(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2004-537122
(P2004-537122A)
(43)公表日　平成16年12月9日(2004.12.9)

(51)Int.Cl.[7]	F I		テーマコード（参考）
G06F 9/44	G06F 9/06	620A	5B076
G06F 9/46	G06F 9/46	360B	5B098

審査請求 有　予備審査請求 有　（全 69 頁）

(21)出願番号	特願2003-515967(P2003-515967)
(86)(22)出願日	平成13年7月23日(2001.7.23)
(85)翻訳文提出日	平成16年1月22日(2004.1.22)
(86)国際出願番号	PCT/US2001/041389
(87)国際公開番号	WO2003/010659
(87)国際公開日	平成15年2月6日(2003.2.6)
(81)指定国	AP(GH,GM,KE,LS,MW,MZ,SD,SL,SZ,TZ,UG,ZW),EA(AM,AZ,BY,KG,KZ,MD,RU,TJ,TM),EP(AT,BE,CH,CY,DE,DK,ES,FI,FR,GB,GR,IE,IT,LU,MC,NL,PT,SE,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GW,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AT,AU,AZ,BA,BB,BG,BR,BY,BZ,CA,CH,CN,CO,CR,CU,CZ,DE,DK,DM,DZ,EC,EE,ES,FI,GB,GD,GE,GH,GM,HR,HU,ID,IL,IN,IS,JP,KE,KG,KP,KR,KZ,LC,LK,LR,LS,LT,LU,LV,MA,MD,MG,MK,MN,MW,MX,MZ,NO,NZ,PL,PT,RO,RU,SD,SE,SG,SI,SK,SL,TJ,TM,TR,TT,TZ,UA,UG,UZ,VN,YU,ZA,ZW

(71)出願人　504027071
ファブレス　ウイレイ
ＦＡＢＬＥＳ，　Ｗｙｌｅｉ
アメリカ合衆国　ハワイ州　９６７９０
マウイ，クラ，カウエヒ　プレイス　２１４

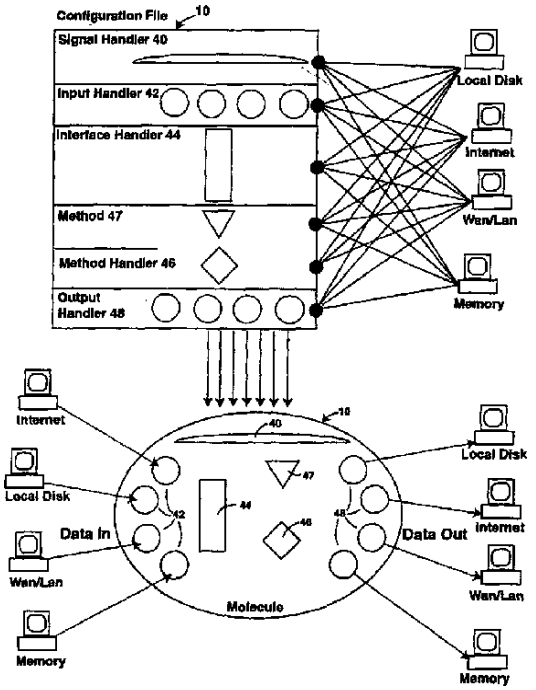
(71)出願人　504027082
パーク　ジョア
ＰＡＲＫ，　Ｊｏｒｅ
アメリカ合衆国　ハワイ州　９６７９０
マウイ，クラ，カウエヒ　プレイス　２１４

(74)代理人　100087583
弁理士　田中　増顕

最終頁に続く

(54)【発明の名称】自律的なデータハンドラを用いるコンピュータ処理兼プログラミング方法

(57)【要約】
【構成】コンピュータ処理およびプログラミング方法はデータを自律的に処理するようにダイナミックに構成された複数のソフトウエアエンティティ（“分子”）を形成することを要求する。分子は分子の状態、すなわち、処理タスクが充足されたか、処理タスクの結果、分子の動作が終了されたか、中断されたか、再構成されたか、１つまたはそれ以上の“次の”分子の形成によって継続されるか、を指示する信号を送受信できる。分子はソフトウエアのマイクロコンポーネント（４０、４２、４４、４６、４８）の共通の組み合わせから形成され、マイクロコンポーネントは任意のオペレーティングシステムで実行する任意のプログラミング言語でプログラムされる。分子は単一のコンピューティングリソースとともに存在してもよいが、理想的には、ネットワーク上または並列処理環境内の異なったタイプの分散型コンピューティングリソースで展開するのに適したものである。データ処理タスクの全体はタスクを果たすようにダイナミックに適応される“ロジックウエブ”内に分子を形成することによって実行される。ロジックウエブは、異なった



