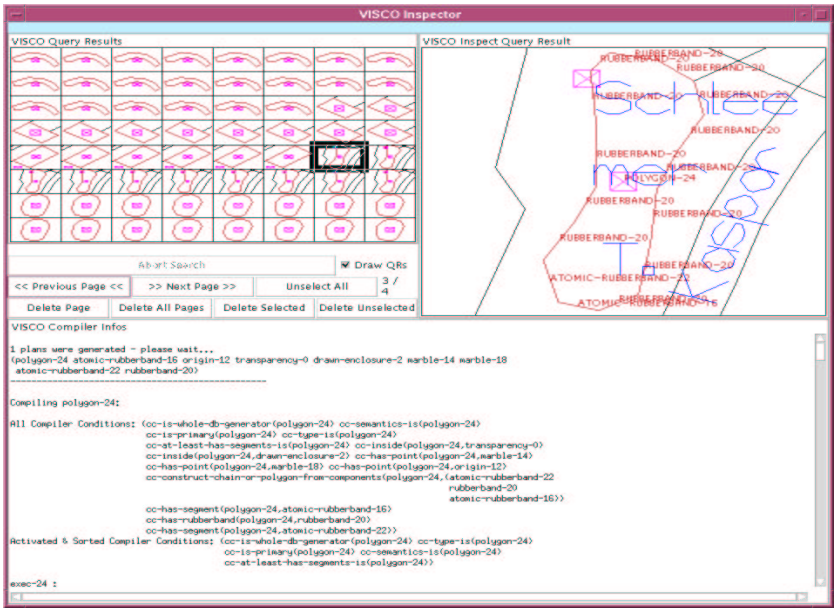
Schritt 7: Nun kommt dem Benutzer die Idee, eine der Teleskopantennen des Dreieckes durch ein Gummiband zu ersetzen. Hierzu selektiert er im Schalterfenster den entsp. Schalter und wählt dann den Menüpunkt „Ersetze“ („Recreate“). Dieser Menüpunkt erlaubt es, einzelne Objekte (und Komponenten) auszutauschen bzw. einzelne Attribute dieser zu verändern. Der Benutzer selektiert die Teleskopantenne (rechts oben im Dreieck), die alsdann gegen ein Gummiband ausgetauscht wird. Die Attribute für das Gummiband werden dem Schalterfenster entnommen, denn dort wird ja der *Zustand* dargestellt.

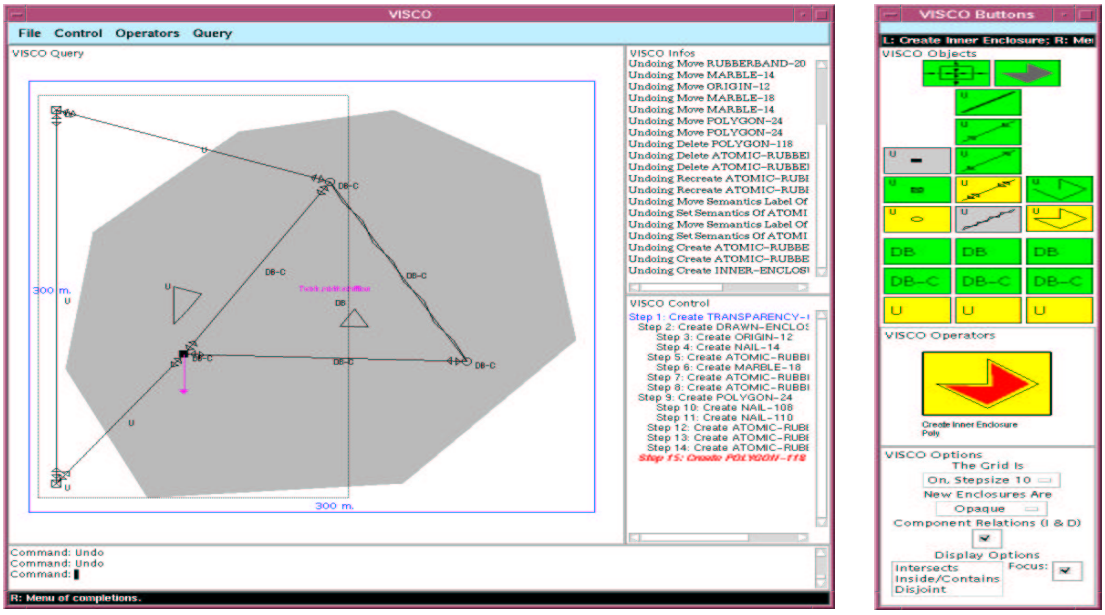


Schritt 8: Die erneute Ausführung der so veränderten Anfrage ergibt nun eine überwältigende Anzahl passender Konstellationen, da durch diese Definition die Klasse aller Polygone mit mindestens drei Seiten beschrieben wird. Der Benutzer bricht die Suche daher mit dem „Abort“-Taster ab.

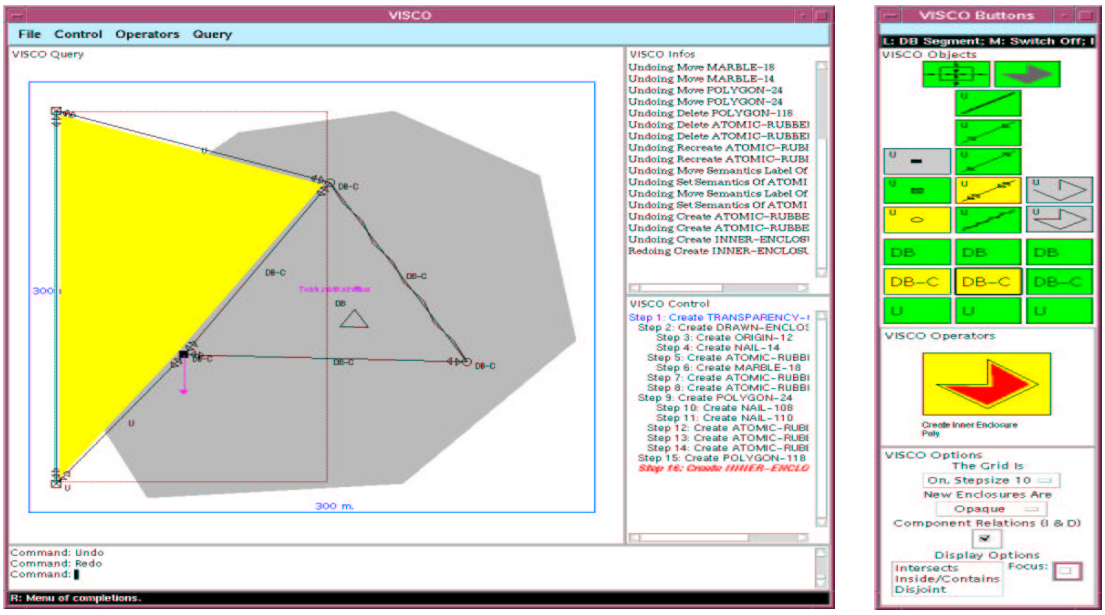


Schritt 9 und 10: Nun sucht der Benutzer nach einem nicht zu großen Teich. Hierzu weist er dem Dreieck zunächst die Thematik „Teich, nicht schiffbar“ zu. Da der Teich nicht zu groß sein soll, werden die Abmessungen der Folie auf 300 mal 300 Meter festgelegt. Offensichtlich ist diese Anfrage zu streng, da die eine Seite des Teiches nun eine genau festgelegte Länge hat (die Antenne zwischen Ursprung und Nagel). Der Nagel wird deshalb gegen eine Murmel ersetzt. Derartige Anfragen (mit nur einem Punkt mit bekannter relativer Position) kann der Prototyp nur beantworten, wenn die Orientierung der Folie ebenfalls festgelegt wird (zur eindeutigen Berechnung der Folien-Transformationsmatrix mit mehreren Freiheitsgraden würden weitere Fixpunkte benötigt, s. Kap. 6). Nach erneuter Ausführung der so stark veränderten Anfrage werden eine große Anzahl von Teichen gefunden – wiederum erscheinen Permutationen „identischer“ Teiche. Natürlich gibt es für einen Teich mit n Segmenten n zyklische Belegungen der Seiten.

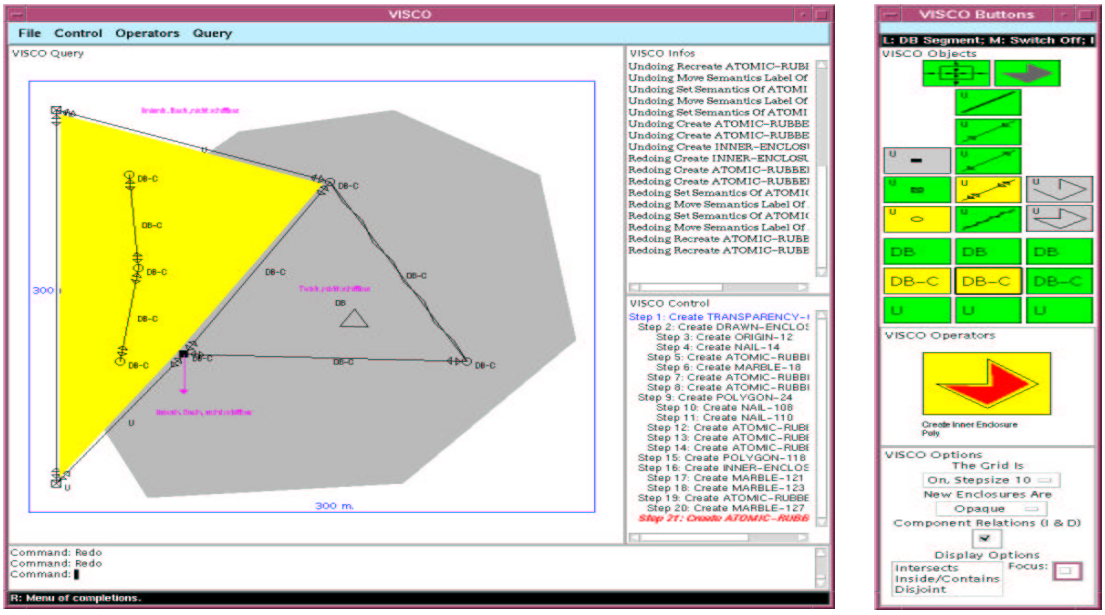




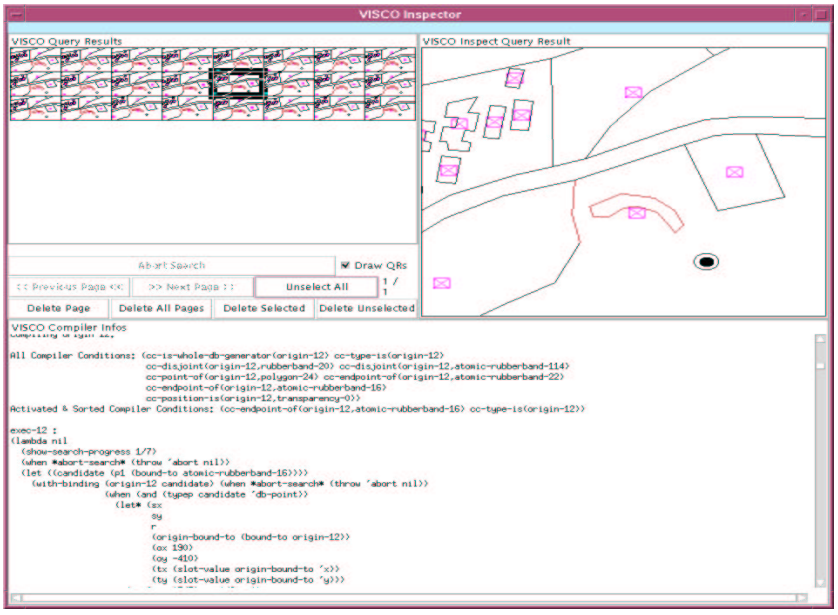
Schritt 11: Der Benutzer möchte nun die Anfrage weiter verfeinern, um einen speziellen Teich aufzufinden – westlich des Teiches soll nun ein kleiner Bach verlaufen. Obwohl es auch andere Möglichkeiten der Formulierung gäbe, entschließt er sich, ein geometrisches Hilfspolygon zu konstruieren, welches eine Seite mit dem Teich *teilt*, so daß sich die Form des Polygones der „westlichen Seite“ des Teiches anpaßt. Das Innere des Polygones kann dann als Aufenthaltsgebiet für einen Teil des Baches angesehen werden. Da es sich um ein Hilfspolygon (also ein Metaobjekt) handelt, ist es nicht im Datenbestand vorhanden (Anmerkung: die Meldungen im Info-Fenster haben keine Relevanz).



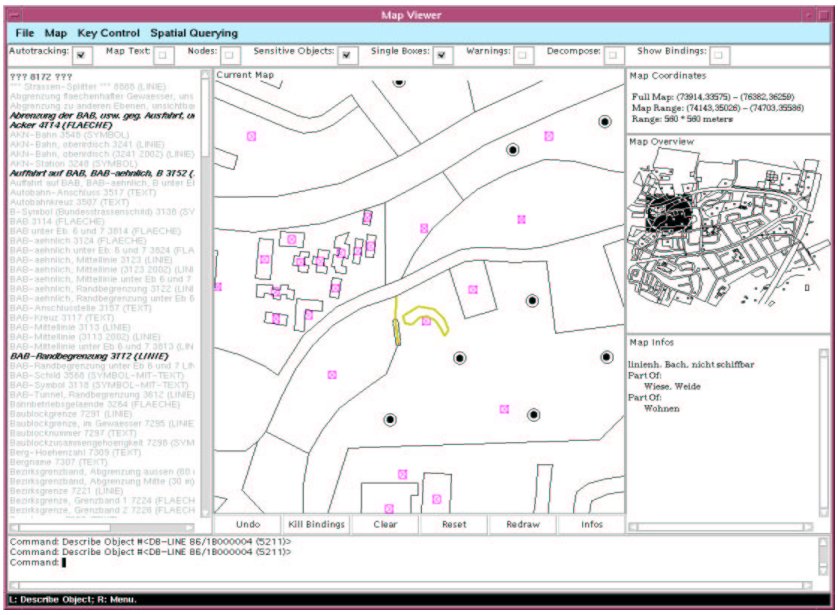
Schritt 12: Mit Hilfe des Operators „Erzeuge Inneres“ wird das innere Gebiet des so konstruierten und mit dem Teich verknüpften bzw. von diesem abhängigen Hilfspolygones konstruiert. Wie eine Gummihaut spannt es nun ein Gebiet zwischen den beiden festgelegten Ecknägeln und der Seite des Teiches auf. Würde nun diese Seite (oder ein Eckpunkt) verschoben, so würde das System das abgeleitete Gebiet automatisch neu berechnen und anpassen.



Schritt 13: Der Benutzer konstruiert zwei Segmente des Baches in das innere Gebiet und weist diesen die entsp. Thematik zu. Für die beiden Bachsegmente wird lediglich verlangt, daß sie sich innerhalb des opaquen inneren Gebietes aufhalten, denn das skizzierte Gebiete wird ja verdeckt. Die thematischen Beschriftungen verschiebt er interaktiv mit der Maus, damit sie besser lesbar sind.

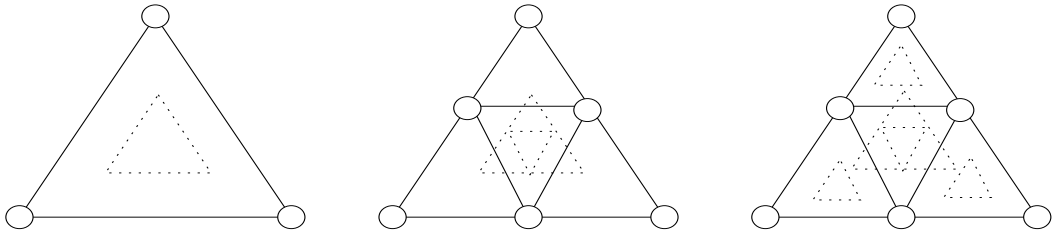


Schritt 14: Die Ausführung der Anfrage ergibt schließlich die gewünschte Konstellation.



Sucht der Benutzer nun z.B. ein zum Verkauf stehendes Haus in der Nähe dieser idyllischen Konstellation, so kann er sich über die Umgebung mit Hilfe des Map Viewer informieren, oder aber die Anfrage entsp. verfeinern und erneut stellen (im zur Verfügung stehenden Datenbestand gibt es jedoch nur eine derartige Konstellation).