【特許請求の範囲】
【請求項１】
各々が信号ハンドラ（４０）、少なくとも１つの入力ハンドラ（４２）、少なくとも１つの出力ハンドラ（４８）、インターフェイスハンドラ（４４）および少なくとも１つの手法ハンドラ（４６）を含むソフトウエアのマイクロコンポーネントで構成される複数のソフトウエアエンティティ（“分子１０”）を形成し、ここで、前記入力ハンドラ、出力ハンドラおよび信号ハンドラはそれぞれの分子の外部から他の分子またはロジックウエブと通信信号を送受信するために動作でき、かつ他のマイクロコンポーネントと接続されており、前記少なくとも１つの入力ハンドラは入力データを持ち行列に入れるために動作でき、前記インターネットハンドラは前記入力ハンドラによって受け取られる所要の入力データの所定の入力条件がいつ充足されるかを決定し、次に前記手法ハンドラを呼び出し、前記手法ハンドラは入力データを処理するために前記関連する手法を呼び出すように動作でき、前記少なくとも１つの出力ハンドラは前記手法による入力の処理を出力するために動作でき、
その後の実時間の使用のために形成した分子をライブラリ内に格納し、
所定のコンピューティングリソース上に、ライブラリから取り出されて所定のコンピューティングリソース上で実行される第１の分子を呼び出すことによって所定の処理タスクを実行するように選択される複数の分子からなるロジックウエブを配置する、
ステップからなり、
前記第１の分子は、前記他の分子を呼び出す通信信号を送る前記第１の分子の前記信号信号ハンドラによって“進行中”に前記ロジックウエブを徐々に拡張するために１つまたはそれ以上の他の分子を呼び出す、
ことを特徴とするコンピュータ処理方法。
【請求項２】
請求項１記載のコンピュータ処理方法において、前記形成ステップは複数のロジックウエブを形成することを含み、ロジックウエブの各々はコンピューティングリソースでデータ処理タスクを自律的に実行するように構成されたソフトウエアエンティティのウエブを有し、前記配置ステップは複数のコンピューティングリソースのうちの１つで複数のロジックウエブを配置することを含むことを特徴とするコンピュータ処理方法。
【請求項３】
請求項２記載のコンピュータ処理方法において、前記複数のコンピューティングリソースが分散型コンピューティング環境内のネットワーク上に分散されていることを特徴とするコンピュータ処理方法。
【請求項４】
請求項２記載のコンピュータ処理方法において、前記複数のコンピューティングリソースが並列処理環境内で並列に配列した中央処理ユニット（ＣＰＵ）を含むことを特徴とするコンピュータ処理方法。
【請求項５】
請求項１記載のコンピュータ処理方法において、少なくとも１つのロジックウエブは複数の分子空なり、前記ロジックウエブは分子の内の初期ホストを形成する初期化ファイルとして存在し、初期ホストは増分処理ステップの連続層で他の分子を呼び出してかつ形成することを特徴とするコンピュータ処理方法。
【請求項６】
請求項１記載のコンピュータ処理方法において、前記形成ステップは増分処理ステップの連続層で次の分子を形成するハンドラ機能を有する分子を形成することを特徴とするコンピュータ処理方法。
【請求項７】
請求項１記載のコンピュータ処理方法において、前記形成ステップは分子が終了されるときその機能の消去を実行するための組み込まれたハンドラ機能を有する分子を形成することを特徴とするコンピュータ処理方法。