7.2 Beispiel 2: Interaktion mit Komplexobjekten



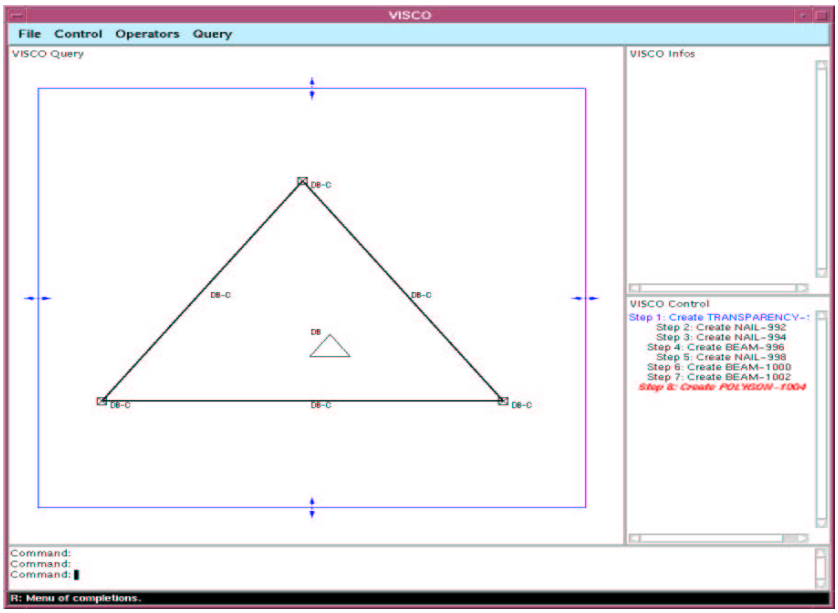
(a) Ein explizites Dreieck

(b) Zwei explizite Dreiecke, fünf implizite

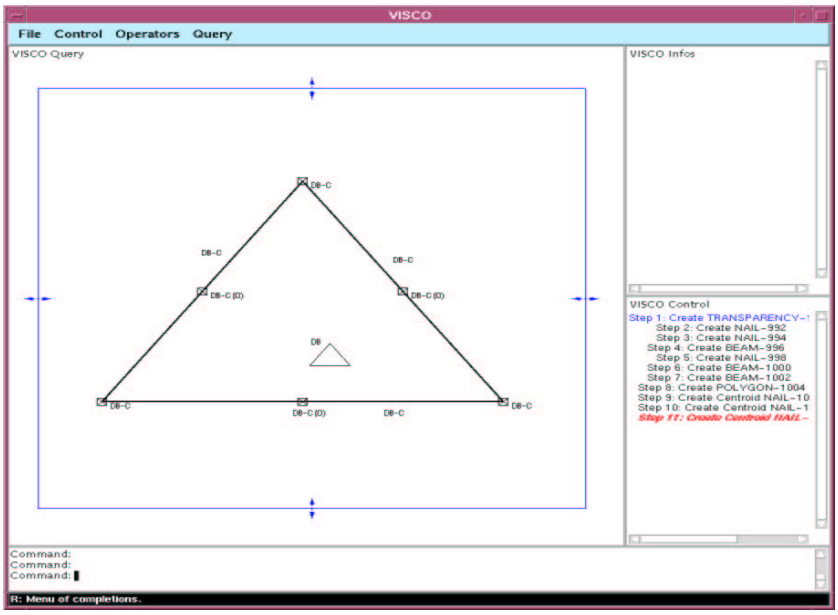
(c) Fünf explizite Dreiecke

Abbildung 7.1: Explizierung auftauchender Polygone

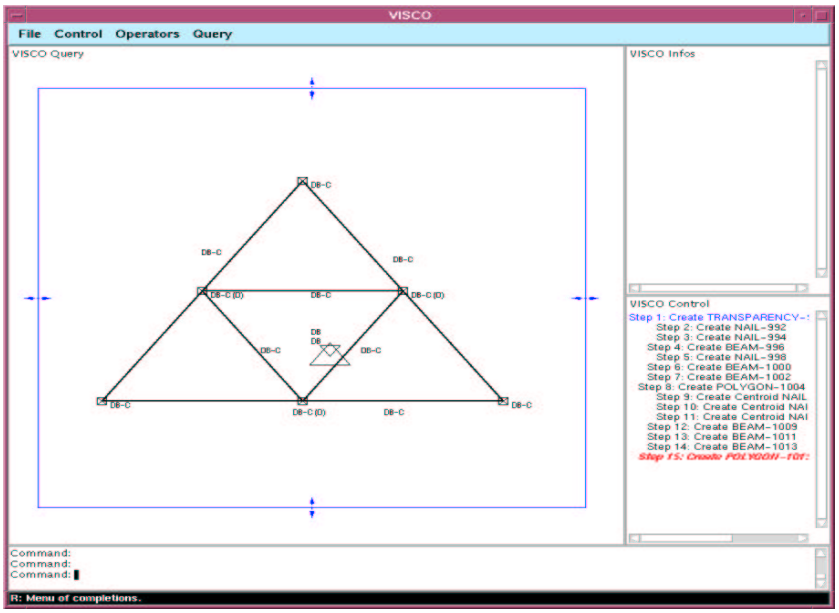
Ein weiteres Beispiel soll die Interaktion mit Komplexobjekten im Grafikeditor und die Explizierung auftauchender Objekte verdeutlichen. Hierzu dient das bereits in Kap. 2 diskutierte Beispiel des in ein Dreieck über die Mittelpunkte der Seiten eingeschriebenen Dreieckes (s. Abb. 7.1). Die folgenden Abbildungen stellen keine legalen VISCO-Definition dar und würden vom Compiler daher nicht übersetzt werden, da sie unvollständig sind.



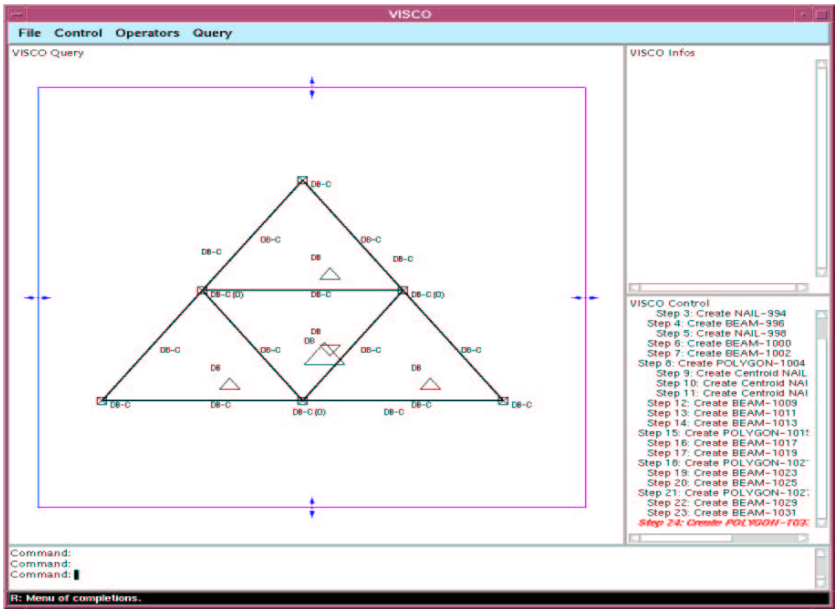
Schritt 1: Erzeugung eines expliziten Polygones.



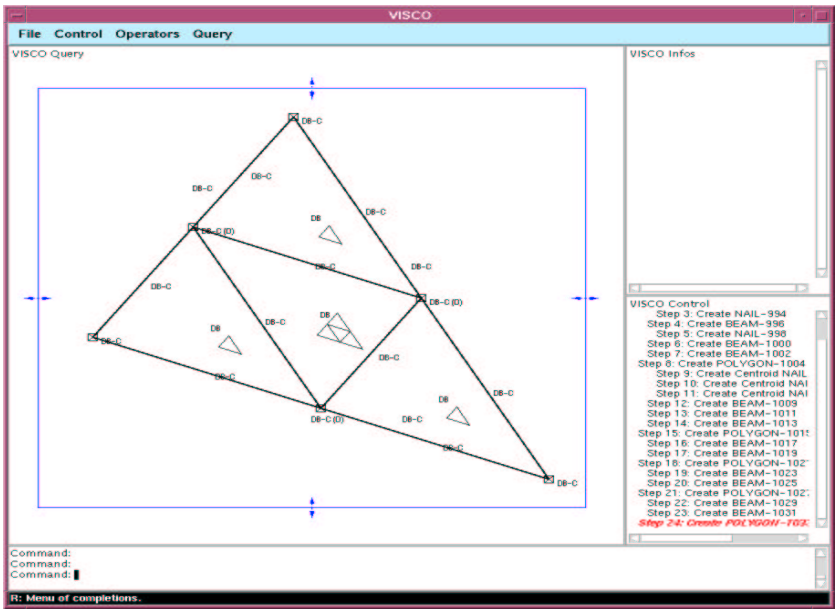
Schritt 2: Auf jede Dreiecksseite wird der Operator „Erzeuge Mittelpunkt“ angewendet.



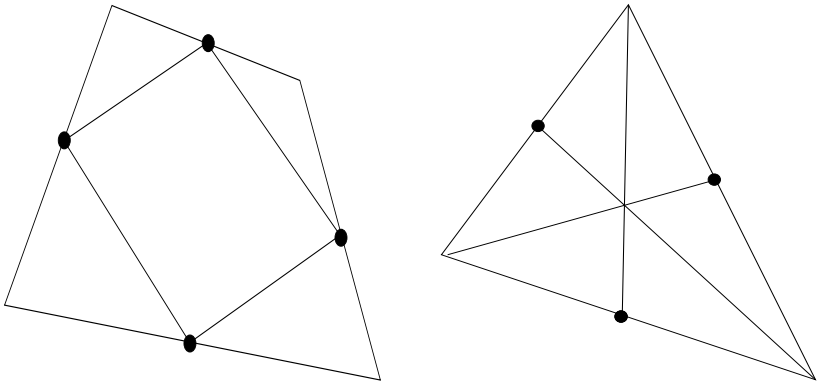
Schritt 3: Anhand der Mittelpunkte wird das eingeschriebene Dreieck konstruiert, was durch Aggregierung geschieht. Nun existieren zwei explizite Dreieck, die durch die – hier leicht verschobenen – ikonischen Repräsentanten explizit sichtbar sind. Aber auch drei weitere implizite Dreieck sind ersichtlich (in den Ecken des großen Dreiecks).



Schritt 4: Durch entsp. Kantenbildung und Aggregierung der Eck- sowie Mittelpunke werden die drei impliziten Dreiecke nun explizit gemacht.



Schritt 5: Wird nun ein beliebiges nicht-abgeleitetes Objekt verschoben (die abgeleiteten Mittelpunkte können nicht verschoben werden), so wird das gesamte Gebilde angepaßt.



(a) (b)

Abbildung 7.2: Geometrische Theoreme

Das bekannte „ThingLab“-Beispiel ([Bor81]) bzw. Theorem, daß das durch die Mittelpunkte der Seiten eines beliebigen Viereckes gebildete Viereck selbst stets ein Parallelogramm ist (s. Abb. 7.2(a)), könnte hier ebenfalls demonstriert werden. Eine anderes Theorem besagt, daß sich die Meridiane eines beliebigen Dreieckes stets in einem Punkt schneiden (s. Abb. 7.2(b), aus [JF93]). Natürlich handelt es sich bei den so durch Interaktion gewonnenen Einsichten nicht um mathematische Beweise dieser geometrischen Theoreme, dennoch aber um gute Anhaltspunkte bzw. Verdachtsmomente für das Aufstellen solcher. Ein interessantes interaktives System in der Tradition von „SketchPad“, „ThingLab“ und „LOGO“ ist „The Geometer's Sketchpad (GSP)“: mit diesem tutoriellen interaktiven System kann explorativ Geometrie gelehrt bzw. *erfahrbar* gemacht werden (lt. [JF93] wird es an tausenden amerikanischen High Schools eingesetzt).

Kapitel 8

Bewertung und Ausblick

Die geführte Diskussion zeigte, daß der Entwurf und die Realisierung einer neuen visuellen Sprache ein aufwendiges und ambitioniertes Vorhaben ist, welches in vollem Umfang und Tiefe im Rahmen einer Diplomarbeit nicht geleistet werden kann. Dennoch konnte im Verlauf der Arbeit ein lauffähiger Prototyp für die neue visuelle Sprache VISCO entwickelt werden. Es wurde „operational“ argumentiert, daß die durch den Prototypen implementierte Teilmenge der Sprache entscheidbar ist und somit alle Anfragen stets nach endlicher Zeit beantwortet werden können, also terminieren (s. Kap. 6).

Insbesondere die zunächst für VISCO als wesentlich angesehene Animationskomponente (s. [HW97]) konnte aufgrund zeitmangels und großer technischer Probleme bei der Umsetzung leider nicht realisiert werden. Aber auch bzgl. theoretischer Fragestellungen herrscht an vielen Stellen noch Unklarheit: Ob z.B. *Subsumption* von Anfragen entschieden werden kann, ist ungeklärt (eine Anfrage bzw. Definition A subsumiert eine Definition bzw. Anfrage B, wenn die entsp. Extensionen dieser intensional definierten „Konzepte“ in der Teilmengenrelation zueinander stehen, also $B \subseteq A$ gilt). Wäre Subsumption entscheidbar, so könnten z.B. Anfrageergebnisse vorheriger Anfragen aufbewahrt und im Falle einer neu gestellten, von der vorherigen Anfrage subsumierten oder auch subsumierenden Anfrage *wiederverwendet* werden (im Fall einer subsumierten Anfrage müssten für diese noch weitere Einschränkungen geprüft werden, im Fall einer subsumierenden Anfrage könnten diese jedoch direkt übernommen werden – zusätzlich wäre dann aber eine weitere Suche erforderlich). Entsprechende Kalküle zur Entscheidung derartiger Fragen konnten jedoch nicht entwickelt werden, auch die Normalisierung (s. Kap. 6) von Anfragen wurde nicht weiter untersucht. Offensichtlich gibt es in VISCO in der Regel sehr viele äquivalente Möglichkeiten, ein und dasselbe „räumliche Konzept“ (also mit derselben Extension) zu definieren (gleiches gilt übrigens für SQL). Eine Subsumptionsentscheidung würde daher wahrscheinlich einen Normalisierer erfordern. In Zukunft könnten weitere Untersuchungen evtl. auch unter Verwendung von Beschreibungslogiken stattfinden ([WS92, BMPS$^+$91]): mit ihrer Hilfe könnte evtl. Eigenschaften von VISCO-Anfragen auf der Meta-Ebene noch *vor* Ausführung der Suche entschieden werden, so z.B., ob eine Anfrage überhaupt beantwortbar ist oder „inkonsistent“ ist, also die leere Extension besitzt. Um räumliche Konzepte erweiterte Beschreibungslogiken werden von Haarslev, Lutz und Möller (in alphabetischer Reihenfolge) untersucht ([LHM97, MHL97]) und würden in diesem Zusammenhang relevant werden. Da die Ausdrucksmächtigkeit im Vergleich zu gewöhnlicher Prädikatenlogik von Beschreibungslogiken jedoch aufgrund formaler Kriterien (wie Entscheidbarkeit best. Fragen) in der Regel stark beschnitten wird, stellt sich die Frage, ob die von den untersuchten um räumliche Konzepte erweiterten Beschreibungslogiken ausdrucksmächtig genug sind, oder ob nur eine Teilmenge von VISCO beschrieben werden kann (insbesondere für die Semantik von VISCO-Definitionen sehe ich große Probleme).

Auch die Frage der abstrakten Syntax- und Semantikbeschreibung von VISCO ist nicht abschließend geklärt: im Anhang findet sich jedoch ein vollständiger (alle Aspekte der Sprache betreffen-

der) Definitionsversuch der Sprache, der auf gewöhnlicher Prädikatenlogik (erster Stufe) basiert. Untersuchungen zum Einsatz von Beschreibungslogiken für VISCO könnten sich an den dort zu findenden eingesetzten Sprachmitteln bzw. -formalien orientieren. Es wurden auch diverse Formalisierungsexperimente mit anderen Formalismen unternommen (s. Anhang). Aus meiner Perspektive stellt sich die Formalisierung visueller Sprachen als noch weitgehend ungeklärtes Problem dar und ist als erheblich schwieriger einzustufen, als die Formalisierung konventioneller textueller Progammiersprachen (s. auch Kap. 2).

Zum Compiler ist anzumerken, daß der Optimierer noch sehr rudimentär ist und recht grobe (aber wirkungsvolle) Heuristiken verwendet. Es stellt sich die Frage (angesichts schon sehr guter bzw. nahezu „optimaler“ erzeugter Ausführungspläne), ob eine weitergehende Verfeinerung der Bewertungsstrategien noch einen Effiziengewinn bzw. „optimalere“ Suchprogramme ergeben würde. Technisch wäre auch eine Kopplung mit einer leistungsfähigeren (kommerziellen) und wahrscheinlich performanteren Schicht „räumlicher Datenbestand“ (z.B. einer räumlichen Datenbank) wünschenswert (s. Kap. 2), um zu umfangreicheren und effizienteren räumlichen Indizierungsverfahren zu gelangen. Beispielsweise könnten zusätzliche Indizes aufgebaut werden (z.B. thematische Indizes). In Kap. 6 wurde ausführlich dargestellt, daß Indizies ein *sehr* wesentliches Mittel zur Eingrenzung der Suchkomplexität darstellen. Zudem stellt sich die Frage, wie gut das Zeitverhalten der Suche auf einem sehr viel größeren räumlichen Datenbestand ist (der verwendete DISK-Datenbestand ist sehr klein). Der VISCO-Prototyp ist ein komplexes Programm, das sicherlich (wie jedes umfangreiche Programm) noch eine Menge unentdeckter Fehler beinhaltet. Aufgrund einer meineserachtenes konzeptionell bzw. logisch sauberen Architektur und Aufgabentrennung scheint eine Wartung jedoch gut möglich.

Zur Anfrageformulierung mit dem Prototypen VISCO ist anzumerken, daß man teilweise Überraschungen beim Formulieren bzw. bei der Ausführung von Anfragen erlebt: teilweise haben subtile geometrische Beziehungen in den VISCO-Definitionen bzw. Diagrammen einen starken Einfluß auf die Extension bzw. das Anfrageergebnis. Beispielsweise kann manchmal die feste Form eines vom Benutzer jedoch entsp. vage gemeinten konstanten Gebietes einen Einfluß haben. Oftmals hat man das Problem, daß man nicht genau weiß, wie man nun eine best. Anfrage konkret umsetzen soll. Im Rahmen der Anfrageformulierung müssen vom Benutzer zwangsweise räumliche Aspekte wie Form von konstanten Gebieten, Positionen von Punkten etc. festgelegt werden. Selbst mit den vorhandenen Abstraktionsmöglichkeiten (räumlichen „don't care“-Relationen) in Form von Verdeckungen entstehen teilweise überspezifizierte Anfragen (s. Kap. 4). Die Anfrageformulierung ist als schwieriger einzustufen, als zunächst erwartet. An mir selbst konnte ich beobachten, daß auch der Umgang mit bzw. das Verständnis derartiger Konzepte durch „mentale Animation“ umfangreiches Training erfordert und keinesfalls so einfach wie vermutet ist, da die verwendeten Objekte sehr schnell ein überaus komplexes Verhalten offenbaren, welches mental sehr schwer faßbar ist. Eventuell kann eine Animationskomponente hierbei Unterstützung bieten. Als Alternative böte sich eine dynamische Inspektionsmethode an: der Benutzer könnte einzelne Objekte versuchsweise (mit der Maus) anfassen und manipulieren, wobei die Physik der Objekt simuliert werden soll. Eine Murmel könnte er mit der Maus umherrollen, einen Nagel jedoch nicht; eine rotiertbare Folie könnte er rotieren, ein Gummiband „eindellen“, etc. So könnten gezielt „Frage“ an das Constraintsystem gestellt werden, und die physikalischen Metaphern wären tatsächlich in gewisser Weise „mehr“ als nur Metaphern und bekämen „Leben“.

Attraktiv wäre auch die Möglichkeit, VISCO im World Wide Web als Dienst (z.B. als Java-Client) einzusetzen. Benutzer könnten somit interessierende räumliche Konstellationen (z.B. „alle Supermärkte in der Nähe der eigenen Wohnstätte“ – diese Anfrage hätte Ähnlichkeit mit der Def. in Abb. 5.1 in Kap. 5) von einem speziellen Dienst-Server erhalten, welcher einen lokalen räumlichen (und für VISCO geeigneten) räumlichen Datenbestand verwaltet. Da die Sprache in ihrer jetzigen Form für Endanwender wahrscheinlich zu kompliziert ist, sollte man die in diesem Kontext wahrscheinlich irrelevanten Sprachmittel streichen (z.B. mögen für Anfragen obiger Art Folien fester Größe, konstante Gebiete und Murmeln ausreichen, evtl. noch Gummibänder für Straßen o.ä.). Die Eignung der Sprache VISCO für End- und Gelegenheitsbenutzer konnte mangels Testpersonen

nicht evaluiert werden. Meine Vermutung ist, daß die Sprache (und auch der syntaxgesteuerte Grafikeditor des Prototypen) momentan noch zu kompliziert in der Handhabung sind.