【請求項８】
請求項１記載のコンピュータ処理方法において、前記形成ステップはそのマイクロコンポーネントハンドラの状態に関する情報を記録しかつ前記信号ハンドラを通して前記状態情報を外部に送るタイプのハンドラの分子を形成することを特徴とするコンピュータ処理方法。
【請求項９】
請求項１記載のコンピュータ処理方法において、前記信号ハンドラは信号を受け取ることができ、かつ処理タスクを実行している間、分子のマイクロコンポーネントハンドラをダイナミックに再構成するタイプのハンドラを有することを特徴とするコンピュータ処理方法。
【請求項１０】
請求項１記載のコンピュータ処理方法において、前記インターフェイスハンドラは、前記手法ハンドラを関連する手法のために呼び出す前に所定の入力条件が存在することを前記入力ハンドラが示すまで待機することによって、自律的に待機し、検索し、入力データを処理するための前記関連する手法とともに進める特性を分子に与えるタイプのハンドラを含むことを特徴とするコンピュータ処理方法。
【請求項１１】
請求項１記載のコンピュータ処理方法において、前記インターフェイスハンドラは、いつそれぞれの所要のデータの存在に対するそれぞれの所定の入力条件が充足されるかを決定するための、かつ複数の手法ハンドラと関連する手法の内のそれぞれ１つを呼び出すためのタイプの複数のハンドラを含むことを特徴とするコンピュータ処理方法。
【請求項１２】
請求項１記載のコンピュータ処理方法において、前記入力ハンドラは複数の異なってデータソースの形式にそれぞれ対応するタイプの複数の入力ハンドラの内の１つから選ばれることを特徴とするコンピュータ処理方法。
【請求項１３】
各々がそれぞれの分子の外部の分子と通信信号を送受信するためのソフトウエアマイクロコンポーネントで構成された複数のソフトエンティティ（“分子１０”）を形成し、ここで、各分子の前記マイクロコンポーネントは前記分子が配置された所定のコンピューティング環境で入力を処理するために互いに接続されており、かつ入力データを処理する出力結果を与えるものであり、
複数のコンピューティング環境の内のそれぞれの１つに複数の分子の各々を配置し、
増分処理ステップの連続層でデータ処理機能の分子の“ロジックウエブ”を初期化するためにそれぞれのコンピューティング環境内に配置された各分子を初期化するステップを有し、
第１の分子が“進行中”に前記ロジックウエブを増分的に拡張するために１つまたはそれ以上の他の分子を呼び出す、
ことを特徴とする分散型コンピューティング方法。
【請求項１４】
請求項１３記載の分散型コンピューティング方法において、各コンピューティング環境内の各ロジックウエブはそれぞれのコンピューティング環境内でそのデータ処理機能を自律的に実行し、かつそのコンピューティング環境から得ることが望まれた出力を戻すことを特徴とする分散型コンピューティング方法。
【請求項１５】
請求項１４記載の分散型コンピューティング方法において、各ロジックウエブはそれぞれのコンピューティング環境に対する出力を外部の監視エンティティに戻し、前記外部監視エンティティは他のコンピューティング環境からの出力を組み合わせて分散コンピューティングの組み合わせた出力を得ることを特徴とする分散型コンピューティング方法。
【請求項１６】
請求項１５記載の分散型コンピューティング方法において、コンピューティング環境はネ

ットワーク上に分散した複数のコンピューティングサイトであり、ロジックウエブはネットワーク上に信号を送ることによって出力を戻すことを特徴とする分散型コンピューティング方法。

【請求項１７】

請求項１５記載の分散型コンピューティング方法において、コンピューティング環境は並列処理環境内で並列に動作される処理ユニット（ＣＰＵ）の配列内の複数のコンピューティングリソースであることを特徴とする分散型コンピューティング方法。

【請求項１８】

各々がそれぞれの分子の外部の分子と通信信号を送受信するためのソフトウエアマイクロコンポーネントで構成された複数のソフトエンティティ（“分子１０”）を形成し、ここで、各分子の前記マイクロコンポーネントは前記分子が配置された所定のコンピューティング環境で入力を処理するために互いに接続されており、かつ入力データを処理する出力結果を与えるものであり、

ネットワーク上に分散したコンピューティングサイトである複数のコンピューティング環境の内のそれぞれの１つに複数の分子の各々を配置し、

増分処理ステップの連続層でデータ処理機能の分子の“ロジックウエブ”を初期化するためにそれぞれのコンピューティング環境内に配置された各分子を初期化し、ここで、第１の分子が“進行中”に前記ロジックウエブを増分的に拡張するために１つまたはそれ以上の他の分子を呼び出し、

それぞれのコンピューティング環境内でデータ処理機能を実行しかつネットワークコンピューティングサイトから得られることが望まれる出力を戻す各ロジックウエブを各ネットワークコンピューティングサイトに持つ、

ことを特徴とするネットワークコンピューティング方法。

【請求項１９】

請求項１８記載のネットワークコンピューティング方法において、各ロジックウエブはそれぞれのネットワークコンピューティングサイトの出力をネットワーク監視エンティティに戻し、前記ネットワークエンティティがネットワークコンピューティングサイトからの出力を組み合わせてネットワークのための組み合わせ出力を得ることを特徴とするネットワークコンピューティング方法。

【請求項２０】

請求項１９記載のネットワークコンピューティング方法において、ネットワークはネットワーク内のネットワーク（“インターネット”）であり、ロジックウエブがインターネット上のウエブサイトで展開され、ウエブサイトからデータを自律的にコンピュータ処理してネットワーク監視エンティティにその出力を戻すことを特徴とするネットワークコンピューティング方法。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は一般にコンピュータ処理およびプログラミング方法に関し、特に、新規な形式の自律的データハンドラを用いる方法に関する。