Abschließend ist zu bemerken, daß die Sprache VISCO eine neue visuelle Sprache ist und einige neue Ideen beinhaltet, die bisher in dieser Form bzw. Kombination noch keine Verwendung gefunden haben. Insbesondere die empirische Klärung der Frage, ob durch die verwendeten physikalischen Metaphern *tatsächlich* ein Verständnisgewinn beim Benutzer erzielt wird, wäre eine Untersuchung wert (man könnte einer Testgruppe den Prototypen ohne Metaphern, der anderen diesen jedoch mit Metaphern erklären und benutzen lassen). Derartige Untersuchung erfordern jedoch eine noch sehr viel weitgehendere Weiterentwicklung des Prototypen, der in seiner jetzigen Form wahrscheinlich nur vom Autor bedient werden kann.

Ob das Experiment VISCO als gelungen oder als Fehlschlag anzusehen ist, muß der Beurteilung des Lesers überlassen werden.

Anhang A

Formale Sprachdefinition von VISCO

Die folgende formale Sprachbeschreibung von VISCO basiert auf gewöhnlicher Prädikatenlogik erster Ordnung ([NS93, Sch92]). Axiome definieren dabei notwendige Eigenschaften einzelner Objekte, die zu Klassen oder Sorten zusammengefaßt werden: Instanzen bzw. Objekte dieser Klassen müssen entsp. Konsistenzbedingungen erfüllen. Diese Konsistenzbedingungen beschreiben sowohl Syntax als auch Semantik.

Diverse Experimente mit anderen Formalismen wurden unternommen: unter anderem mit Graphgrammatiken ([RS96, RS95, MV95, Min97]), der Spezifikationssprache Z ([Spi89]), normaler EBNF ([Mey90]) und Beschreibungslogiken. Stets gerieten die Spezifikationen entweder unnötig kompliziert und unübersichtlich, oder aber die verwendeten Formalismen waren nicht ausdrucksmächtig genug.

Um eine bessere Lesbarkeit und Modularität der einzelnen Axiome zu erreichen, wird eine spezielle Notation verwendet. Diese Notation stellt einige spezielle Relationen in den Vordergrund - so z.B. die Teilmengenbeziehung zwischen „Klassen“ (in Form der Implikation). Die Notation legt also besonderes Augenmerk auf Vererbung. Diese wird jedoch so genutzt, daß sich keine der aus der Wissenrepräsentation bekannten Probleme (s. [Bib93, Kap. 2.6]). VISCO ist vollständig formalisiert: sowohl Syntax als auch Semantik der Sprache werden im folgenden vollständig beschrieben. Allerdings sind die formalen Eigenschaften der Sprache weitestgehend unbekannt - für den implementierten Prototypen wurde jedoch in Kap. 6 argumentiert (und könnte auch bewiesen werden), daß die Frage, ob eine VISCO-Definition erfüllbar ist oder nicht, entscheidbar ist (solange der Datenbestand endlich ist). Der Prototyp implementiert jedoch nur eine gewisse Teilmenge der in der Spezifikation dargestellten Sprache. In der Spezifikation wird die implementierte Teilmenge explizit durch zusätzliche Einschränkungen ausgewiesen

Von der Syntax von VISCO wird nur der abstrakte Teil spezifiziert: die konkrete Syntax wird hier größtenteils nicht festgelegt. Metaobjekte, die lediglich bestimmte Attribute festlegen oder einschränken (wie z.B. „Winkeleinschränker“, „Zeiger“ und „Skala“ etc.) werden nur als prädikatenlogische Relationen oder Attribute (bzw. Funktionen) auf einzelnen Objekten definiert. Letztlich sollten „gute Visualisierungen“ dieser Metaobjekte aber auch best. Bedingungen erfüllen, die hier nicht festgelegt werden (und zudem schwer zu formalisieren sind). Die Ausgestaltung der konkreten Syntax anhand der abstrakten Syntax ist Aufgabe einer Präsentationskomponente (s. Kap. 6).

Von der konkreten Syntax wird jedoch die Geometrie festgelegt: dies ist in gewisser Weise unvermeidlich, da ja die VISCO-Objekte selbst geometrische Objekte repräsentieren sollen. Anhand der Geometrie der VISCO-Objekte können z.B. die verwendeten räumlichen Relationen ermittelt werden. Dies erfordert eine formale Definition der verwendeten räumlichen Relationen, und auch

der in Kap. 5 erläuterten *Sichtbarkeitsregeln.* Letztendlich handelt es sich jedoch um eine „abstrakte“ Geometrie, da z.B. eine Murmel nicht als nulldimensionaler Punkt verwendet werden kann, sondern selbst ein zweidimensionales Objekt mit einer bestimmten Form und Ausdehnung etc. ist. Diese Teile der konkreten Syntax werden hier nicht festgelegt, und auch die sich daraus ergebende Problematik wird nicht weiter diskutiert. Hier werden somit abstrakte Syntax als auch Semantik der Sprache definiert.

Auf dieser abtrakten Ebene ist eine konkrete VISCO-Definition in Form eines Modelles der folgenden Axiome gegeben. Da auch die Semantik definiert werden soll, ist auch der verwendete räumliche Datenbestand Teil des Modelles bzw. die durch die VISCO-Objekte repräsentierten Domänenobjekte. Der Grund- oder Individuenbereich des Universum wird in Sorten aufgeteilt: so gibt es die Objekte der aktuell betrachteten VISCO-Definition (stets endlich viele), die Objekte des geometrischen Universums (überabzählbar viele) und die Objekte des räumlichen Datenbestandes (stets endlich viele) bzw. der repräsentierten Domäne.

Für die Semantik sind letztendlich die Bindungen der Funktion *bound_to* wesentlich: VISCO-Objekte repräsentieren die Objekte des Universums, an die sie gebunden werden können. Die Funktion *bound_to* ist also auf geometrischen VISCO-Objekten definiert und liefert das korrespondierende (repräsentierte) geometrische Objekt des Universums zurück. Spezielle VISCO-Objekte stellen spezielle Anforderungen an die Bilder der *bound_to*-Funktion (entsp. der Semantik). Jede VISCO-Definition hat also (eine evtl. überabzählbare) Menge von Modellen. Jedes Modell stellt eine andere Belegungs- bzw. Bindungsmöglichkeit der VISCO-Objekte der Definition her. Jedes Modell ist abzählbar, vorausgesetzt, die Anzahl der verwendeten VISCO-Objekte der aktuellen Definition ist abzählbar.

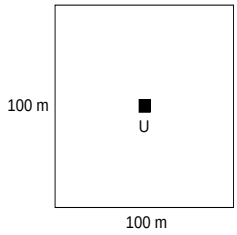


Abbildung A.1: Definition mit überabzählbarer Anzahl endlicher Modelle

Der Satz von „Löwenheim-Skolem“ (s. [Sch92, S. 84])

> *Jede erfüllbare Formel der Prädikatenlogik besitzt bereits ein abzählbares Modell (also eines mit abzählbarer Grundmenge).*

ist also - *für genau eine betrachtete Definition mit einer abzählbaren Anzahl von* VISCO*-Objekten* - erfüllt. Der Satz von Löwenheim-Skolem bedingt, daß z.B. die Menge der reellen Zahlen nicht prädikatenlogisch axiomatisiert werden kann (hierfür würde eine Logik zweiter Stufe benötigt). Das es für eine einzige Definition eine überabzählbare Anzahl (von endlichen und somit abzählbaren Modellen) geben kann, verdeutlicht die Definition in Abb. A.1. Der Ursprung könnte gegen jedes Punktobjekt im Universum gebunden werden, und die Anzahl der Punkte im Grundbereich ist als überabzählbar angenommen ($\mathbb{R}^2$). Wie bereits diskutiert, soll hiervon die Menge der Punkte der interessierenden Domäne bzw. des Datenbestandes eine spezielle endliche Teilmenge sein. Schränkt man VISCO entsprechend ein, so gibt es sogar nur noch eine endliche Anzahl von endlichen Modellen. Diese Einschränkungen wurden für den implementierten Prototypen gemacht (s. Kap. 6) - dort wurde ja argumentiert (wenn auch nicht formal bewiesen), daß die gestellten Anfragen stets „sicher“ sind, also terminieren.

A.1 Verwendete Notationen

Die in der Spezifikation aufgeführten Axiome definieren eine Reihe von Integritätsbedingungen: diese umfassen sowohl abstrakte Syntax als auch Semantik der Sprache.

Ein Axiom hat folgende Form:

class **Class** **Is Subclass Of (Implies)** $super\_class_1, super\_class_2, \ldots, super\_class_i$
Slots (Functions): $var_1 : Type_1$ $var_2 : Type_2$ $\vdots$ $var_j : Type_j$
Slots* (Functions, Exclusively For Semantics): $svar_1 : SType_1$ $svar_2 : SType_2$ $\vdots$ $svar_k : SType_k$
Uses (Additional Bindings): $v_1, v_2, \ldots, v_l$
Constraints (Predicates): c_1 c_2 $\vdots$ c_m
Semantics: s_1 s_2 $\vdots$ s_n

Dies ist lediglich eine andere Schreibweise für folgende Formel:

$$
\begin{aligned}
\forall self\ (\ & class(self) \Rightarrow \\
& super\_class_1(self) \wedge \\
& super\_class_2(self) \wedge \\
& \vdots \\
& super\_class_i(self) \wedge \\
& (\lambda(var_1, var_2, \ldots, var_j, \\
& \quad svar_1, svar_2, \ldots, svar_k, \\
& \quad v_1, v_2, \ldots, v_l) \bullet \\
& \quad\ var_1 \in \mathsf{Type_1} \wedge var_2 \in \mathsf{Type_2} \wedge \cdots \wedge var_j \in \mathsf{Type_j} \wedge \\
& \quad\ svar_1 \in \mathsf{SType_1} \wedge svar_2 \in \mathsf{SType_2} \wedge \cdots \wedge svar_k \in \mathsf{SType_k} \wedge \\
& \quad\ c_1 \wedge c_2 \wedge \cdots \wedge c_m \wedge \\
& \quad\ s_1 \wedge s_2 \wedge \cdots \wedge s_n \\
& \ (var_1(self), var_2(self), \ldots, var_j(self), \\
& \ \ svar_1(self), svar_2(self), \ldots, svar_k(self), \\
& \ \ v_1(self), v_2(self), \ldots, v_l(self))))
\end{aligned}
$$

Zusätzlich wird für jede auf diese Art definierte „Sorte“ oder „Klasse“ die Extension angenommen:

$\mathsf{class} = \{x \mid class(x)\}$

Taucht in den „Class“-Axiomen die Klausel „Current Implementation Limitations (Predicates)“ auf, so werden hiermit die momentan aktuellen Beschränkungen des implementierten Prototypen ausgewiesen (s. auch Kap. 6). Zusätzlich zu den „Class“-Axiomen (wie oben) gibt es noch die sogenannten „Abstract Class“-Axiome: diese unterscheiden sich formal in nichts von den „Class“-Axiomen. Eine Klasse, von der keine direkten Instanzen erzeugt werden, wird in der objektorientierten Programmierung auch als abstrakte Klasse bezeichnet. In den „Abstract Class“-Axiomen finden sich teilweise Formeln der Form $\oplus(a(self), b(self), \ldots, x(self))$. Dies bedeutet natürlich, daß die entsp. abstrakte Klasse vollständig disjunkt durch die Klassen $a, b, \ldots, x$ partitioniert wird. Der Operator $\oplus$ bezeichnet dabei das verallgemeinerte XOR:

$$
\begin{array}{l}
\oplus(a_1, a_2, a_3, \ldots, a_n) \Leftrightarrow \\
\quad\begin{array}{rcrcrcccrcr}
a_1 & \wedge & \neg a_2 & \wedge & \neg a_3 & \wedge & \ldots & \wedge & \neg a_n & \vee \\
\neg a_1 & \wedge & a_2 & \wedge & \neg a_3 & \wedge & \ldots & \wedge & \neg a_n & \vee \\
\neg a_1 & \wedge & \neg a_2 & \wedge & a_3 & \wedge & \ldots & \wedge & \neg a_n & \vee \\
 & & & & \vdots & & & & & \\
\neg a_1 & \wedge & \neg a_2 & \wedge & \neg a_3 & \wedge & \ldots & \wedge & a_n &
\end{array}
\end{array}
$$

λ definiert eine annonyme Funktion und wird hier nur verwendet, um eine Bindungsumgebung aufzuspannen (meist werden nur andere Namen eingeführt, um die Lesbarkeit zu erhöhen). λ's werden also so verwendet, daß die Ausdrucksmächtigkeit der Prädikatenlogik 1. Stufe nicht erweitert wird (s. [RN95, S. 195]).

$\mathcal{P}(A)$ bezeichnet die Potenzmenge der Menge A.

$\mathbb{R}$ bezeichnet die Menge der reellen Zahlen.

$\mathbb{B}$ bezeichnet die Menge der Wahrheitswerte, $\{\bot, \top\}$.

$\exists!\ v\ (body) \equiv \exists\ v\ (body \wedge \forall v'\ (body_{\{v \to v'\}} \Rightarrow v = v'))$

$body_{\{v \to v'\}}$ bezeichnet die Umbennung aller Vorkommen der Variablen v in *body* nach v'.

Auf Tupeln der Form $(x_1, x_2, x_3, \ldots, x_n)$ sind eine Reihe von Funktionen (Projektionen) definiert: $first((x_1, x_2, x_3, \ldots, x_n)) = x_1$, $second((x_1, x_2, x_3, \ldots, x_n)) = x_2$, ..., $nth((x_1, x_2, x_3, \ldots, x_n)) = x_n$.

Mengenbezeichner werden in dieser Schriftart gesetzt.

Das **let** bindet eine Reihe von Variablen:

$$
\begin{array}{l}
\textbf{let}\ v_1 = expr_1 \\
\quad\ v_2 = expr_2 \\
\quad\ \vdots \\
\quad\ v_n = expr_n \\
\ \ body \\
\equiv \\
\lambda(v_1, v_2, \ldots, v_n) \bullet \\
\quad body \\
(expr_1, expr_2, \ldots, expr_n)
\end{array}
$$

Einige spezielle Relationen werden in Infixform dargestellt ($a = b$ für $equal(a, b)$, $e \in \mathsf{Menge}$ für $element\_of(e, \mathsf{Menge})$ bzw. $menge(e)$, $\mathsf{A} \subseteq \mathsf{B}$ für $subset\_of(\mathsf{A}, \mathsf{B})$, etc.). Zusätzlich werden Intervalle und die üblichen Prädikate bzw. Funktionen auf Mengen (wie $\subseteq$, $\setminus$, $\cup$, $\cap$ etc.) verwendet. Eine weitere verwendete Funktion ist die Matrizen-Komposition, $\circ$. Zeilenvektoren werden in der Form $[x; y; c]$ notiert.

Für die verwendeten Junktoren gelten die folgenden üblichen Präzedenzen (s. [Hei95, S. 357]):

1. $\forall \quad \exists \quad \exists! \quad \neg$
2. $\wedge$
3. $\vee$
4. $\Rightarrow \quad \Leftarrow \quad \Leftrightarrow$

A.2 Axiomatisierung von VISCO

Geometrische Objekte

geom_object
Abstract Class

Slots (Functions):

$parts : \mathcal{P}(\text{geom\_object})$
$vertices : \mathcal{P}(\text{geom\_point})$
$shape : \mathcal{P}(\mathbb{R}^2)$
$interior : \mathcal{P}(\mathbb{R}^2)$
$boundary : \mathcal{P}(\mathbb{R}^2)$

Constraints (Predicates):

$shape = interior \cup boundary$
$interior \cap boundary = \emptyset$
$\forall p\ (p \in parts \Leftrightarrow dpo(p, self))$
$\forall m\ (part\_of(self, m) \Rightarrow geom\_object(m))$
$\forall other$
$(\ geom\_object(other) \Rightarrow$
$(part\_of(self, other) \Leftrightarrow$
$dpo(self, other) \vee$
$\exists x\ (part\_of(self, x) \wedge part\_of(x, other))) \wedge$
$(related(self, other) \Leftrightarrow$
$part\_of(self, other) \vee part\_of(other, self)) \wedge$
$(intersects(self, other) \Leftrightarrow$
$boundary \cap boundary(other) \neq \emptyset) \wedge$
$(disjoint(self, other) \Leftrightarrow$
$\neg intersects(self, other)) \wedge$
$(inside(self, other) \Leftrightarrow$
$shape \subseteq interior(other) \wedge$
$(geom\_aggregate(other) \Rightarrow \exists p\ (part\_of(p, other) \wedge inside(self, p)))) \wedge$
$(contains(self, other) \Leftrightarrow$
$inside(other, self)) \wedge$
$(outside(self, other) \Leftrightarrow$
$interior(other) \wedge shape \cap shape(other) = \emptyset) \wedge$
$(excludes(self, other) \Leftrightarrow$
$outside(other, self)))$
$\neg geom\_point(self) \Rightarrow$
$\exists p\ (geom\_point(p) \wedge centroid\_of(p, self) \wedge$
$x(p) = \frac{\sum_{v \in vertices} x(v)}{|vertices|} \wedge$
$y(p) = \frac{\sum_{v \in vertices} y(v)}{|vertices|})$
$\oplus(geom\_point(self),$
$geom\_line(self),$
$geom\_chain(self),$
$geom\_polygon(self),$
$geom\_aggregate(self))$

geom_point **Class** **Is Subclass Of (Implies)** *geom_object*
Slots (Functions): $x, y : \mathbb{R}$
Uses (Additional Bindings): *vertices*, *parts*, *shape*, *interior*
Constraints (Predicates): $vertices = \{self\}$ $parts = \emptyset$ $shape = \{(x, y)\}$ $interior = \emptyset$

geom_line **Class** **Is Subclass Of (Implies)** *geom_object*
Slots (Functions): p_1, p_2 : geom_point $orientation : \mathbb{R}$ $length : \mathbb{R}$
Uses (Additional Bindings): *vertices*, *parts*, *shape*, *interior*
Constraints (Predicates): $vertices = \{p_1\ p_2\}$ $parts = vertices$ $shape = create\_line\_from\_to(p_1, p_2)$ $interior = \emptyset$ $\neg(x(p_1) = x(p_2) \wedge y(p_1) = y(p_2))$ $length = \sqrt{(x(p_1) - x(p_2))^2 + (y(p_1) - y(p_2))^2}$ $orientation = \textbf{let}\ dx = x(p_2) - x(p_1)$ $dy = y(p_2) - y(p_1)$ $\textbf{let}\ \alpha = \arctan \frac{\lvert dy \rvert}{\lvert dx \rvert}$ $\begin{cases} dx \geq 0, dy \geq 0 & : \quad \alpha \\ dx < 0, dy \geq 0 & : \quad \pi - \alpha \\ dx \geq 0, dy < 0 & : \quad 2\pi - \alpha \\ dx < 0, dy < 0 & : \quad \pi + \alpha \end{cases}$ $\forall other\ (geom\_line(other) \Rightarrow$ $(neighbours(self, other) \Leftrightarrow \lvert vertices \cap vertices(other) \rvert = 1))$

geom_chain_or_polygon **Abstract Class** **Is Subclass Of (Implies)** *geom_object*
Slots (Functions): $segments : \mathcal{P}(\mathsf{geom\_line})$
Uses (Additional Bindings): $vertices, segments, boundary$
Constraints (Predicates): $vertices = \bigcup_{s \in segments} vertices(s)$ $parts = segments$ $boundary = \bigcup_{s \in segments} boundary(s)$ $\forall s, s' \ (s, s' \in segments \wedge s \neq s' \Rightarrow$ $(shape(s) \cap shape(s') \neq \emptyset \Leftrightarrow neighbours(s, s')))$

geom_chain **Class** **Is Subclass Of (Implies)** *geom_chain_or_polygon*
Slots (Functions): $p_1, p_2 : \mathsf{geom\_point}$
Uses (Additional Bindings): $segments, interior, vertices$
Constraints (Predicates): $\lvert segments \rvert \geq 2$ $interior = \emptyset$ $\forall v \ (v \in vertices \Rightarrow$ $\lvert \{s \mid s \in segments \wedge dpo(v, s)\} \rvert \leq 2)$ **let** $end\_vertices =$ $\{v \mid v \in vertices \wedge \exists! \ s \ (s \in segments \wedge dpo(v, s))\}$ $\lvert end\_vertices \rvert = 2 \wedge$ $p_1 \in end\_vertices \wedge$ $p_2 \in end\_vertices \wedge$ $p_1 \neq p_2$

geom_polygon **Class** **Is Subclass Of (Implies)** *geom_chain_or_polygon*
Uses (Additional Bindings): $segments, vertices, interior$
Constraints (Predicates): $\forall v \ (v \in vertices \Rightarrow \lvert \{s \mid s \in segments \wedge dpo(v, s)\} \rvert = 2)$ $interior = create\_polygon\_interior\_of(self)$

geom_polygon_with_optional_holes **Class** **Is Subclass Of (Implies)** *geom_object*
Current Implementation Limitations (Predicates): Not implemented!
Slots (Functions): *master* : geom_polygon *holes* : $\mathcal{P}$(geom_polygon)
Uses (Additional Bindings): *parts*, *shape*, *boundary*, *interior*
Constraints (Predicates): $parts = \emptyset$ $shape = shape(master) \setminus \bigcup_{h \in holes} interior(h)$ $boundary = boundary(master) \cup \bigcup_{h \in holes} boundary(h)$ $\forall h \ (h \in holes \Rightarrow shape(hole) \subseteq interior)$ $\forall h1, h2 \ (\ h1, h2 \in holes \wedge h1 \neq h2 \Rightarrow$ $shape(h1) \cap shape(h2) = \emptyset)$

geom_rectangle **Class** **Is Subclass Of (Implies)** *geom_polygon*
Uses (Additional Bindings): *segments*, *shape*, *vertices*
Constraints (Predicates): $\|segments\| = 4$ $shape = create\_bounding\_box\_for(vertices)$

geom_aggregate **Class** **Is Subclass Of (Implies)** *geom_object*
Uses (Additional Bindings): *vertices*, *boundary*, *interior*
Constraints (Predicates): $vertices = \bigcup_{p \in parts} vertices(p)$ $boundary = \bigcup_{p \in parts} boundary(p)$ $interior = \bigcup_{p \in parts} interior(p)$

Geometrische Hilfsfunktionen

$create\_bounding\_box\_for(vertices) =$
$[min(\{x \mid \exists\, v \ (v \in vertices \wedge x(v) = x)\}); min(\{y \mid \exists\, v \ (v \in vertices \wedge y(v) = y)\})] \times$
$[max(\{x \mid \exists\, v \ (v \in vertices \wedge x(v) = x)\}); max(\{y \mid \exists\, v \ (v \in vertices \wedge y(v) = y)\})]$

$create\_line\_from\_to(a, b) =$
$\quad \{p \mid p = a + \lambda(b - a), \lambda \in [0; 1]\}$

$create\_polygon\_interior\_of(polygon) =$
$\quad \{p \mid p \in \mathbb{R}^2 \wedge \exists\, p'\ (p' \in \mathbb{R}^2\ \wedge$
$\qquad\qquad p \notin boundary(polygon)\ \wedge$
$\qquad\qquad \textbf{let}\ l = \{p'' \mid p'' = p + \lambda(p' - p), \lambda \in [0; 1]\}$
$\qquad\qquad\ \exists!\ s\ (s \in segments(polygon) \wedge l \cap shape(s) \neq \emptyset)\}$

Objekte des räumlichen Datenbestandes

thematic_object **Abstract Class**
Slots (Functions): $thematic\_descriptor : \mathcal{P}(\mathbb{N})$
Constraints (Predicates): $\oplus(db\_object(self), visco\_d\_object(self))$

Anmerkung: die Behandlung thematischer Aspekte als einfache und uninterpretierte Attribute geometrischer Objekte ist unzureichend und sollte in Zukunft anders erfolgen.

db_object **Abstract Class** **Is Subclass Of (Implies)** *geom_object thematic_object*
Slots (Functions): $db\_primary : \mathbb{B}$
Uses (Additional Bindings): *vertices*
Constraints (Predicates): $db\_primary \Leftrightarrow$ $\quad \neg\exists\, m\ (dpo(x, m) \wedge db\_object(m))$ $\oplus($ $db\_point(self)$, $\quad db\_line(self)$, $\quad db\_chain(self)$, $\quad db\_polygon(self)$, $\quad db\_aggregate(self))$
Current Implementation Limitations (Predicates): $\forall x\ (x \neq self \wedge visco\_base\_object(x) \Rightarrow vertices(x) \neq vertices)$

db_point **Class** **Is Subclass Of (Implies)** *db_object geom_point*
Uses (Additional Bindings): x, y
Current Implementation Limitations (Predicates): $\forall p\ (p \neq self \wedge db\_point(p) \Rightarrow \neg(x(p) = x \wedge y(p) = y))$

db_line **Class** **Is Subclass Of (Implies)** *db_object geom_line*
Uses (Additional Bindings): p_1, p_2
Constraints (Predicates): $db\_point(p_1)$ $db\_point(p_2)$

db_chain_or_polygon **Abstract Class** **Is Subclass Of (Implies)** *db_object geom_chain_or_polygon*
Uses (Additional Bindings): *segments*
Constraints (Predicates): $\forall s\ (s \in segments \Rightarrow db\_line(s))$

db_chain **Class** **Is Subclass Of (Implies)** *db_chain_or_polygon geom_chain*

db_polygon **Class** **Is Subclass Of (Implies)** *db_chain_or_object geom_polygon*

db_aggregate **Class** **Is Subclass Of (Implies)** *db_object geom_aggregate*
Uses (Additional Bindings): *parts*
Constraints (Predicates): $\forall p\ (p \in parts \Rightarrow db\_object(p))$

VISCO-Basisobjekte

visco_base_object **Abstract Class** **Is Subclass Of (Implies)** *geom_object*
Uses (Additional Bindings): *vertices*
Constraints (Predicates): $\oplus(visco\_object(self), visco\_query(self))$
Current Implementation Limitations (Predicates): $\forall x\ (x \neq self \wedge visco\_base\_object(x) \Rightarrow vertices(x) \neq vertices)$

visco_base_point **Abstract Class** **Is Subclass Of (Implies)** *visco_base_object geom_point*
Uses (Additional Bindings): x, y
Current Implementation Limitations (Predicates): $\forall p\ (p \neq self \wedge visco\_base\_point(p) \Rightarrow \neg(x(p) = x \wedge y(p) = y))$

visco_base_line **Abstract Class** **Is Subclass Of (Implies)** *visco_base_object geom_line*
Uses (Additional Bindings): p_1, p_2
Constraints (Predicates): $visco\_base\_point(p_1)$ $visco\_base\_point(p_2)$

visco_base_chain_or_polygon **Abstract Class** **Is Subclass Of (Implies)** *visco_base_object geom_chain_or_polygon*
Uses (Additional Bindings): *segments*
Constraints (Predicates): $\forall s\ (s \in segments \Rightarrow visco\_base\_line(s))$

visco_base_chain **Abstract Class** **Is Subclass Of (Implies)** *visco_base_chain_or_polygon geom_chain*

visco_base_polygon **Abstract Class** **Is Subclass Of (Implies)** *visco_base_chain_or_polygon geom_polygon*

visco_base_rectangle **Abstract Class** **Is Subclass Of (Implies)** *visco_base_polygon geom_rectangle*

visco_base_aggregate **Abstract Class** **Is Subclass Of (Implies)** *visco_base_object geom_aggregate*
Uses (Additional Bindings): *parts*
Constraints (Predicates): $\forall p\ (p \in parts \Rightarrow visco\_base\_object(p))$

Verwendete Hilfsprädikate und -Funktionen

$$
\begin{aligned}
&\mathit{fully\_visible}(x, \mathit{from}, \mathit{to}) \Leftrightarrow \\
&\quad \mathit{visco\_object}(\mathit{from}) \wedge \\
&\quad \mathit{visco\_object}(\mathit{to}) \wedge \\
&\quad \forall e\ (\mathit{visco\_enclosure}(e) \wedge \mathit{opaque}(e) \wedge \\
&\qquad \mathit{after}(e, \mathit{from}) \wedge \mathit{before}(e, \mathit{to}) \Rightarrow \\
&\qquad \mathit{shape}(e) \cap \mathit{shape}(x) = \emptyset)
\end{aligned}
$$

$$
\mathbf{E} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

$$
\mathbf{T}(tx, ty) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ tx & ty & 1 \end{bmatrix}
$$

$$
\mathbf{S}(sx, sy) = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

$$
\mathbf{R}(\alpha) = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

$$
\begin{aligned}
&\mathit{make\_matrix}(sx, sy, \alpha, ox, oy, tx, ty) = \\
&\quad \mathbf{T}(-tx, -ty) \circ \mathbf{T}(-ox, -oy) \circ \mathbf{R}(-\alpha) \circ \mathbf{S}(\tfrac{1}{sx}, \tfrac{1}{sy})
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{get\_matrix\_for\_transparency}(\mathit{transparency}) = \\
&\quad \textbf{let}\ \mathit{origin} = \mathit{origin}(\mathit{transparency}) \\
&\qquad \mathit{t\_parent} = \mathit{t\_parent}(\mathit{transparency}) \\
&\qquad \mathit{width} = \mathit{width}(\mathit{transparency}) \\
&\qquad \mathit{height} = \mathit{height}(\mathit{transparency}) \\
&\qquad \mathit{rectangle} = \mathit{rectangle}(\mathit{transparency}) \\
&\qquad sx = sx(\mathit{transparency}) \\
&\qquad sy = sy(\mathit{transparency}) \\
&\qquad tx = tx(\mathit{transparency}) \\
&\qquad ty = ty(\mathit{transparency}) \\
&\qquad \alpha = \alpha(\mathit{transparency}) \\
&\quad\ \ \textbf{let}\ ox = x(\mathit{origin}) - \textbf{if}\ \mathit{t\_parent}\ x(\mathit{origin}(\mathit{t\_parent}))\ \textbf{else}\ 0 \\
&\qquad\ \ oy = y(\mathit{origin}) - \textbf{if}\ \mathit{t\_parent}\ y(\mathit{origin}(\mathit{t\_parent}))\ \textbf{else}\ 0 \\
&\qquad\ \ \mathit{isx} = \textbf{if}\ \mathit{width}\ \tfrac{\mathit{width}}{\mathit{width}(\mathit{rectangle})}\ \textbf{else}\ 1 \\
&\qquad\ \ \mathit{isy} = \textbf{if}\ \mathit{height}\ \tfrac{\mathit{height}}{\mathit{height}(\mathit{rectangle})}\ \textbf{else}\ 1 \\
&\qquad \textbf{if}\ \mathit{t\_parent} \\
&\qquad\quad \mathit{make\_matrix}(sx * \mathit{isx}, sy * \mathit{isy}, \alpha, ox, oy, tx, ty) \circ \\
&\qquad\qquad \mathit{get\_matrix\_for\_transparency}(\mathit{t\_parent}) \\
&\qquad \textbf{else} \\
&\qquad\quad \mathit{make\_matrix}(sx * \mathit{isx}, sy * \mathit{isy}, \alpha, ox, oy, tx, ty)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{element\_of\_circle\_intervall}(e, \mathit{circle\_intervall}) \Leftrightarrow \\
&\quad \textbf{let}\ \mathit{from} = \mathit{first}(\mathit{circle\_intervall}) \\
&\qquad \mathit{to} = \mathit{second}(\mathit{circle\_intervall}) \\
&\quad\ \ (\mathit{from} \leq \mathit{to} \wedge e \in [\mathit{from}; \mathit{to}]\ ) \vee \\
&\quad\ \ (\mathit{from} > \mathit{to} \wedge e \notin\ ]\mathit{to}; \mathit{from}[\ )
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{equal\_matrix\_images}(\mathit{from}, \mathit{to}, \mathit{matrix\_from}, \mathit{matrix\_to}) \Leftrightarrow \\
&\quad \mathit{geom\_object}(\mathit{from}) \wedge \mathit{geom\_object}(\mathit{other}) \wedge \\
&\quad \{(xt, yt) \mid \exists\, p\ (p \in \mathit{vertices}(\mathit{from}) \wedge \\
&\qquad\qquad [x(p), y(p), 1] \circ \mathit{matrix\_from} = [xt, yt, 1])\} = \\
&\quad \{(xt, yt) \mid \exists\, p\ (p \in \mathit{vertices}(\mathit{to}) \wedge \\
&\qquad\qquad [x(p), y(p), 1] \circ \mathit{matrix\_to} = [xt, yt, 1])\}
\end{aligned}
$$

VISCO-Anfrage (VISCO-Definition)

visco_query **Class** **Is Subclass Of (Implies)** *visco_base_aggregate*
Slots* (Functions, Exclusively For Semantics): $\xi : \mathbb{B}$
Uses (Additional Bindings): *parts*
Constraints (Predicates): $\forall p\ (p \in \mathit{parts} \Rightarrow \mathit{visco\_transparency}(p) \wedge \neg \mathit{t\_parent}(p))$ $\exists!\ \mathit{other}\ (\mathit{visco\_query}(\mathit{other}))$
Current Implementation Limitations (Predicates): $\lvert\mathit{parts}\rvert = 1$
Semantics: $\xi = \bigwedge_{p \in \mathit{parts}} \xi(p)$

VISCO-Objekte

bindable_object **Abstract Class**
Slots* (Functions, Exclusively For Semantics): *bound_to* : geom_object
Constraints (Predicates): $\mathit{visco\_object}(\mathit{self})$
Semantics: $\exists!\ \mathit{other}\ (\mathit{bindable\_object}(\mathit{other}) \wedge \mathit{bound\_to} = \mathit{bound\_to}(\mathit{other}))$

visco_object **Abstract Class**
Constraints (Predicates): $\forall v\ (visco\_object(v) \wedge v \neq self \Rightarrow$ $\quad \oplus(before(v, self), before(self, v)) \wedge$ $\quad (before(v, self) \Leftrightarrow after(self, v)))$ $\neg before(self, self)$ $\forall a, b\ (visco\_object(a) \wedge visco\_object(b) \wedge$ $\quad before(a, self) \wedge before(self, b) \Rightarrow$ $\quad\quad before(a, b))$ $\oplus(visco\_db\_object(self),$ $\quad visco\_universe\_object(self),$ $\quad visco\_enclosure(self))$

visco_on_transparency_object **Abstract Class** **Is Subclass Of (Implies)** *visco_object*
Slots (Functions): *on_transparency* : visco_transparency
Constraints (Predicates): $part\_of(self, on\_transparency)$ $\neg visco\_origin(self) \Rightarrow \exists!\ t\ (visco\_transparency(t) \wedge part\_of(self, t))$ $\forall v\ (visco\_on\_transparency\_object(v) \Rightarrow$ $\quad (both\_on\_same\_transparency(self, v) \Leftrightarrow$ $\quad\quad \exists\ t\ (visco\_transparency(t) \wedge part\_of(self, t) \wedge part\_of(v, t))))$

visco_non_enclosure_object
Abstract Class
Is Subclass Of (Implies)
visco_object

Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_non\_enclosure\_object : \mathbb{B}$

Slots (Functions):
$primary : \mathbb{B}$
$ignore\_intersects\_and\_disjoint : \mathbb{B}$
$direct\_instantiable : \mathbb{B}$

Uses (Additional Bindings):
$boundary, bound\_to$

Constraints (Predicates):
$ignore\_intersects\_and\_disjoint \Rightarrow$
  $\neg primary \wedge \forall p\ (dpo(p, self) \Rightarrow ignore\_intersects\_and\_disjoint(p))$
$ignore\_intersects\_and\_disjoint \Rightarrow$
  $\forall m\ (\ dpo(self, m) \Rightarrow$
    $\forall p\ (dpo(p, m) \Rightarrow ignore\_intersects\_and\_disjoint(p)))$
$primary \Leftrightarrow \neg\exists\, m\ (visco\_non\_enclosure\_object(m) \wedge dpo(self, m))$
$direct\_instantiable \Leftrightarrow$
  $visco\_db\_object(self) \vee$
  $visco\_nail(self) \vee$
  $visco\_point(self) \wedge$
  $\exists\, a, b\ (\ intersection\_point\_of(self, a, b) \wedge$
    $direct\_instantiable(a) \wedge$
    $direct\_instantiable(b)) \vee$
  $\exists\, arg\ (\ arg \in explicit\_centroid\_of(self) \wedge$
    $direct\_instantiable(arg)) \vee$
  $visco\_universe\_object(self) \wedge$
  $\forall p\ (dpo(p, self) \Rightarrow direct\_instantiable(p))$
$\forall p\ (dpo(p, self) \Rightarrow fully\_visible(p, p, self) \wedge before(p, self))$
$\forall v\ (\ visco\_non\_enclosure\_object(v) \Rightarrow$
    $(\ disjoint(self, v) \wedge$
     $before(v, self) \wedge fully\_visible\_from\_to(v, v, self) \wedge$
     $\neg ignore\_intersects\_and\_disjoint \wedge \neg ignore\_intersects\_and\_disjoint(v) \Leftrightarrow$
       $visible\_disjoint(self, v)) \wedge$
    $(\ intersects(self, v) \wedge$
     $before(v, self) \wedge \neg related(v, self) \wedge$
     $\neg ignore\_intersects\_and\_disjoint \wedge \neg ignore\_intersects\_and\_disjoint(v) \wedge$
     $\exists\, p\ (p \in boundary(v) \cap boundary \wedge fully\_visible(p, v, self)) \Leftrightarrow$
       $visible\_intersects(self, v)))$