【背景技術】

【０００２】

最も初期の理論的基礎から現在まで機械指向のパラダイムとしてコンピュータ処理が開発されてきており、このコンピュータ処理では、機械命令が列データを変換してシーケンシャルステップに入力して有効な出力を得るように中央処理装置（ＣＰＵ）によって実行される。ますます多量のデータを含むますます複雑なタスクを達成するために、コンピュータハードウエアの開発が速度、能力、容量を指数関数的に増加させながら継続されてきて、今や原子の寸法や光の速度および波長の物理的限界に近づいている。線形プログラミング手法によって複雑なタスクをプログラミングすることは集中的なコードの書き込みをますます必要とし、コードの書き込みは何百万のコード列を要し、莫大なデバッグの努力を

必要とするエラーやクロスコードの衝突を受け易い。また、プログラミングは選択したオペレーティングシステムに対して１つの選ばれたプログラミング言語で行われることが本質的に必要であり、結局、陳腐化した古いシステムとなることになる。

【０００３】

オブジェクト指向プログラミング（ＯＯＰ）手法が多数のプログラミングタスクを内蔵式で自己実行型であるモジュールに分散させるために開発されてきた。特定の機能に対して最適化されたモジュールは分類でき、多様の多数のプログラミングタスク内でモジュラー形式で再使用できる。しかしながら、ＯＯＰ手法は全体としてプログラミング環境でクラス定義階層の実行を必要とし、他の環境を支援するように再コンパイルされねばならない閉鎖系やＯＯＰモジュールのライブラリを生じさせる結果となる。

【０００４】

処理タスクをタンデムに配列したＣＰＵによって実行される部分に分散することによって能力を増大させようとするために、並列処理手法が過去何十年にもわたって開発されてきた。しかしながら、並列処理に対する従来のアプローチはプログラムされたタスクの集中的管理と御制を必要とし、集中的管理と制御はタスクの調整と同期に厳しい必要条件を課すものである。集中的であり、特に並列動作中に誤動作を受けやすく、さらに選択したプログラミング言語とオペレーティングシステムに拘束される線形プログラミング手法をデータの並列処理手法はやはり用い続けているものである。

【０００５】

近年、大規模な分散型ネットワークの拡大で、極めて大規模なタスクまたは複雑なタスクを小さいピース（要素）に分けネットワーク上で協働するコンピュータに分散することによってタスクを実行することが提案されてきた。なお、これは「分散コンピューティング」または「ネットワークコンピューティング」と呼ばれている。分散コンピューティングの提案により、計算を高度に集中してタスクを処理することから、データがネットワーク上のリソース（資源）に広範囲に分散されて分散努力を通して計算を高度に集中させないＣＰＵで局所的に処理するものに、コンピューティングパラダイムが移行することを表している。しかしながら、同じ線形プログラミング手法と集中的管理および制御が用いられてきたので、データに対する分散コンピューティングパラダイムは有効な結果を達成するためにはやはり集中的プログラミングと作業の協働を必要としている。１つまたはそれ以上の個々の処理演算の失敗や通信または集中型処理管理との協働に伴う時間の遅れは、このアプローチの有効性を台無しにするボトルネックを生じさせるかもしれない。

【０００６】

多重非同期式コンピュータ処理を利用する１つの手法は、「フローに基づくプログラミング（Ｆｌｏｗ　Ｂａｓｅｄ　Ｐｒｏｇｒａｍｍｉｎｇ）（“ＦＢＰ”）」と呼ばれ、それは著作名「フローに基づくプログラミング：アプリケーション開発への新しいアプローチ」　著者　ポール　モリソン、出版社　ホン　ノストランド　レインホルド（ニューヨーク州）、１９９４年出版　に一層詳細に説明されている。この文献は、３つの原理「非同期処理、それ自体の寿命を持ったデータパケット、および接続の外部定義」に基づくものとしてフローに基づくプログラミングを記載している。ＦＢＰでは、アプリケーションは所定の条件を横切ってデータを交換する“ブラックボックス”処理のネットワークとして定義される。ＦＢＰは、“ブラックボックス”処理のネットワークをとおしてタグ付きのデータパケットをステップ処理するプロセスであり、したがって、データ中心処理である。ＦＢＰによるアプローチは極めて予測的な能力が十分働くことを必要とする。

【非特許文献１】

著作名「フローに基づくプログラミング：アプリケーション開発への新しいアプローチ」　著者　ポール　モリソン、出版社　ホン　ノストランド　レインホルド（ニューヨーク州）、１９９４年出版