Semantics:
$\xi\_visco\_non\_enclosure\_object =$
  $\forall other$
    $(\ visco\_non\_enclosure\_object(self) \Rightarrow$
      $(visible\_intersects(self, other) \Rightarrow intersects(bound\_to, bound\_to(other))) \wedge$
      $(visible\_disjoint(self, other) \Rightarrow disjoint(bound\_to, bound\_to(other)))) \wedge$
  $(\neg visco\_transparency(self) \Rightarrow$
    let $rectangle = rectangle(on\_transparency)$
      $matrix = get\_matrix\_for\_transparency(on\_transparency)$
      $org\_matrix = \mathbf{T}(-x(origin(on\_transparency)),$
        $-y(origin(on\_transparency)))$
    $\exists\, domain\_rectangle, image\_of\_self$
      $(\ geom\_rectangle(domain\_rectangle) \wedge$
       $equal\_matrix\_images(domain\_rectangle, rectangle, org\_matrix, \mathbf{E}) \wedge$
       $equal\_matrix\_images(bound\_to, image\_of\_self, matrix, \mathbf{E}) \wedge$
       $contains(rectangle, image\_of\_self))) \wedge$
  $\forall p\ (\ dpo(p, self) \Rightarrow$
    $dpo(bound\_to(p), bound\_to) \vee$
    $(\ visco\_rubberband(p) \wedge geom\_chain(bound\_to(p)) \wedge$
     $\forall s\ (dpo(s, bound\_to(p)) \Rightarrow dpo(s, bound\_to))) \vee$
    $(\ visco\_rubberband(self) \wedge geom\_chain(bound\_to) \wedge$
     $(\ bound\_to(p) = p_1(bound\_to) \vee$
      $bound\_to(p) = p_2(bound\_to))))$

Geometrische VISCO-Objekte

visco_transparency
Abstract Class
Is Subclass Of (Implies)
visco_non_enclosure_object visco_base_aggregate bindable_object

Slots* (Functions, Exclusively For Semantics):

$\xi\_visco\_transparency : \mathbb{B}$
$tx, ty, sx, sy, \alpha : \mathbb{R}^{\oplus} \setminus \{0\}$

Slots (Functions):

$at\_most\_constraint : \mathbb{N} \setminus \{0\} \cup \{\bot\}$
$rectangle$: visco_base_rectangle
$origin$: visco_origin
t_parent : visco_transparency $\cup \{\bot\}$
$t\_children : \mathcal{P}$(visco_transparency)
$origin\_is\_fixed\_at\_parent : \mathbb{B}$
$origin\_can\_float\_in\_parent : \mathbb{B}$
$width, height : \mathbb{R}^{\oplus} \cup \{\bot\}$
$sx\_min, sy\_min :]0; 1] \cup \{\bot\}$
$sx\_max, sy\_max : [1; \infty[\cup \{\bot\}$
$orientation\_constraint : \mathcal{P}([0; 2\pi[\times [0; 2\pi[)$
$rotateable : \mathbb{B}$
$x\_scaleable, y\_scaleable : \mathbb{B}$
$proportional : \mathbb{B}$

Constraints (Predicates):

$origin \Rightarrow part\_of(origin, self)$
$t\_parent \Rightarrow part\_of(origin, t\_parent) \wedge before(t\_parent, self)$
$at\_most\_constraint \Rightarrow |parts| \leq at\_most\_constraint$
$\forall other\ (\ other \neq self \wedge visco\_transparency(other) \Rightarrow$
$\quad \forall cp\ (\ part\_of(cp, other) \wedge part\_of(cp, self) \Leftrightarrow$
$\quad\quad cp = origin \vee cp = origin(other)))$
$\forall p\ (part\_of(p, self) \Rightarrow visco\_on\_transparency\_object(p) \wedge inside(p, rectangle))$
$width \vee height \Rightarrow \neg t\_parent$
$sx\_min \vee sx\_max \Rightarrow t\_parent \vee width$
$sy\_min \vee sy\_max \Rightarrow t\_parent \vee height$
$proportional \Rightarrow$
$\quad sx\_min = sy\_min\ \wedge$
$\quad sx\_max = sy\_max$
$rotateable \Leftrightarrow$
$\quad \neg(\ |orientation\_constraint| = 1 \wedge$
$\quad\quad \exists\ c\ (\ c \in orientation\_constraint \wedge$
$\quad\quad\quad first(c) = second(c))) \vee$
$\quad |orientation\_constraint| \neq 1$
$x\_scaleable \Leftrightarrow$
$\quad \neg sx\_min = 1 \vee \neg sx\_max = 1$
$y\_scaleable \Leftrightarrow$
$\quad \neg sy\_min = 1 \vee \neg sy\_max = 1$
$origin\_can\_float\_in\_parent \Leftrightarrow$
$\quad (t\_parent \Rightarrow visco\_marble(origin))$
$t\_children = \{child \mid t\_parent(child) = self \wedge visco\_transparency(child)\}$
$origin\_is\_fixed\_at\_parent \Leftrightarrow$
$\quad \neg origin\_can\_float\_in\_parent$

visco_transparency (continued!)

Abstract Class

Current Implementation Limitations (Predicates):

$transparency\_independed\_instantiable(origin) \wedge$

let $independent\_nails =$

$\{n \mid visco\_nail(n) \wedge part\_of(n, self) \wedge transparency\_independed\_instantiable(n)\}$

$(proportional \vee rotateable \Rightarrow$

$|independent\_nails| \geq 1) \wedge$

$(x\_scaleable \wedge y\_scaleable \wedge rotateable \wedge \neg proportional \Rightarrow$

$\exists\, a, b, c\ (a, b, c \in independent\_nails \wedge det(a, b, c) \neq 0 \wedge$ Not implemented! $)) \wedge$

$(x\_scaleable \wedge y\_scaleable \wedge \neg rotateable \Rightarrow$

$\exists\, n\ (n \in independent\_nails \wedge \neg(x(n) = x(origin) \vee y(n) = y(origin)))) \wedge$

$((x\_scaleable \oplus y\_scaleable) \wedge rotateable \Rightarrow$ Not implemented! $) \wedge$

$(x\_scaleable \Rightarrow \exists\, n\ (n \in independent\_nails \wedge x(n) \neq x(origin))) \wedge$

$(y\_scaleable \Rightarrow \exists\, n\ (n \in independent\_nails \wedge y(n) \neq y(origin)))$

Uses (Additional Bindings):

bound_to

Semantics:

$\xi\_visco\_transparency =$

$geom\_aggregate(bound\_to) \wedge$

$(sx\_min \Rightarrow sx\_min \leq sx) \wedge$

$(sx\_max \Rightarrow sx\_max \geq sx) \wedge$

$(sy\_min \Rightarrow sy\_min \leq sy) \wedge$

$(sy\_max \Rightarrow sy\_max \geq sy) \wedge$

$(proportional \Rightarrow sx = sy) \wedge$

$(rot\_constraint \Rightarrow \exists\, c\ (c \in rot\_constraint \wedge \alpha \in c)) \wedge$

$(at\_most\_constraint \Rightarrow |parts(bound\_to)| \leq at\_most\_constraint) \wedge$

$\bigwedge_{p \in parts} \xi(p) \wedge$

$\bigwedge_{child \in t\_children} \xi(child) \wedge$

$\xi\_visco\_non\_enclosure\_object(self)$

visco_point
Abstract Class
Is Subclass Of (Implies)
visco_non_enclosure_object visco_on_transparency_object visco_base_point bindable_object

Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_point : \mathbb{B}$

Slots (Functions):
$transparency\_independent\_instantiable : \mathbb{B}$
$explicit\_centroid\_of : \mathcal{P}(\mathsf{visco\_line} \cup \mathsf{visco\_chain} \cup \mathsf{visco\_polygon} \cup \mathsf{visco\_transparency})$

Uses (Additional Bindings):
$bound\_to$

Constraints (Predicates):
$$\forall arg\ (arg \in explicit\_centroid\_of \Rightarrow before(arg, self) \wedge centroid\_of(self, arg))$$
$$\forall a, b\ (a \neq b \wedge visco\_line(a) \wedge visco\_line(b) \Rightarrow (intersection\_point\_of(self, a, b) \Leftrightarrow visible\_intersects(self, a) \wedge visible\_intersects(self, b)))$$
$$transparency\_independent\_instantiable \Leftrightarrow visco\_db\_object(self) \vee \exists\, a, b\ (intersection\_point\_of(self, a, b) \wedge transparency\_independent\_instantiable(a) \wedge transparency\_independent\_instantiable(b)) \vee \exists\, arg\ (arg \in explicit\_centroid\_of \wedge transparency\_independent\_instantiable(arg))$$

Semantics:
$$\xi\_visco\_point = geom\_point(bound\_to) \wedge \forall arg\ (arg \in explicit\_centroid\_of \Rightarrow centroid\_of(bound\_to, bound\_to(arg))) \wedge \xi\_visco\_non\_enclosure\_object(self)$$

visco_nail
Abstract Class
Is Subclass Of (Implies)
visco_point

Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_nail : \mathbb{B}$

Uses (Additional Bindings):
$bound\_to, on\_transparency$

Semantics:
$$\xi\_visco\_nail = \textbf{let}\ matrix = get\_matrix\_for\_transparency(on\_transparency)\ org\_matrix = \mathbf{T}(-x(origin(on\_transparency)), -y(origin(on\_transparenc))) \ equal\_matrix\_images(bound\_to, self, matrix, org\_matrix) \wedge \xi\_visco\_point(self)$$

visco_marble **Abstract Class** **Is Subclass Of (Implies)** *visco_point*
Slots* (Functions, Exclusively For Semantics): ξ*_visco_marble* : $\mathbb{B}$
Constraints (Predicates): $\exists\ e\ (visco\_enclosure(e) \wedge visible\_contains(e, sef))$
Semantics: ξ*_visco_marble* = $\quad \xi$*_visco_point*(*self*)

visco_origin **Abstract Class** **Is Subclass Of (Implies)** *visco_nail*
Slots* (Functions, Exclusively For Semantics): ξ*_visco_origin* : $\mathbb{B}$
Semantics: ξ*_visco_origin* = $\quad \xi$*_visco_nail*(*self*)

visco_line **Abstract Class** **Is Subclass Of (Implies)** *visco_non_enclosure_object visco_on_transparency_object visco_base_line bindable_object*
Slots* (Functions, Exclusively For Semantics): ξ*_visco_line* : $\mathbb{B}$
Uses (Additional Bindings): p_1, p_2
Constraints (Predicates): *visco_point*(p_1) *visco_point*(p_2)
Semantics: ξ*_visco_line* = $\quad \xi$*_visco_non_enclosure_object*(*self*)

visco_atomic_segment
Abstract Class
Is Subclass Of (Implies)
visco_line

Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_atomic\_segment : \mathbb{B}$

Slots (Functions):
$orientation\_constraint : \mathcal{P}([0; 2\pi[\times [0; 2\pi[)$

Uses (Additional Bindings):
$bound\_to$, $on\_transparency$

Constraints (Predicates):
$$\begin{aligned}
&\forall other, d\ (relative_orientation_constraint(self, other, d) \Rightarrow \\
&\qquad self \neq other \wedge \\
&\qquad relative_orientation_constraint(other, self, d) \wedge \\
&\qquad visco_atomic_segment(other) \wedge \\
&\qquad both_on_same_transparency(self, other) \wedge \\
&\qquad visible_intersects(self, other) \wedge \\
&\qquad d \in [0; 2\pi[) \\
&\forall other, d1, d2\ (relative_orientation_constraint(self, other, d1) \wedge \\
&\qquad relative_orientation_constraint(self, other, d2) \Rightarrow \\
&\qquad\quad d1 = d2)
\end{aligned}$$

Semantics:
$$\begin{aligned}
&\xi_visco_atomic_segment = \\
&\quad \textbf{let}\ matrix = get_matrix_for_transparency(on_transparency) \\
&\qquad geom_line(bound_to) \wedge \\
&\qquad \exists\, image_of_self \\
&\qquad\quad (equal_matrix_images(bound_to, image_of_self, matrix, \mathbf{E}) \wedge \\
&\qquad\quad (orientation_constraint \Rightarrow \\
&\qquad\qquad \exists\, c\ (c \in orientation_constraint \wedge \\
&\qquad\qquad\quad element_of_circle_intervall(orientation(image_of_self), c))) \wedge \\
&\qquad\quad \forall other, d \\
&\qquad\qquad (relative_orientation_constraint(self, other, d) \Rightarrow \\
&\qquad\qquad \exists\, image_of_other \\
&\qquad\qquad\quad (equal_matrix_images(bound_to(other), image_of_other, matrix, \mathbf{E}) \wedge \\
&\qquad\qquad\quad |orientation(image_of_self) - \\
&\qquad\qquad\qquad orientation(image_of_other)| \leq d))) \wedge \\
&\quad \xi_visco_line(self)
\end{aligned}$$

visco_antenna
Abstract Class
Is Subclass Of (Implies)
visco_atomic_segment

Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_antenna : \mathbb{B}$

Semantics:
$$\begin{aligned}
&\xi_visco_antenna = \\
&\quad \xi_visco_atomic_segment(self)
\end{aligned}$$

*visco_* $\leq$ *_antenna*
Abstract Class
Is Subclass Of (Implies)
visco_atomic_segment

Slots* (Functions, Exclusively For Semantics):
ξ*_visco_* $\leq$ *_antenna* : $\mathbb{B}$

Uses (Additional Bindings):
bound_to, *on_transparency*

Semantics:
ξ*_visco_* $\leq$ *_antenna* =
  let *matrix* = *get_matrix_for_transparency*(*on_transparency*)
    $\exists$ *image_of_self*
      (*equal_matrix_images*(*bound_to*, *image_of_self*, *matrix*, $\mathbf{E}$) $\wedge$
        length(*image_of_self*) $\leq$ *length*(*self*)) $\wedge$
  ξ*_visco_atomic_segment*(*self*)

*visco_* $\geq$ *_antenna*
Abstract Class
Is Subclass Of (Implies)
visco_atomic_segment

Slots* (Functions, Exclusively For Semantics):
ξ*_visco_* $\geq$ *_antenna* : $\mathbb{B}$

Uses (Additional Bindings):
bound_to, *on_transparency*

Semantics:
ξ*_visco_* $\geq$ *_antenna* =
  let *matrix* = *get_matrix_for_transparency*(*on_transparency*)
    $\exists$ *image_of_self*
      (*equal_matrix_images*(*bound_to*, *image_of_self*, *matrix*, $\mathbf{E}$) $\wedge$
        length(*image_of_self*) $\geq$ *length*(*self*)) $\wedge$
  ξ*_visco_atomic_segment*(*self*)

visco_stick
Abstract Class
Is Subclass Of (Implies)
visco_atomic_segment

Slots* (Functions, Exclusively For Semantics):
ξ*_visco_stick* : $\mathbb{B}$

Uses (Additional Bindings):
bound_to, *on_transparency*

Semantics:
ξ*_visco_stick* =
  let *matrix* = *get_matrix_for_transparency*(*on_transparency*)
    $\exists$ *image_of_self*
      (*equal_matrix_images*(*bound_to*, *image_of_self*, *matrix*, $\mathbf{E}$) $\wedge$
        length(*image_of_self*) = *length*(*self*)) $\wedge$
  ξ*_visco_atomic_segment*(*self*)

visco_rubberband
Abstract Class
Is Subclass Of (Implies)
visco_line

Slots* (Functions, Exclusively For Semantics):
ξ*_visco_rubberband* : $\mathbb{B}$

Slots (Functions):
at_most_constraint : $\mathbb{N} \setminus \{0\} \cup \{\perp\}$

Uses (Additional Bindings):
bound_to

Semantics:
ξ*_visco_rubberband* =
$(geom\_chain(bound\_to) \vee geom\_line(bound\_to)) \wedge$
$(geom\_chain(bound\_to) \wedge at\_most\_constraint \Rightarrow$
$|segments(bound\_to)| \leq at\_most\_constraint) \wedge$
$\xi visco\_line(self)$

visco_chain_or_polygon
Abstract Class
Is Subclass Of (Implies)
visco_non_enclosure_object visco_on_transparency_object visco_base_chain_or_polygon bindable_object

Slots* (Functions, Exclusively For Semantics):
ξ*_visco_chain_or_polygon* : $\mathbb{B}$

Slots (Functions):
at_most_constraint : $\mathbb{N} \setminus \{0\} \cup \{\perp\}$

Uses (Additional Bindings):
segments

Constraints (Predicates):
$\forall s\ (s \in segments \Rightarrow visco\_line(s))$
$at\_most\_constraint \Rightarrow |segments| \leq at\_most\_constraint$

Semantics:
ξ*_visco_chain_or_polygon* =
$(at\_most\_constraint \Rightarrow |segments(bound\_to)| \leq at\_most\_constraint) \wedge$
$\xi visco\_non\_enclosure\_object(self)$

visco_chain
Abstract Class
Is Subclass Of (Implies)
visco_chain_or_polygon visco_base_chain

Slots* (Functions, Exclusively For Semantics):
ξ*_visco_chain* : $\mathbb{B}$

Uses (Additional Bindings):
bound_to

Semantics:
ξ*_visco_chain* =
$geom\_chain(bound\_to) \wedge$
$\xi visco\_chain\_or\_polygon(self)$

visco_polygon
Abstract Class
Is Subclass Of (Implies)
visco_chain_or_polygon visco_base_polygon

Slots* (Functions, Exclusively For Semantics):
ξ*_visco_polygon* : $\mathbb{B}$

Uses (Additional Bindings):
bound_to

Semantics:
ξ*_visco_polygon* =
$geom\_polygon(bound\_to) \wedge$
$\xi visco\_chain\_or\_polygon(self)$

Geometrische VISCO-D-Objekte

visco_db_object
Abstract Class
Is Subclass Of (Implies)
visco_non_enclosure_object visco_on_transparency_object thematic_object

Slots* (Functions, Exclusively For Semantics):
ξ*_visco_db_object* : $\mathbb{B}$

Slots (Functions):
must_match_primary_db_object : $\mathbb{B}$

Uses (Additional Bindings):
bound_to

Constraints (Predicates):
$must\_match\_primary\_db\_object \Rightarrow$
$\forall m\ (dpo(self, m) \Rightarrow visco\_universe\_object(m))$
$\oplus(visco\_db\_point(self),$
$visco\_db\_line(self),$
$visc\_db\_chain(self),$
$visco\_db\_polygon(self),$
$visco\_db\_transparency(self))$