【発明の開示】

【発明が解決しようとする課題】

【０００７】

したがって、本発明の第１の目的はコンピュータ処理タスクの管理を分散し、最も効率的な経路を利用するような順応性のある選択と共に自律的にデータの分散処理、すなわち並列処理を進めるコンピュータ処理と関連するプログラミング方法を提供することにある。

【０００８】

本発明の他の目的は、拘束されたプログラミング環境の要件を回避することによって並列データ処理すなわち分散データ処理を容易にするように最適に設計される新規なプログラミング方法を提供するものであり、それにより多重プログラミング言語の同時の使用が支援され、集中処理管理がなくされるものである。また、複雑な処理タスクに必要とされるプログラミングの量や固有の性質を減少させ、誤りに至るようなプログラミングの脆さを減少させることである。

【課題を解決するための手段】

【０００９】

本発明によると、コンピュータ処理および関連するプログラミング方法（手法）は、データを自律的に処理するようにダイナミックに構成された複数のソフトウエアエンティティ（本明細書では“分子”と呼ぶ）を形成することを要求する。分子は、分子の状態、処理タスクが実行されたか否か、処理タスクの結果、分子の作業が終了されたか、中断されたか、再構成されたか、１つまたはそれ以上の“次の”分子の形成によって継続されているか否かを指示する信号を送受信する。分子はソフトウエアのマイクロコンポーネントの共通の組み合わせから形成され、マイクロコンポーネントは任意のオペレーティングシステム環境に実行する任意のプログラミング言語でプログラミングされてもよい。

【００１０】

処理タスクの全体に対して形成された分子は単一のコンピューティングリソース（ＣＰＵおよびデータベース）だけと共存してもよいが、理想的には、ネットワーク上の異なった形式の分散コンピューティングリソースと共に、または並列処理環境内に配置されるのが適している。各分子はプログラミング言語、すなわち、操作しようとするコンピューティングリソースに適した言語でプログラムされてもよい。分子は、コンパイルされた、またはランタイム（実行時）コンパイルされたコードと共に機能でき、コンピューティングリソース上で実行するのに適した処理の一部だけと共にコンピューティングリソース上で実行可能である。分子は互いに伝達でき、共通のネットワークプロトコルを用いてデータを送受信できる。分子は、データ処理タスク全体が完了するまで、それぞれのデータ処理タスクを自律的に完了するように作業する。データ処理タスクの全体は、分子を形成し、かつタスクを達成するようにダイナミックに順応できる“ロジックウエブ”内で作業と相互作用を構成することによって実行される。

【００１１】

並列処理環境に順応することによって、分子は並列コンピューティングリソース上で自律的に実行するようにダイナミンクに構成できる。ロジックウエブは、各分子が処理しようとするデータオブジェクトおよび中間ＣＰＵリソースとしての並列プロセッサ配列内の任意の利用可能なプロセッサに効率的に接近できるように定義される。プロセッサ接近のスケジューリングは、メインのコントローラプログラムによって、または別個のスケジューラ分子またはミニロジックウエブによって取り扱うことができる。分子は、それらの状態、結果、および終了や次の分子の再構成すなわち形成を伝達するようにプログラムされる。分子の分散した自律的な作業によって、データ処理タスクを一層効率的に迅速に達成でき、従来の並列処理手法における融通の効かないスケジューリング構造と時間遅延の課題を回避できる。

【００１２】

ソフトウエア構成のエンティティとして、分子は以下のソフトウエアマイクロコンポーネントからなる。すなわち、信号ハンドラ、入力ハンドラ、インターフェイスハンドラ、手法ハンドラ、関連した手法（手法要素）、および出力ハンドラからなる。信号ハンドラは、他の分子を呼び出し、ネットワーク上の他の分子およびリソースに伝達するための信号を送受信できる。入力ハンドラはデータ入力を受信し、データをキュー（待ち行列）に入

力する。インターフェイスハンドラは、いつ分子への入力が手法ハンドラへの経路を開くための所定の条件を満たすかを決定する。手法ハンドラは所定の手法（手法要素）に従って入力ハンドラのキューから送られたデータを処理し、出力を出力ハンドラに送る。