Semantics:
ξ*_visco_db_object* =
$(must\_match\_primary\_db\_object \wedge \neg visco\_db\_rubberband(self) \Rightarrow$
$db\_primary(bound\_to)) \wedge$
$(must\_match\_primary\_db\_object \wedge visco\_db\_rubberband(self) \wedge$
$geom\_chain(bound\_to) \Rightarrow$
$\forall s\ (dpo(s, bound\_to) \Rightarrow db\_primary(s))) \wedge$
$\xi visco\_non\_enclosure\_object(self)$

visco_db_point **Abstract Class** **Is Subclass Of (Implies)** *visco_db_object visco_point*
Slots* (Functions, Exclusively For Semantics): ξ*_visco_db_point* : $\mathbb{B}$
Uses (Additional Bindings): *bound_to*, *thematic_descriptor*
Constraints (Predicates): $\neg(visco\_db\_nail(self) \wedge visco\_db\_marble(self))$ $visco\_db\_nail(self) \vee visco\_db\_marble(self) \vee visco\_db\_origin(self)$
Semantics: $\xi\_visco\_db\_point =$ $db\_point(bound\_to) \wedge$ $thematic\_descriptor \subseteq thematic\_descriptor(bound\_to) \wedge$ $\xi\_visco\_db\_object(self) \wedge \xi\_visco\_point(self)$

visco_db_nail **Class** **Is Subclass Of (Implies)** *visco_db_point visco_nail*
Slots* (Functions, Exclusively For Semantics): ξ*_visco_db_nail* : $\mathbb{B}$
Semantics: $\xi\_visco\_db\_nail =$ $\xi\_visco\_db\_point(self) \wedge \xi\_visco\_nail(self)$

visco_db_origin **Class** **Is Subclass Of (Implies)** *visco_db_point visco_origin*
Slots* (Functions, Exclusively For Semantics): ξ*_visco_db_origin* : $\mathbb{B}$
Semantics: $\xi\_visco\_db\_origin =$ $\xi\_visco\_db\_point(self) \wedge \xi\_visco\_origin(self)$

visco_db_marble **Class** **Is Subclass Of (Implies)** *visco_db_point visco_marble*
Slots* (Functions, Exclusively For Semantics): ξ*_visco_db_marble* : $\mathbb{B}$
Semantics: $\xi\_visco\_db\_marble =$ $\xi\_visco\_db\_point(self) \wedge \xi\_visco\_marble(self)$

visco_db_line **Abstract Class** **Is Subclass Of (Implies)** *visco_db_object visco_line*
Slots* (Functions, Exclusively For Semantics): $\xi\_visco\_db\_line : \mathbb{B}$
Uses (Additional Bindings): p_1, p_2
Constraints (Predicates): $visco\_db\_point(p_1)$ $visco\_db\_point(p_2)$
Constraints (Predicates): $\oplus(visco\_db\_atomic\_segment(self), visco\_db\_rubberband(self))$
Semantics: $\xi\_visco\_db\_line =$ $\quad \xi\_visco\_db\_object(self) \wedge \xi\_visco\_line(self)$

visco_db_atomic_segment **Abstract Class** **Is Subclass Of (Implies)** *visco_db_line visco_atomic_segment*
Slots* (Functions, Exclusively For Semantics): $\xi\_visco\_db\_atomic\_segment : \mathbb{B}$
Uses (Additional Bindings): $bound\_to, thematic\_descriptor$
Constraints (Predicates): $\oplus(visco\_db\_antenna(self),$ $\quad visco\_db\_\leq\_antenna(self),$ $\quad visco\_db\_\geq\_antenna(self),$ $\quad visco\_db\_stick(self))$
Semantics: $\xi\_visco\_db\_atomic\_segment =$ $\quad db\_line(bound\_to) \wedge$ $\quad thematic\_descriptor \subseteq thematic\_descriptor(bound\_to) \wedge$ $\quad \xi\_visco\_db\_line(self) \wedge \xi\_visco\_atomic\_segment(self)$

visco_db_antenna **Class** **Is Subclass Of (Implies)** *visco_db_atomic_segment visco_antenna*
Slots* (Functions, Exclusively For Semantics): $\xi\_visco\_db\_antenna : \mathbb{B}$
Semantics: $\xi\_visco\_db\_antenna =$ $\quad \xi\_visco\_db\_atomic\_segment(self) \wedge \xi\_visco\_antenna(self)$

$visco\_db\_ \leq \_antenna$
Class
Is Subclass Of (Implies)
$visco\_db\_atomic\_segment\ visco\_ \leq \_antenna$
Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_db\_ \leq \_antenna : \mathbb{B}$
Semantics:
$\xi\_visco\_ \leq \_antenna =$
$\quad \xi\_visco\_db\_atomic\_segment(self) \wedge \xi\_visco\_ \leq \_antenna(self)$

$visco\_db\_ \geq \_antenna$
Class
Is Subclass Of (Implies)
$visco\_db\_atomic\_segment\ visco\_ \geq \_antenna$
Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_db\_ \geq \_antenna : \mathbb{B}$
Semantics:
$\xi\_visco\_db\_ \geq \_antenna =$
$\quad \xi\_visco\_db\_atomic\_segment(self) \wedge \xi\_visco\_ \geq \_antenna(self)$

$visco\_db\_stick$
Class
Is Subclass Of (Implies)
$visco\_db\_atomic\_segment\ visco\_stick$
Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_db\_stick : \mathbb{B}$
Semantics:
$\xi\_visco\_db\_stick =$
$\quad \xi\_visco\_db\_atomic\_segment(self) \wedge \xi\_visco\_stick(self)$

$visco\_db\_rubberband$
Class
Is Subclass Of (Implies)
$visco\_db\_line\ visco\_rubberband$
Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_db\_rubberband : \mathbb{B}$
Uses (Additional Bindings):
$bound\_to, thematic\_descriptor$
Semantics:
$\xi\_visco\_db\_rubberband =$
$\quad \forall s\ (dpo(s, bound\_to) \Rightarrow$
$\qquad db\_line(s) \wedge$
$\qquad thematic\_descriptor \subseteq thematic\_descriptor(s)) \wedge$
$\quad \xi\_visco\_db\_line(self) \wedge \xi\_visco\_rubberband(self)$

visco_db_chain **Class** **Is Subclass Of (Implies)** *visco_db_object visco_chain*
Slots* (Functions, Exclusively For Semantics): ξ*_visco_db_rubberband* : $\mathbb{B}$
Uses (Additional Bindings): *segments*, *bound_to*, *thematic_descriptor*
Constraints (Predicates): $\forall s : s \in segments \Rightarrow visco\_db\_line(s)$
Semantics: ξ*_visco_db_chain* = *db_chain*(*bound_to*) $\wedge$ *thematic_descriptor* $\subseteq$ *thematic_descriptor*(*bound_to*) $\wedge$ ξ*_visco_db_object*(*self*) $\wedge$ ξ*_visco_chain*(*self*)

visco_db_polygon **Class** **Is Subclass Of (Implies)** *visco_db_object visco_polygon*
Slots* (Functions, Exclusively For Semantics): ξ*_visco_db_polygon* : $\mathbb{B}$
Uses (Additional Bindings): *segments*, *bound_to*, *thematic_descriptor*
Constraints (Predicates): $\forall s : s \in segments \Rightarrow visco\_db\_line(s)$
Semantics: ξ*_visco_db_polygon* = *db_polygon*(*bound_to*) $\wedge$ *thematic_descriptor* $\subseteq$ *thematic_descriptor*(*bound_to*) $\wedge$ ξ*_visco_db_object*(*self*) $\wedge$ ξ*_visco_polygon*(*self*)

visco_db_transparency **Class** **Is Subclass Of (Implies)** *visco_db_object visco_transparency*
Current Implementation Limitations (Predicates): Not implemented!
Slots* (Functions, Exclusively For Semantics): ξ*_visco_db_transparency* : $\mathbb{B}$
Uses (Additional Bindings): *segments*, *bound_to*, *thematic_descriptor*
Constraints (Predicates): $\forall p : p \in segments \Rightarrow visco\_db\_object(p)$
Semantics: ξ*_visco_db_transparency* = *db_aggregate*(*bound_to*) $\wedge$ *thematic_descriptor* $\subseteq$ *thematic_descriptor*(*bound_to*) $\wedge$ ξ*_visco_db_object*(*self*) $\wedge$ ξ*_visco_transparency*(*self*)

Geometrische VISCO-Hilfsobjekte

visco_universe_object
Abstract Class
Is Subclass Of (Implies)
visco_non_enclosure_object visco_on_transparency_object

Slots* (Functions, Exclusively For Semantics):
ξ*_visco_universe_object* : $\mathbb{B}$

Current Implementation Limitations (Predicates):
direct_instantiable

Constraints (Predicates):
$\oplus$(*visco_universe_nail*(*self*),
visco_universe_marble(*self*),
visco_universe_origin(*self*),
visco_universe_antenna(*self*),
*visco_universe_*$\leq$*_antenna*(*self*),
*visco_universe_*$\geq$*_antenna*(*self*),
visco_universe_stick(*self*),
visc_universe_chain(*self*),
visco_universe_polygon(*self*),
visco_universe_transparency(*self*))

Semantics:
ξ*_visco_universe_object* =
ξ*_visco_non_enclosure_object*(*self*)

visco_universe_nail
Class
Is Subclass Of (Implies)
visco_universe_object visco_nail

Slots* (Functions, Exclusively For Semantics):
ξ*_visco_universe_nail* : $\mathbb{B}$

Semantics:
ξ*_visco_universe_nail* =
ξ*_visco_universe_object*(*self*) $\wedge$ ξ*_visco_nail*(*self*)

visco_universe_origin
Class
Is Subclass Of (Implies)
visco_universe_object visco_origin

Slots* (Functions, Exclusively For Semantics):
ξ*_visco_universe_origin* : $\mathbb{B}$

Semantics:
ξ*_visco_universe_origin* =
ξ*_visco_universe_object*(*self*) $\wedge$ ξ*_visco_origin*(*self*)

visco_universe_marble
Class
Is Subclass Of (Implies)
visco_universe_object visco_marble
Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_universe\_marble : \mathbb{B}$
Semantics:
$\xi\_visco\_universe\_marble =$
$\quad \xi\_visco\_universe\_object(self) \wedge \xi\_visco\_marble(self)$

visco_universe_antenna
Class
Is Subclass Of (Implies)
visco_universe_object visco_antenna
Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_universe\_antenna : \mathbb{B}$
Semantics:
$\xi\_visco\_universe\_antenna =$
$\quad \xi\_visco\_universe\_object(self) \wedge \xi\_visco\_antenna(self)$

*visco_universe_* $\leq$ *_antenna*
Class
Is Subclass Of (Implies)
*visco_universe_object visco_* $\leq$ *_antenna*
Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_universe\_ \leq \_antenna : \mathbb{B}$
Semantics:
$\xi\_visco\_universe\_ \leq \_antenna =$
$\quad \xi\_visco\_universe\_object(self) \wedge \xi\_visco\_ \leq \_antenna(self)$

*visco_universe_* $\geq$ *_antenna*
Class
Is Subclass Of (Implies)
*visco_universe_object visco_* $\geq$ *_antenna*
Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_universe\_ \geq \_antenna : \mathbb{B}$
Semantics:
$\xi\_visco\_universe\_ \geq \_antenna =$
$\quad \xi\_visco\_universe\_object(self) \wedge \xi\_visco\_ \geq \_antenna(self)$

visco_universe_stick
Class
Is Subclass Of (Implies)
visco_universe_object visco_stick
Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_universe\_stick : \mathbb{B}$
Semantics:
$\xi\_visco\_universe\_stick =$
$\quad \xi\_visco\_universe\_object(self) \wedge \xi\_visco\_stick(self)$

visco_universe_chain **Class** **Is Subclass Of (Implies)** *visco_universe_object visco_chain*
Slots* (Functions, Exclusively For Semantics): ξ_*visco_universe_chain* : $\mathbb{B}$
Semantics: ξ_*visco_universe_chain* = $\quad \xi$_*visco_universe_object*$(self) \wedge \xi$_*visco_chain*$(self)$

visco_universe_polygon **Class** **Is Subclass Of (Implies)** *visco_universe_object visco_polygon*
Slots* (Functions, Exclusively For Semantics): ξ_*visco_universe_polygon* : $\mathbb{B}$
Semantics: ξ_*visco_universe_polygon* = $\quad \xi$_*visco_universe_object*$(self) \wedge \xi$_*visco_polygon*$(self)$

visco_universe_transparency **Class** **Is Subclass Of (Implies)** *visco_universe_object visco_transparency*
Slots* (Functions, Exclusively For Semantics): ξ_*visco_universe_transparency* : $\mathbb{B}$
Semantics: ξ_*visco_universe_transparency* = $\quad \xi$_*visco_universe_object*$(self) \wedge \xi$_*visco_transparency*$(self)$

VISCO-Gebiete

visco_enclosure

Abstract Class

Is Subclass Of (Implies)

visco_on_transparency_object geom_object

Slots* (Functions, Exclusively For Semantics):

$\xi\_visco\_enclosure : \mathbb{B}$

Slots (Functions):

$opaque : \mathbb{B}$

$at\_most\_constraint : \mathbb{N} \setminus \{0\} \cup \{\bot\}$

Uses (Additional Bindings):

$parts, boundary$

Constraints (Predicates):

$$
\begin{aligned}
&parts = \emptyset \\
&boundary = \emptyset \\
&\forall other\ (visco\_non\_enclosure\_object(other) \Rightarrow \\
&\qquad (inside(other, self) \wedge \\
&\qquad\ (before(other, self) \wedge fully\_visible\_from\_to(other, other, self) \vee \\
&\qquad\ \ before(self, other) \wedge fully\_visible\_from\_to(other, self, other)) \\
&\qquad\ \Leftrightarrow \\
&\qquad\ visible\_contains(self, other))) \\
&at\_most\_constraint \Rightarrow \\
&\quad |\{o \mid primary(o) \wedge visco\_db\_object(o) \wedge visible\_contains(self, o)\}| \\
&\qquad \leq at\_most\_constraint \\
&\oplus(visco\_derived\_enclosure(self), \\
&\quad visco\_derived\_enclosure(self), \\
&\quad visco\_sketched\_enclosure(self))
\end{aligned}
$$

Current Implementation Limitations (Predicates):

$at\_most\_constraint = \bot$

Semantics:

$$
\begin{aligned}
&\xi\_visco\_enclosure = \\
&\quad \xi\_visco\_on\_transparency\_object(self)
\end{aligned}
$$

visco_derived_enclosure **Abstract Class** **Is Subclass Of (Implies)** *visco_enclosure*
Slots* (Functions, Exclusively For Semantics): ξ*_visco_derived_enclosure* : $\mathbb{B}$
Slots (Functions): *argument* : visco_point ∪ visco_line ∪ visco_chain ∪ visco_polygon
Constraints (Predicates): *before*(*argument*, *self*) *fully_visible_from_to*(*argument*, *argument*, *self*) ⊕(*visco_inner_enclosure*(*self*), *visco_outer_enclosure*(*self*), *visco_epsilon_enclosure*(*self*), *visco_epsilon*$^\oplus$*_enclosure*(*self*), *visco_epsilon*$^\ominus$*_enclosure*(*self*))
Semantics: ξ*_visco_derived_enclosure* = ξ*_visco_enclosure*(*self*)

visco_sketched_enclosure **Class** **Is Subclass Of (Implies)** *visco_enclosure*
Current Implementation Limitations (Predicates): Not implemented!
Slots* (Functions, Exclusively For Semantics): ξ*_visco_sketched_enclosure* : $\mathbb{B}$
Slots (Functions): *polygon* : geom_polygon_with_optional_holes
Uses (Additional Bindings): *on_transparency*, *at_most_constraint*
Constraints (Predicates): *interior* = *interior*(*polygon*)
Semantics: ξ*_visco_sketched_enclosure* = **let** *matrix* = *get_matrix_for_transparency*(*on_transparency*) *org_matrix* = **T**(−*x*(*origin*(*on_transparency*)), −*y*(*origin*(*on_transparency*))) ∃ *domain_polygon* (*equal_matrix_images*(*domain_polygon*, *polygon*, *matrix*, *org_matrix*) ∧ **let** *contained_objects* = {*o* | *geom_object*(*o*) ∧ *contains*(*domain_polygon*, *o*)} ∀*other* (*visco_non_enclosure_object*(*other*) ∧ *visible_contains*(*self*, *other*) ⇒ *bound_to*(*other*) ∈ *contained_objects* ∧ (*at_most_constraint* ⇒ |{*o* | *db_primary*(*o*) ∧ *o* ∈ *contained_objects*}| ≤ *at_most_constraint*))) ∧ ξ*_visco_enclosure*(*self*)

stattdessen:

visco_sketched_enclosure
Class
Is Subclass Of (Implies)
visco_enclosure

Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_sketched\_enclosure : \mathbb{B}$

Slots (Functions):
polygon : geom_polygon

Uses (Additional Bindings):
on_transparency, *at_most_constraint*

Constraints (Predicates):
$\neg visco\_polygon(self) \wedge$
$interior = interior(polygon)$

Semantics:
$\xi\_visco\_sketched\_enclosure =$
let $matrix = get\_matrix\_for\_transparency(on\_transparency)$
$org\_matrix = \mathbf{T}(-x(origin(on\_transparency)),$
$-y(origin(on\_transparency)))$
$\exists\, domain\_polygon$
$(equal\_matrix\_images(domain\_polygon, polygon, matrix, org\_matrix) \wedge$
let $contained\_objects = \{o \mid geom\_object(o) \wedge contains(domain\_polygon, o)\}$
$\forall other$
$(visco\_non\_enclosure\_object(other) \wedge visible\_contains(self, other) \Rightarrow$
$bound\_to(other) \in contained\_objects \wedge$
$(at\_most\_constraint \Rightarrow$
$|\{o \mid db\_primary(o) \wedge o \in contained\_objects\}|$
$\leq at\_most\_constraint))) \wedge$
$\xi\_visco\_enclosure(self)$

visco_inner_enclosure
Class
Is Subclass Of (Implies)
visco_derived_enclosure

Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_inner\_enclosure : \mathbb{B}$

Uses (Additional Bindings):
interior, *argument*, *at_most_constraint*

Constraints (Predicates):
$interior = interior(argument)$
$visco\_polygon(argument)$

Semantics:
$\xi\_visco\_inner\_enclosure =$
let $contained\_objects = \{o \mid geom\_object(o) \wedge contains(bound\_to(argument), o)\}$
$\forall other$
$(visco\_non\_enclosure\_object(other) \wedge visible\_contains(self, other) \Rightarrow$
$bound\_to(other) \in contained\_objects) \wedge$
$(at\_most\_constraint \Rightarrow$
$|\{o \mid db\_primary(o) \wedge o \in contained\_objects\}|$
$\leq at\_most\_constraint) \wedge$
$\xi\_visco\_derived\_enclosure(self)$

visco_outer_enclosure
Class
Is Subclass Of (Implies)
visco_derived_enclosure

Slots* (Functions, Exclusively For Semantics):
$\xi\_visco\_outer\_enclosure : \mathbb{B}$

Uses (Additional Bindings):
interior, *argument*, *at_most_constraint*, *on_transparency*

Constraints (Predicates):
$interior = interior(rectangle(on\_transparency(argument))) \setminus shape(argument)$
$visco\_polygon(argument)$

Semantics:
$\xi\_visco\_outer\_enclosure =$
let $rectangle = rectangle(on\_transparency)$
$matrix = get\_matrix\_for\_transparency(on\_transparency)$
$org\_matrix = \mathbf{T}(-x(origin(on\_transparency)),$
$-y(origin(on\_transparency)))$
$\exists\, domain\_rectangle$
$(\, geom\_rectangle(domain\_rectangle) \wedge$
$equal\_matrix\_images(domain\_rectangle, rectangle, matrix, org\_matrix) \wedge$
let $excluded\_objects =$
$\{o \mid geom\_object(o) \wedge excludes(bound\_to(argument), o) \wedge contains(domain\_rectangle, o)\}$
$\forall other$
$(\, visco\_non\_enclosure\_object(other) \wedge visible\_contains(self, other) \Rightarrow$
$bound\_to(other) \in excluded\_objects \wedge$
$(at\_most\_constraint \Rightarrow$
$|\{o \mid db\_primary(o) \wedge o \in excluded\_objects\}|$
$\leq at\_most\_constraint))) \wedge$
$\xi\_visco\_derived\_enclosure(self)$

visco_epsilon_enclosure

Class

Is Subclass Of (Implies)

visco_derived_enclosure

Slots* (Functions, Exclusively For Semantics):

$\xi\_visco\_epsilon\_enclosure : \mathbb{B}$

Uses (Additional Bindings):

$interior, argument, at\_most\_constraint, on\_transparency$

Slots (Functions):

$radius : \mathbb{R}^{\oplus} \setminus \{0\}$

Constraints (Predicates):

$interior = create\_epsilon\_enclosure\_of(argument, radius)$

Semantics:

$$
\begin{array}{l}
\xi\_visco\_epsilon\_enclosure = \\
\quad \textbf{let}\ rectangle = rectangle(on\_transparency) \\
\qquad matrix = get\_matrix\_for\_transparency(on\_transparency) \\
\qquad org\_matrix = \mathbf{T}(-x(origin(on\_transparency)), \\
\qquad\qquad\qquad -y(origin(on\_transparency))) \\
\quad\ \exists\, image\_of\_rectangle, image\_of\_argument \\
\qquad (geom\_rectangle(image\_of\_rectangle) \wedge \\
\qquad\ equal\_matrix\_images(rectangle, image\_of\_rectangle, org\_matrix, \mathbf{E}) \wedge \\
\qquad\ equal\_matrix\_images(bound\_to(argument), image\_of\_argument, matrix, \mathbf{E}) \wedge \\
\qquad\ \textbf{let}\ epsiloned\_objects = \\
\qquad\qquad \{\, o \mid \exists\, image\_o\ (equal\_matrix\_images(o, image\_o, matrix, \mathbf{E}) \wedge \\
\qquad\qquad\qquad\qquad \Delta(image\_of\_argument, image\_o) \leq radius \wedge \\
\qquad\qquad\qquad\qquad contains(image\_of\_rectangle, image\_o))\} \\
\qquad\ \forall other \\
\qquad\quad (visco\_non\_enclosure\_object(other) \wedge visible\_contains(self, other) \Rightarrow \\
\qquad\qquad bound\_to(other) \in epsiloned\_objects \wedge \\
\qquad\qquad (at\_most\_constraint \Rightarrow \\
\qquad\qquad\quad |\{o \mid db\_primary(o) \wedge o \in epsiloned\_objects\}| \\
\qquad\qquad\qquad \leq at\_most\_constraint))) \wedge \\
\quad \xi\_visco\_derived\_enclosure(self)
\end{array}
$$

visco_epsilon$^{\oplus}$_enclosure

Class

Is Subclass Of (Implies)

visco_derived_enclosure

Slots* (Functions, Exclusively For Semantics):

ξ*_visco_epsilon*$^{\oplus}$*_enclosure* : $\mathbb{B}$

Uses (Additional Bindings):

interior, *argument*, *at_most_constraint*, *on_transparency*

Constraints (Predicates):

$interior = create\_epsilon\_enclosure\_of(argument, radius) \setminus shape(argument)$

$visco\_polygon(argument)$

Semantics:

$\xi visco\_epsilon^{\oplus}\_enclosure =$

let $rectangle = rectangle(on\_transparency)$

$matrix = get\_matrix\_for\_transparency(on\_transparency)$

$org\_matrix = \mathbf{T}(-x(origin(on\_transparency)),$

$-y(origin(on\_transparency)))$

$\exists\, image\_of\_rectangle, image\_of\_argument$

$(geom\_rectangle(image\_of\_rectangle) \wedge$

$equal\_matrix\_images(rectangle, image\_of\_rectangle, org\_matrix, \mathbf{E}) \wedge$

$equal\_matrix\_images(bound\_to(argument), image\_of\_argument, matrix, \mathbf{E}) \wedge$

let $epsiloned\_objects =$

$\{\, o \mid \exists\, image\_o\ (equal\_matrix\_images(o, image\_o, matrix, \mathbf{E}) \wedge$

$\Delta(image\_of\_argument, image\_o) \leq radius \wedge$

$contains(image\_of\_rectangle, image\_o) \wedge$

$excludes(image\_of\_argument, o))\}$

$\forall other$

$(visco\_non\_enclosure\_object(other) \wedge visible\_contains(self, other) \Rightarrow$

$bound\_to(other) \in epsiloned\_objects \wedge$

$(at\_most\_constraint \Rightarrow$

$|\{o \mid db\_primary(o) \wedge o \in epsiloned\_objects\}|$

$\leq at\_most\_constraint))) \wedge$

$\xi visco\_derived\_enclosure(self)$

$visco\_epsilon^{\ominus}\_enclosure$

Class

Is Subclass Of (Implies)

$visco\_derived\_enclosure$

Slots* (Functions, Exclusively For Semantics):

$\xi\_visco\_epsilon^{\ominus}\_enclosure : \mathbb{B}$

Uses (Additional Bindings):

$interior, argument, at\_most\_constraint, on\_transparency$

Constraints (Predicates):

$interior = create\_epsilon\_enclosure\_of(argument, radius) \cap interior(argument)$

$visco\_polygon(argument)$

Semantics:

$$
\begin{array}{l}
\xi\_visco\_epsilon^{\ominus}\_enclosure = \\
\quad \textbf{let } matrix = get\_matrix\_for\_transparency(on\_transparency) \\
\qquad org\_matrix = \mathbf{T}(-x(origin(on\_transparency)), \\
\qquad\qquad\qquad -y(origin(on\_transparency))) \\
\quad \exists\, image\_of\_rectangle, image\_of\_argument \\
\qquad (geom\_rectangle(image\_of\_rectangle) \wedge \\
\qquad\; equal\_matrix\_images(bound\_to(argument), image\_of\_argument, matrix, \mathbf{E}) \wedge \\
\qquad\; \textbf{let } epsiloned\_objects = \\
\qquad\qquad \{o \mid \exists\, image\_o\ (equal\_matrix\_images(o, image\_o, matrix, \mathbf{E}) \wedge \\
\qquad\qquad\qquad\qquad \Delta(image\_of\_argument, image\_o) \leq radius \wedge \\
\qquad\qquad\qquad\qquad contains(image\_of\_argument, image\_o))\} \\
\qquad\; \forall other \\
\qquad\qquad (visco\_non\_enclosure\_object(other) \wedge visible\_contains(self, other) \Rightarrow \\
\qquad\qquad\quad bound\_to(other) \in epsiloned\_objects \wedge \\
\qquad\qquad\quad (at\_most\_constraint \Rightarrow \\
\qquad\qquad\qquad |\{o \mid db\_primary(o) \wedge o \in epsiloned\_objects\}| \\
\qquad\qquad\qquad\quad \leq at\_most\_constraint))) \wedge \\
\quad \xi\_visco\_derived\_enclosure(self)
\end{array}
$$

Literaturverzeichnis

[AO93] D. Abel and B. C. Ooi, editors. *Advances in Spatial Databases – Third International Symposium, SSD '93*, volume 692 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[AP95] M. A. Aufaure-Portier. A High Level Interface for GIS. *Journal of Visual Languages and Computing*, 6:167–182, 1995.

[ASU88] A. V. Aho, R. Sethi und J.D. Ullman. *Compilerbau*. Addison-Wesley, 1988.

[Bar95] N. Bartelme. *Geoinformatik – Modelle, Strukturen, Funktionen*. Springer-Verlag, 1995.

[BCCL91] C. Battini, T. Catarci, M. F. Costabile, and S. Levialdi. Visual Stratgies for Querying Databases. In *1991 IEEE Workshop on Visual Languages* [VL91], pages 183–189.

[BCLM95] P. Bottoni, M. F. Costabile, S. Levialdi, and P. Mussio. Formalising Visual Languages. In *1995 IEEE Symposium on Visual Languages* [VL95], pages 45–52.

[BCLM97] P. Bottoni, M. F. Costabile, S. Levialdi, and P. Mussio. From Visual Language Specification to Legal Visual Interaction. In *1997 IEEE Symposium on Visual Languages* [VL97], pages 234–241.

[BE96] H. T. Bruns and M. J. Egenhofer. Similarity of Spatial Scenes. In Kraak and Molenaar [KM96]. Session 4A.

[BF91] R. Bill und D. Fritsch. *Grundlagen der Geo-Informationssysteme*. Wichmann, 1991.

[BF97] T. Barkowsky and Ch. Freksa. Cognitive Requirements on Making and Interpreting Maps. In *Conference on Spatial Information Theory, COSIT '97*, pages 347–361, 1997.

[Bib93] W. Bibel et al. *Wissensrepräsentation und Inferenz*. Vieweg & Sohn Verlagsgesellschaft, 1993.

[BMPS+91] R. J. Brachman, D.L. McGuinness, P.F. Patel-Schneider, L.A. Reswnick, and A. Borgida. Living with CLASSIC - When and How to Use a KL-ONE-like Language. In J. F. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 401–456. Morgan Kaufmann Publishers, 1991.

[Boo94] G. Booch. *Object-oriented Analysis and Design with Applications, 2nd Edition*. Benjamin/Cummings, 1994.

[Bor81] A. H. Borning. The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory. *ACM Transactions on Progamming Languages and Systems*, 3(4):353–387, Oktober 1981. Auch zu finden in [Gli90b, S. 416–449].

[BPS94] A. Del Bimbo, P. Pala, and S. Santini. Visual Image Retrieval by Elastic Deformation of Object Sketches. In *1994 IEEE Symposium on Visual Languages* [VL94], pages 216–223.

[CCLB97] T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini. Visual Query Systems for Databases: A Survey. *Journal of Visual Languages and Computing*, 8:215–260, 1997.

[CCR93] Z. Cui, A. G. Cohn, and D. A. Randell. Qualitative and Topological Relationsships in Spatial Databases. In Abel and Ooi [AO93], pages 296–315.

[CDF97] E. Clementini and P. Di Felici. Approximate topological relations. *International Journal of Approximate Reasoning*, 16:173–204, 1997.

[CDFC95] E. Clementini, P. Di Felici, and G. Califano. Composite Regions in Topological Queries. *Information Systems*, 20(7):579–594, 1995.

[CDFvO93] E. Clementini, P. Di Felici, and P. van Oosterom. A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In Abel and Ooi [AO93], pages 277–295.

[Cha87] S. K. Chang. Visual Languages: A Tutorial and Survey. *IEEE Software*, pages 29–39, Januar 1987. Auch zu finden in [Gli90b, S. 7–17].

[Cha90] S. K. Chang, editor. *Visual Programming Systems*. Prentice-Hall, 1990.

[CIL86] S. K. Chang, T. Ichikawa, and P. A. Ligomenides, editors. *Visual Languages*. Plenum Press, 1986.

[CM94] D. Calcinelli and M. Mainguenaud. Cigales, a Visual Query Language for a Geographical Information System: the User Interface. *Journal of Visual Languages and Computing*, 5:113–132, 1994.

[CM95] S. S. Chok and K. Marriott. Automatic Construction of User Interfaces from Constraint Multiset Grammars. In *1995 IEEE Symposium on Visual Languages* [VL95], pages 242–249.

[CMS91] T. Catarci, A. Massari, and G. Santucci. Iconic and Diagrammtic Interfaces: An Integrated Approach. In *1991 IEEE Workshop on Visual Languages* [VL91], pages 199–204.

[Com94] VL Steering Committee. Ten Years of Visual Language Research. In *1994 IEEE Symposium on Visual Languages* [VL94], pages 196–205.

[CTBY89] S. K. Chang, M. J. Tauber, Y. Bing, and J. S. Yu. A Visual Language Compiler. *IEEE Transactions on Software Engineering*, 15(5):506–525, Mai 1989. Auch zu finden in [Gli90a, S. 518–537].

[Dat95] C. J. Date. *An Introduction to Database Systems*. Addison-Wesley, 6. edition, 1995.

[Dud86] Dudenverlag. *Duden: Die Rechtschreibung, Band 1, 19. Auflage*, 1986.

[Dud93] Dudenverlag. *Duden: Informatik, 2. Auflage*, 1993.

[Dud94] Dudenverlag. *Duden: Rechnen und Mathematik, 5. Auflage*, 1994.

[Ege90] M. J. Egenhofer. Manipulation the Graphical Representation of Query Results in Geographic Information Systems. In *1990 IEEE Workshop on Visual Languages* [VL90], pages 119–124.

[Ege91] M. J. Egenhofer. Reasoning about Binary Topological Relations. In Günther and Schek [GS91], pages 143–160.

[Ege92] M. J. Egenhofer. Why not SQL! *International Journal on Geographical Information Systems*, 6:71–85, 1992.

[Ege96a] M. J. Egenhofer. Multi-Modal Spatial Querying. In Kraak and Molenaar [KM96]. Session 12B.

[Ege96b] M. J. Egenhofer. Spatial-Query-by-Sketch. In *1996 IEEE Symposium on Visual Languages* [VL96], pages 60–67.

[EH93] M. J. Egenhofer and J. R. Herring. *Querying a Geographical Information System*, pages 124–135. In Medyckyj-Scott and Hearnshaw [MSH93], 1993.

[ES97] M. Erwig and M. Schneider. Vague Regions. In Scholl and Voisard [SV97], pages 298–320.

[FB95] Ch. Freksa and T. Barkowsky. On the Relation between Spatial Concepts and Geographic Objects. In P. Borrough and A. Frank, editors, *Geographic objects with indeterminate boundaries*, pages 109–121. Taylor and Franics, 1995.

[FB96] R.W. Floyd und R. Beigel. *Die Sprache der Maschinen*. International Thomson Publishing GmbH, 1996.

[FDFH96] J. D. Foley, A. van Dam, S. K. Feiner, and F. H. Hughes. *Computer Graphics – Principles and Practice, 2nd Edition in C*. Addison-Wesley, 1996.

[FH90] C. Freksa und C. Habel (Hrsg.). *Repräsentation und Verarbeitung räumlichen Wissens*, Band Nr. 245 der *Informatik-Fachberichte*. Springer-Verlag, 1990.

[Fis79] Fischer Taschenbuch Verlag. *Das neue Fischer Lexikon in Farbe*, 1979.

[FK91] H. Fujii and R. R. Korfhage. Features and a Model for Icon Morphological Transformation. In *1991 IEEE Workshop on Visual Languages* [VL91], pages 240–245.

[Fra93] A. U. Frank. *The Use of Geographical Information Systems: The User Interface is the System*, pages 3–14. In Medyckyj-Scott and Hearnshaw [MSH93], 1993.

[Fra94] Franz Inc. *Common Lisp Interface Manager (CLIM) 2.0: User Guide*, 1994.

[Fre87] Freie und Hansestadt Hamburg, Baubehörde – Vermessungsamt. *Auszug aus dem Konzept für die Einrichtung der Digitalen Stadtgrundkarte in Hamburg*, 1987.

[Fre88] Ch. Freksa. Intrinsische vs. extrinische Repräsentation zum Aufgabenlösen oder die Verwandlung von Wasser in Wein. In G. Heyer, J. Krems und G. Görz (Hrsg.), *Wissensarten und ihre Darstellung – Beiträge aus Philosophie, Psychologie, Informatik und Linguistik*, Band Nr. 169 der *Informatik-Fachberichte*. Springer-Verlag, 1988.

[Fre91] Ch. Freksa. Wissensdarstellung und Kognitionsforschung. In P. Struß (Hrsg.), *Wissensrepräsentation*, Seiten 61–69. R. Oldenburg Verlag, 1991.

[Fre92a] Freie und Hansestadt Hamburg, Baubehörde – Vermessungsamt. *Beschreibung der Datenschnittstelle für Datenlieferungen der Digitalen Stadtkarte (DISK)*, 1992. Entwurf.

[Fre92b] Ch. Freksa. Temporal reasoning based on semi-intervals. *Artificial Intelligence*, 54:199–227, 1992.

[Fre92c] Ch. Freksa. Using Orientation Information for Qualitative Spatial Reasoning. In A. U. Frank, I. Campari, and U. Formentini, editors, *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space. International Conference GIS – From Space to Territory*, number 639 in Lecture Notes in Computer Science, pages 162–178. Springer-Verlag, 1992.

[Fre96a] Freie und Hansestadt Hamburg, Baubehörde – Vermessungsamt. *Digitale Karten und Stadtpläne von Hamburg*, 1996. Broschüre.

[Fre96b] Freie und Hansestadt Hamburg, Baubehörde – Vermessungsamt. *Digitale Stadtkarte von Hamburg (DISK) und zugeordnete Daten: Verzeichnis der Ebenen und Schlüssel*, 1996.

[FZ92] Ch. Freksa and K. Zimmermann. On the Utilization of Spatial Structures for Cognitively Plausible and Efficient Reasoning. In *Proceedings of the IEEE Conference on Systems, Man, Cybernetics*, Chicagao, 1992.

[GHJV96] E. Gamma, R. Helm, R. Johnson und J. Vlissides. *Entwurfsmuster*. Addison-Wesley, 1996. Deutsche Übersetzung von D. Riehle.

[Gli90a] E. P. Glinert, editor. *Visual Programming Environments – Applications and Issues*. IEEE Computer Society Press, 1990.

[Gli90b] E. P. Glinert, editor. *Visual Programming Environments – Paradigms and Systems*. IEEE Computer Society Press, 1990.

[GM95a] F. Gardin and B. Meltzer. *Analogical Representations of Naive Physics*, pages 669—689. In Glasgow et al. [GNC95], 1995.

[GM95b] B. Gerken und R. Meyer. Geographische Informationssysteme – Eine Einführung für Informatiker. Mitteilung Nr. 249 des Fachbereichs Informatik der Universität Hamburg, 1995.

[GNC95] J. Glasgow, N. H. Narayanan, and B. Chandrasekaran, editors. *Diagrammatic Reasoning*. AAAI Press / The MIT-Press, 1995.

[Gol91] E. J. Golin. Parsing Visual Languages with Picture Layout Grammars. *Journal of Visual Languages and Computing*, 5:371–393, 1991.

[GPP95] M. Grigni, D. Papadias, and C. Papadimitriou. Topological Inference. In C. Mellish, editor, *14th International Joint Conference on Artificial Intelligence*, pages 901–906, 1995.