【００１３】

ソフトウエアプログラミング手法として採用された、分子のロジックウエブは任意の形式の任意の利用できるコンピューティングリソース上で実行でき、かつ任意の位置で任意のファーマットでデータを処理できるように形成される。初期化ファイルは意図したコンピューティングリソース上で分子の初期ホストを形成するためにプログラムされる。初期分子はプログラムステップのプログレション（工程）を表す連続的な層、すなわちウエーブ内で次の分子を発生するように作業する。各初期化ファイルは標準のマイクロコンポーネント形式のライブラリから分子のソフトウエアのマイクロコンポーネントを組立てるように構成される。これらの標準形式はそれらの機能の項目で目的に合うように選ばれ、ＯＯＰヒエラルキー（階層）内の融通の効かないクラス形式ではない。新しい形式のソフトウエアのマイクロコンポーネントは必要に応じて容易に形成でき、作業を意図している環境に適したプログラミング言語でプログラムされる。

【００１４】

本発明のコンピュータ処理およびプログラミング方法で、複雑な計算のタスクおよびデータ処理タスクがネットワークを含む任意の位置で任意の形式の任意の利用できるコンピューティングリソースを利用するコンポーネント分子から構成されるロジックウエブを設計することによって実行される。かくして、ロジックウエブは、異なった従来のシステムで、異なった言語でプログラムされたアプリケーションで、任意のファーマットに格納された任意の形式のデータとともに作業するように分子と共に設計できる。その結果、分散すなわち並列処理環境でのデータ処理タスクは一層効率的に実行され、完全に新しい形式のデータ処理タスクが行われる。

【発明の効果】

【００１５】

本発明では、はコンピュータ処理タスクの管理を分散し、最も効率的な経路を利用するような順応性のある選択と共に自律的にデータの分散処理、すなわち並列処理を進めるコンピュータ処理と関連するプログラミング方法が得られる。

【００１６】

また、本発明では、拘束されたプログラミング環境の要件を回避することによって並列データ処理すなわち分散データ処理を容易にするように最適に設計される新規なプログラミング方法を提供するものであり、それにより多重プログラミング言語の同時の使用が支援され、集中処理管理がなくされるものである。また、複雑な処理タスクに必要とされるプログラミングの量や固有の性質を減少させ、誤りに至るようなプログラミングの脆さを減少させることである。

【発明を実施するための最良の形態】

【００１７】

本発明は、各々が信号ハンドラ、少なくとも１つの入力ハンドラ、少なくとも１つの出力ハンドラ、インターフェイスハンドラおよび少なくとも１つの手法ハンドラを含むソフトウエアのマイクロコンポーネントで構成される複数のソフトウエアエンティティを形成し、ここで、前記入力ハンドラ、出力ハンドラおよび信号ハンドラはそれぞれの分子の外部から他の分子またはロジックウエブと通信信号を送受信するために動作でき、かつ他のマイクロコンポーネントと接続されており、前記少なくとも１つの入力ハンドラは入力データを持ち行列に入れるために動作でき、前記インターネットハンドラは前記入力ハンドラによって受け取られる所要の入力データの所定の入力条件がいつ充足されるかを決定し、次に前記手法ハンドラを呼び出し、前記手法ハンドラは入力データを処理するために前記関連する手法を呼び出すように動作でき、前記少なくとも１つの出力ハンドラは前記手法による入力の処理を出力するために動作でき、　その後の実時間の使用のために形成した分子をライブラリ内に格納し、所定のコンピューティングリソース上に、ライブラリから

取り出されて所定のコンピューティングリソース上で実行される第１の分子を呼び出すことによって所定の処理タスクを実行するように選択される複数の分子からなるロジックウエブを配置する、ステップからなり、前記第１の分子は、前記他の分子を呼び出す通信信号を送る前記第１の分子の前記信号信号ハンドラによって“進行中”に前記ロジックウエブを徐々に拡張するために１つまたはそれ以上の他の分子を呼び出す、ことを特徴とするコンピュータ処理方法である。

【実施例】

【００１８】

本発明では、新規なコンピュータ処理およびプログラミング方法は、他の分子および利用できるコンピューティングリソース（資源）と通信中に自律的にデータを処理するようにプログラムされた複数のソフトウエアエンティティ（“モレキュール：分子”）を形成すること要求するものである。分子は任意のオペレーティングシステムで実行する任意のプログラミング言語でプログラムされた共通の組み合わせのソフトウエアのマイクロコンポーネントから形成される。分子はロジックウエブの分子で構成されてデータ処理タスクを実行し、データ処理タスクは全体の処理タスクの１つのコンポーネントであってもよい。１つのロジックウエブまたはロジックウエブ内の１つのロジックウエブは単一の処理環境に内にあってもよいが、並列処理および分散処理環境内で用いられるのに特に適している。

【００１９】

図１を参照すると、本発明のソフトウエアエンティティ（“分子”）１０の全体が示されている。各ソフトウエア分子１０はその構成（データ）ファイル、すなわち、信号ハンドラ４０、入力ハンドラ４２、インターフェイスハンドラ４４、手法４７、手法ハンドラ４６および出力ハンドラ４８から基本的に構成されている（一層詳細には以下の“ソフトウエア分子”で述べる）。分子コンポーネントはローカルまたは遠隔のリソースから集められて分子を形成するために組み合わされる（一層詳細には以下の“プログラミング手法”で述べる）。各分子は定義したデータオブジェクトを自律的に処理するようにプログラムされ、他の分子に信号を送信しまた他の分子から信号を受信し、分子は他の分子とロジックウエブ内で相互作用して処理タスクを達成する。ロジックウエブは任意の利用できるコンピューティングリソース上で実行するように割り当てられ、コンピューティング環境内で通信バスを介して、またはネットワーク環境内のネットワークによって（図では接続線で示す）他の分子と連絡する。通信リンクを通して、分子はその信号ハンドラ４０と信号を送受信でき、すなわち、アクセスデータは任意の入力リソースを通して入力ハンドラ４２に入力され、出力データは出力ハンドラ４８を通して出力される。

【００２０】

図２を参照すると、分子のロジックウエブ２０の配置が分散コンピューティング環境に対して示されている。ローカルコンピュータ、ＬＡＮ／ＷＡＮサーバ、ＣＰＵ、ネットワークサーバ、データベース、Ｉ／Ｏ装置等の形式の多数の分散コンピューティングリソースが任意の形式のネットワークで接続されている（接続線で示す）。ネットワークはローカルエリアネットワーク（ＬＡＮ）、ワイドエリアネットワーク（ＷＡＮ）またはネットワークのグローバルネットワーク（インターネット）であってもよい。ロジックウエブ内のウエブ全体が単一のコンピューティングリソースに存在し、ネットワークを通して、分子によって処理されるべきデータを持つ他のコンピューティングリソースと通信してもよいが、もし特定のコンピューティングリソースにおいてデータを処理するためのロジックウエブが存在し、これらのコンピューティングリソースにおいて局所的にデータを処理し、それらの状態情報と結果を他の分子にネットワークを介して通信するならば、システムの効率は飛躍的に増大する。例えば、ＣＯＢＯＬフォーマット済みのデータを処理するための分子がＣＯＢＯＬフォーマット済みのデータが格納されるＣＯＢＯＬサーバに存在してもよく、またはＪＡＶＡ内でデータを処理するための分子が所望のデータと局所的に存在するＪＡＶＡ仮想記憶装置を持つコンピューティングリソースに存在してもよい。この分散モードでは、分子の局所的に存在するロジックウエブが、その環境で容易に操作できる

ように、プログラミング言語が存在するコンピューティングリソースにおいて用いられるプログラミング言語でプログラムされてもよい。

【００２１】

１つのロジックウエブまたはロジックウエブ内のロジックウエブはＣＰＵリソースの配列（ＣＰＵ＃１、・・・ＣＰＵ＃６で示す）および関連するデータベースとＩ／Ｏ装置を持つ並列処理環境内に同様に配置されてもよい。ロジックウエブは、処理を望むデータオブジェクトに各分子が効率的にアクセス（接近）できるように構成されている。ＣＰＵコンピューティングリソースへの分子のアクセスはリソース管理分子またはロジックウエブによって立案されてもよい。他の手法としては、分子は並列処理環境で従来どおり用いられているようなＣＰＵスケジューラーと連絡してもよい。分子がデータを自律的に処理するように設計されているので、分子は集中的処理管理によって監視されて制御されることなく作業を行う。分子は、状態情報、結果、終了、次の分子の再構成、すなわち、生成をシステム内に設定された他の分子またはロジックウエブに伝達する。しかしながら、ロジックウエブをハイブリッド配置内の従来のコントローラプログラムと連絡させることも可能である。

ソフトウエア分子

図３を参照すると、信号ハンドラ４０、入力ハンドラ４２、データインターフェイスハンドラ４４、手法ハンドラ４６および出力ハンドラ４８を持つ基本形式のソフトウエア分子用の好ましい構成が示されている。信号ハンドラ４０は、ＣＢ（市民バンド）または個人ラジオネットワーク上の信号の分散一斉通知と同様に、通信バスまたはネットワーク上で他の分子または他のリソースと信号を送受信できる。信号は、通信バスまたはネットワークで用いられている共通のプロトコル、例えば、インターネット通信で用いられているＴＣＰ／ＩＰプロトコルで一斉通知される。