[GR89] E. J. Golin and S. P. Reiss. The Specification of Visual Language Syntax. In *1989 IEEE Workshop on Visual Languages* [VL89], pages 105–110.

[Gör93] G. Görz (Hrsg.). *Einführung in die künstliche Intelligenz*. Addison-Wesley, 1993.

[GR95] V. N. Gudivada and V. V. Raghavan. Design and Evaluation of Algorithms for Image Retrieval by Spatial Similarity. *ACM Transactions on Information Systems*, 13(2):115–144, 1995.

[Gra90] M. Graf. Visual Programming and Visual Languages: Lessons Learned in the Trenches. In *1990 IEEE Workshop on Visual Languages* [VL90]. Auch zu finden in [Gli90a, S. 452–454].

[Gra94] P. Graham. *On Lisp*. Prentice-Hall, 1994.

[Gra96] P. Graham. *ANSI Common Lisp*. Prentice-Hall, 1996.

[GS91] O. Günther and H.-J. Schek, editors. *Advances in Spatial Databases – Second Symposium, SSD '91*, volume 525 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[GS93] R. Güting and M. Schneider. Realms: A foundation for Spatial Data Types in Database Systems. In Abel and Ooi [AO93], pages 14–35.

[GT84] E. P. Glinert and S. L. Tanimoto. Pict: An Interactive Graphical Progamming Environment. *Computer*, pages 7–25, November 1984. Auch zu finden in [Gli90b, S. 265–283].

[Güt94] R. H. Güting. An Introduction to Spatial Database Systems. *Special Issue on Spatial Database Systems of the Very Large Databases Journal*, 3(4), September 1994.

[Haa95] V. Haarslev. Formal Semantics of Visual Languages using Spatial Reasoning. In *1995 IEEE Symposium on Visual Languages* [VL95], pages 156–163.

[Haa96] V. Haarslev. Using Description Logic for Reasoning about Diagrammatical Notations. In L. Padgham et al., editor, *Proceedings of the International Workshop on Description Logics*, pages 124–128. AAAI Press, 1996.

[Hab87] Ch. Habel. Repräsentation räumlichen Wissens. In G. Rahmstorf (Hrsg.), *Wissensrepräsentation in Expertensystemen*, Band Nr. 172 der *Informatik-Fachberichte*. Springer-Verlag, 1987.

[Hab96] Ch. Habel. Grundlagen der Verarbeitung von Wissen über Raum, Zeit, Ereignisse. Skriptum zur gleichnamigen Vorlesung an der Universität Hamburg, Fachbereich Informatik, 1996.

[Har88] D. Harel. On Visual Formalisms. *Communications of the ACM*, 31(5):514–530, May 1988. Auch zu finden in [Gli90b, S. 171–187] und [GNC95].

[Hei95] J. L. Hein. *Discrete Structures, Logic, and Computability*. Jones and Barlett Publishers International, 1995.

[Heu97] A. Heuer. *Objektorientierte Datenbanken*. Addison-Wesley, 2. Auflage, 1997.

[HHP93] C. Habel, M. Herweg und S. Pribbenow. Wissen über Raum und Zeit. In Görz [Gör93], Seiten 139–204.

[HK92] K. Hirata and T. Kato. Query By Visual Example – Content-Bases Image Retrieval. In *Advances in Database Technology – EDBT '92, 3rd International Conference on Extending Database Technology*, volume 580 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[HM90] R. Helm and K. Marriott. A Declarative Specification of Visual Languages. In *1990 IEEE Workshop on Visual Languages* [VL90], pages 98–103.

[HM95] D. Hernández and A. Mukerjee. Representation of Spatial Knowledge. In *Proceedings of the 14th International Joint Conference on Artficial Intelligence (IJCAI-95)*, 1995.

[HM97a] V. Haarslev and R. Möller. SBox: A Qualitative Spatial Reasoner – Progress Report. In L. Ironi, editor, *11th International Workshop on Qualitative Reasoning*, pages 105–113, 1997.

[HM97b] V. Haarslev and R. Möller. Spatioterminological Reasoning: Subsumption Based on Geometrical Inference. In *Proceedings of the International Workshop on Description Logics*, 1997.

[HMS94] V. Haarslev, R. Möller, and C. Schröder. Combining Spatial and Terminological Resoning. In B. Nebel and L. Dreschler-Fischer, editors, *KI-94: Advances in Artificial Intelligence – Proc. 18th German Annual Conference on Artificial Intelligence*, volume 861 of *Lecture Notes in Computer Science*, pages 142–153. Springer-Verlag, 1994.

[HW96] V. Haarslev and M. Wessel. GenEd – An Editor with Generic Semantics for Formal Reasoning about Visual Notations. In *1996 IEEE Symposium on Visual Languages* [VL96], pages 204–211.

[HW97] V. Haarslev and M. Wessel. Querying GIS with Animated Spatial Sketches. In *1997 IEEE Symposium on Visual Languages* [VL97], pages 197–204.

[JF93] R. N. Jackiw and W. F. Finzer. The Geometer's Sketchpad: Programming by Geometry. In A. Cypher, editor, *Watch What I Do: Progamming by Demonstration*. The MIT-Press, 1993.

[Jäh96] K. Jähnich. *Topologie*. Springer-Verlag, 5. Auflage, 1996.

[JV87] E. Jessen und R. Valk. *Rechensysteme – Grundlagen der Modellbildung*. Springer-Verlag, 1987.

[Kah90] K. M. Kahn. Technical Report SSL–90–38 [P90–00099]. Xerox Palo Alto Research Center, 1990.

[Kee89] S. E. Keene. *Object-Oriented Programming in Common Lisp*. Addison-Wesley, 1989.

[KFAF89] R. Kasturi, R. Fernandez, M. L. Amlani, and W. Feng. Map Data Processing in Geographic Information Systems. *Computer*, Dezember 1989.

[KM96] M. J. Kraak and M. Molenaar, editors. *Advances in GIS Research II, Proceedings on the Seventh International Symposium on Spatial Data Handling*. International Geographical Union, 1996.

[KRB91] G. Kiczales, J. des Rivièeres, and D. G. Bobrow. *The Art of the Metaobject Protocol*. The MIT Press, 1991.

[KS90] K. M. Kahn and V. A. Saraswat. Complete Visualizations of Concurrent Programs and their Executions. In *1990 IEEE Workshop on Visual Languages* [VL90], pages 7–14.

[Kuh96] W. Kuhn. Handling Data Spatially: Spatializing User Interfaces. In Kraak and Molenaar [KM96], pages 877–893.

[KVV97] Vorlesungskommentar des Fachbereichs Informatik – Wintersemester 1997/98. W. Mauke Söhne, Hamburg, 1997.

[LC95] Y. C. Lee and F. L. Chin. An Iconic Query Language for Topological Relationsships in GIS. *International Journal on Geographical Information Systems*, 9:25–46, 1995.

[Lev90] S. Levialdi. Cognition, Models & Metaphors. In *1990 IEEE Workshop on Visual Languages* [VL90], pages 69–79.

[LHM97] C. Lutz, V. Haarslev, and R. Möller. A Concept Language with Role-Forming Predicate Restrictions. Technical Report FBI–HH–M276/97, University of Hamburg – Computer Science Department, 1997.

[Lin95] R. K. Lindsay. *Images and Inference*, pages 111—1135. In Glasgow et al. [GNC95], 1995.

[LJ80] G. Lakoff and M. Johnson. *Metaphors We Live By*. University of Chigaco Press, 1980.

[LM97] Carsten Lutz and Ralf Möller. Definied Topological Relations in Description Logics. In *Proceedings of the International Workshop on Description Logics*, 1997.

[Lod83] K. L. Lodding. Iconic Interfacing. *IEEE Computer Graphics and Applications*, 3(2):11–20, 1983. Auch zu finden in [Gli90a, S. 231–238].

[LS87] P. C. Lockemann und J. W. Schmidt (Hrsg.). *Datenbankhandbuch*. Springer-Verlag, 1987.

[LS95] J. H. Larkin and H. A. Simon. *Why a Diagram is (Sometimes) Worth Ten Thousand Words*, pages 69—109. In Glasgow et al. [GNC95], 1995.

[Mar93] D. M. Mark. *Human Spatial Cognition*, pages 51–60. In Medyckyj-Scott and Hearnshaw [MSH93], 1993.

[Mar94] K. Marriott. Constraint Multiset Grammars. In *1994 IEEE Symposium on Visual Languages* [VL94], pages 118–125.

[McC83] G. F. McCleary, Jr. An Effective Graphic "Vocabulary". *IEEE Computer Graphics and Applications*, 3(2):46–53, 1983. Auch zu finden in [Gli90a, S. 239–246].

[Mey90] B. Meyer. *Introduction to the Theory of Programming Languages*. Prentice Hall International, 1990.

[Mey92] B. Meyer. Pictures Depicting Pictures – On the Specification of Visual Languages by Visual Grammars. In *1992 IEEE Workshop on Visual Languages* [VL92], pages 41–47.

[Mey94a] B. Meyer. Pictorial Deduction in Spatial Information Systems. In *1994 IEEE Symposium on Visual Languages* [VL94], pages 23–30.

[Mey94b] B. Meyer. Visuelle Logiksprachen zur Behandlung räumlicher Information. Dissertation, Fernuniversität Hagen, 1994.

[MHL97] R. Möller, V. Haarslev, and C. Lutz. Spatioterminological Reasoning Based on Geometric Inferences: The ALCRP(D) Approach. Technical Report FBI–HH–M277/97, University of Hamburg – Computer Science Department, 1997.

[Min97] M. Minas. Diagram Editing with Hypergraph Parser Support. In *1997 IEEE Symposium on Visual Languages* [VL97], pages 226–233.

[MK95] K. Myers and K. Konolige. *Reasoning with Analogical Representations*, pages 273—301. In Glasgow et al. [GNC95], 1995.

[Möl95] R. Möller. User Interface Management Systems: The CLIM Perspective. http://kogs-www.informatik.uni-hamburg.de/~moeller/home.html, 1995.

[Möl97] R. Möller. *Generierung von Visualisierungen in einem Rahmensystem zur systematischen Entwicklung von Benutzungsschnittstellen*. Infix, 1997.

[MSH93] D. Medyckyj-Scott and H. M. Hearnshaw, editors. *Human Factors in Geographical Information Systems*. Belhaven Press, 1993.

[MV95] M. Minas and G. Viehstaedt. DiaGen: A generator for diagram editors providing direct manipulation and execution of diagrams. In *1995 IEEE Symposium on Visual Languages* [VL95], pages 203–210.

[Nor92] P. Norvig. *Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kaufmann, 1992.

[NS93] A. Nerode and R. A. Shore. *Logic for Applications*. Springer-Verlag, 1993.

[Pag91] B. Page. *Diskrete Simulation: Eine Einführung mit Modula-2*. Springer-Verlag, 1991.

[Pal78] S. Palmer. Fundamental Aspects of Cognitive Representations. In E. Rosch and B. Lloyd, editors, *Cognition and Categorization*, pages 259–303. Lawrence Erlbaum Associates, Publishers, 1978.

[Pas95] B. Pasternak. Adaptierbares Kernsystem zur Interpretation von Zeichnungen. Motivation – Entwurf – Realisierung. Dissertation am Fachbereich Informatik der Universität Hamburg, 1995.

[PKC89] A. Pizano, A. Klinger, and A. Cardenas. Specification of Spatial Integrity Constraints in Pictorial Databases. *Computer*, 22(12):59–71, 1989.

[PN93] B. Pasternak and B. Neumann. Adaptable Drawing Interpretation using Object-Oriented and Constraint-Based Graphic Specification. In Tsukuba, editor, *Proceedings ICDAR*, pages 359–364, 1993.

[Pol97] K. Pollermann. Visualisierung von Geoinformationen in Decision Supporting Systems. Diplomarbeit am Fachbereich Informatik der Universität Hamburg, 1997.

[Pos95] J. Poswig. Visuelle Sprachen – Die Bedürfnisse der Computerbenutzer erfordern eine Novellierung der Softwaretechnologien, 1995. http://www-bior.sozwi.uni-kl.de/tum/tum_1_95/195poswi.html.

[Pos96] J. Poswig. *Visuelle Programmierung: Computerprogramme auf graphischem Weg erstellen.* Hanser-Verlag, 1996.

[Pot88] K. Potosnak. Do icons make user interfaces easier to use? *IEEE Software*, pages 97–99, Mai 1988. Auch zu finden in [Gli90a, S. 449–451].

[PS95] D. Papadias and T. Sellis. A Pictorial Query-by-Example Language. *Journal of Visual Languages and Computing*, 6:53–72, 1995.

[RCC92] D. A. Randell, Z. Cui, and A. G. Cohn. A Spatial Logic based on Regions and Connections. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning*, pages 165–176, 1992.

[Reh90] K. Rehkämper. Mentale Bilder – Analoge Repräsentationen. In Freksa und Habel [FH90], Seiten 47–67.

[Rek95] J. Rekers. Visuele Talen – Visual Languages. Skript zur gleichnamigen Vorlesung an der Leiden-Universität, 1995.

[RFS88] N. Roussopoulos, C. Faloutsos, and T. Sellis. An Efficient Pictorial Datbase System for PSQL. *IEEE Transactions on Software Engineering*, 14(5):639–650, 1988.

[RN95] S. Russel and P. Norvig. *Artificial Intelligence – A Modern Approach.* Prentice Hall, 1995.

[RS95] J. Rekers and A. Schürr. A graph grammar approach to graphical parsing. In *1995 IEEE Symposium on Visual Languages* [VL95], pages 195–202.

[RS96] J. Rekers and A. Schürr. A graph based framework for the implementation of visual environments. In *1996 IEEE Symposium on Visual Languages* [VL96], pages 148–155.

[Sch87] P. Schefe. *Informatik – Eine konstruktive Einführung.* BI-Wissenschaftsverlag, 2. Auflage, 1987.

[Sch91] P. Schefe. *Künstliche Intelligenz – Überblick und Grundlagen.* BI-Wissenschaftsverlag, 2. Auflage, 1991.

[Sch92] U. Schöning. *Logik für Informatiker.* BI-Wissenschaftsverlag, 3. Auflage, 1992.

[Sch96] M. Schneider. *Spatial Data Types for Database Systems – Finite Resolution Geometry for Geographic Information Systems*. Number 1288 in Lecture Notes in Computer Science. Springer-Verlag, 1996.

[SDK96] A. Schmitt, O. Deussen und M. Kreeb. *Einführung in graphisch-geometrische Algorithmen*. Teubner-Verlag, 1996.

[SE90] J. Star and J. Estes. *Geographic Information Systems – An Introduction* . Prentice Hall, 1990.

[Sed92] R. Sedgewick. *Algorithmen in C*. Addison-Wesley, 1992.

[Shn83] B. Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *Computer*, pages 57–69, August 1983. Auch zu finden in [Gli90a, S. 317–329].

[Shu86] N. C. Shu. *Visual Programming Languages: A Perspective and a Dimensional Analysis*, pages 11–34. In Chang et al. [CIL86], 1986. Auch zu finden in [Gli90b, S. 41–58].

[Üsk94] S. M. Üskürdarli. Generating Editors for Formally Specified Languages. In *1994 IEEE Symposium on Visual Languages* [VL94], pages 276–285.

[Sla98] S. Slade. *Object-oriented Common Lisp*. Prentice-Hall, 1998.

[Slo75] A. Sloman. Afterthoughts on Analogical Representations. In R. Schank and B. Nash-Webber, editors, *Theoretical Issues in Natural Language Processing*. Association for Computational Linguistics, 1975.

[Slo95] A. Sloman. *Musing on the Role of Logical and Nonlogical Representations in Intelligence*, pages 7—32. In Glasgow et al. [GNC95], 1995.

[Smi77] D. C. Smith. Principles of Iconic Programming. In *A Computer Program to Model and Stimulate Creative Thought*, pages 68–153. Birhäuser Verlag, 1977. Auch zu finden in [Gli90b, S. 216–251].

[Som95] I. Sommerville. *Software Engineering*. Addison-Wesley, 5. edition, 1995.

[Spi89] J. M. Spivey. *The Z Notation – A Reference Manual*. Prenctice Hall International, 1989.

[Ste90] Guy L. Steele, Jr. *Common Lisp – The Language, Second Edition*. Digital Press, 1990.

[Sut63] I. E. Sutherland. Sketchpad – A man-machine graphical communication system. In *AFIPS Conference Proceedings, Spring Joint Computer Conference*, pages 2–19. AFIPS, 1963. Auch zu finden in [Gli90b, S. 198–215].

[SV97] M. Scholl and A. Voisard, editors. *Advances in Spatial Databases – Fifth International Symposium, SSD '97*, volume 1262 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[VL89] *1989 IEEE Workshop on Visual Languages*. IEEE Computer Society Press, 1989.

[VL90] *1990 IEEE Workshop on Visual Languages*. IEEE Computer Society Press, 1990.

[VL91] *1991 IEEE Workshop on Visual Languages*. IEEE Computer Society Press, 1991.

[VL92] *1992 IEEE Workshop on Visual Languages*. IEEE Computer Society Press, 1992.

[VL94] *1994 IEEE Symposium on Visual Languages*. IEEE Computer Society Press, 1994.

[VL95] *1995 IEEE Symposium on Visual Languages*. IEEE Computer Society Press, 1995.

[VL96] *1996 IEEE Symposium on Visual Languages*. IEEE Computer Society Press, 1996.

[VL97] *1997 IEEE Symposium on Visual Languages*. IEEE Computer Society Press, 1997.

[Wes96] M. Wessel. Entwicklung eines konzeptorientierten generischen Grafikeditors in Common Lisp. Studienarbeit am Fachbereich Informatik der Universität Hamburg, 1996.

[WH93] P. H. Winston and B. K. P. Horn. *LISP*. Addison-Wesley, 3. edition, 1993.

[Win93] P. H. Winston. *Artificial Intelligence*. Addison-Wesley, 3. edition, 1993.

[WLZ95] D. Wang, J. Lee, and H. Zeevat. *Reasoning with Diagrammatic Representations*, pages 339—3393. In Glasgow et al. [GNC95], 1995.

[Woo93] M. Wood. *Interacting With Maps*, pages 111–123. In Medyckyj-Scott and Hearnshaw [MSH93], 1993.

[WS92] W. Woods and J. Schmolze. The KL-One Family. In F. Lehman, editor, *Semantic Networks in Artificial Intelligence*, pages 133–177. Pergamon Press, 1992.

[ZF95] K. Zimmerman and Ch. Freksa. Qualitative Spatial Reasoning Using Orientation, Distance, and Path Knowledge. *Journal of Applied Intelligence*, 1995.

[Zim94] K. Zimmermann. Measuring without Measures – The Δ-calculus. Universität Hamburg, Graduiertenkolleg Kognitionswissenschaft, Bericht Nr. 39, 1994.

Erklärung

Hiermit erkläre ich, die vorliegende Arbeit und die in ihr beschriebenen Programme selbständig erstellt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

27. Juni 2002

Unterschrift (Michael Wessel)