【００２２】

信号ハンドラ４０はシステム内で以下の基本ルーチンを実行する。

１．データソースからのコンフィギュレーションファイル（構成ファイル）の読み取り　構成ファイルは分子のサブ構成の名前、属性および形式を含む。

２．構成ファイル内で定義されたようにサブ構成を組み立てることによる分子の形成　サブ構成のロードや呼出は、格納されたハンドラの形式に対するファイルのライブラリを持つローカルのリソースまたは遠隔のリソースから構成用の名前と形式を呼ぶことによって進める。構成ファイルは信号ハンドラに通されて分子に追加される。

３．信号環境の選択　信号ハンドラは構成ファイル内で特定された信号環境を聞き取る。

４．分子の消去、停止、再開

５．特定した信号環境との信号の送受信

６．信号を受けたとき信号を解釈して再構成できる能力を持つ分子の適切なサブ構成に信号を伝達すること　信号ハンドラは要請をサブ構成に送ることができる。信号ハンドラの要請の例としては、（ａ）入力ハンドラまたは出力ハンドラへキュー寸法を減少させる要請、（ｂ）５つのサブ構成内の任意のものへ状態を書き込み、その後停止させる要請、（ｃ）５つのサブ構成内の任意のもののリマッピングを指示する要請、（ｄ）バッファの寸法またはメモリの使用の変更

７．“次の”または他の分子をロードするための呼出し　信号ハンドラは、“次の”分子が既に存在するか否かを見るために特定の名前のレジストリ（登録事項）を監視している。もし存在するならば、信号ハンドラは分子をリマップさせる。または、もし特定されているならば、“次の”分子を呼び出さない。もし存在しないならば、“次の”分子をロードする新しい信号ハンドラの形成を要求する。新しい信号ハンドラは子供のスレッドでなく、むしろ完全に別個で特殊な新しいエンティティである。これは、スレッドの全てが初期の親のスレッドに至り、一緒になって１つのアプリケーションを形成する他のプログラミングパラダイムとは異なっている。

【００２３】

信号ハンドラの信号の送受信は、通信に適した、または分子が存在するコンピューティン

グリソースの“信号”環境に適した任意の手法で行われる。従来のコンピュータまたはネットワーク環境では、信号ハンドラはシステム内またはネットワーク上の特定した分子にアドレス指定したデータパケットとして“信号”を送信し、受信することができる。信号ハンドラからの信号は名前レジストリに維持されたアドレスに基づいてターゲット分子に送られる。名前レジストリは、他の分子またはロジックウエブ、または従来のリソースレジスタによって維持されてもよい。例えば、ＪＡＶＡ（商標） Ｓｈａｒｅｄ Ｄａｔａ環境のような従来のデータ共有環境は、サーバ、クライアントおよび所定のアプリケーションに対して設定されたデータチャンネルに対するレジスタを有する。信号ハンドラは、すべての分子に対するグローバルな（地球規模の）一斉通知用の信号環境に対しても“対応”でき、または構成ファイルに載った特定のチャンネルに合わせてもよく、または任意の特定の信号環境からの任意の信号を待ってもよく、またはロジックウエブ内の他の分子のグループと共に同一のチャンネルに合わせられてもよい。

【００２４】

分子のハンドラコンポーネント（構成要素）は分子を形成する、すなわち分子をダイナミックに構成するために呼び込まれることができる形式の共通の組み合わせとして構成される。分子は、格納したメモリからコードの対応するセクション（部分）を取り出すために、所望のハンドラの形式の名前を信号で送ることによって組み立てられる。このようにして、任意の数の形式の入力および出力ハンドラがデータソースまたはメモリモデルの正しい形式に従ってアクセスできる。同様に、分子は多数の異なった形式の手法ハンドラで構成できる。インターフェイスハンドラに新しい構成情報が供給されると、所定の条件を呼び込むための手法ハンドラの形式に関する新たな情報が格納され、その結果、分子は、これらの条件が存在するとき新しい手法を実行する。分子の５つのコンポーネントハンドラの任意のものに対するソースまたはオブジェクトコードの適切なセクションがローカルメモリ、ＬＡＮ、ＷＡＮ、またはインターネットを経由して、格納されたファイルからアクセスできる。

【００２５】

分子構成ファイル内で参照されるようなハンドラの形式と属性の範囲内の基本的な例がＡｐｐｅｎｄｉｘ（付録１）（表１～表４）に設けられたリストに示されている。しかしながら、形成されるハンドラの数、形式および属性は無制限であり、異なった形式のメモリモデル、データ共有環境およびコンピューティングリソースに適合するようにプログラマの要求に比例して拡張できる。

【００２６】

入力ハンドラ４２は、プログラマによって決定されるが、特定の形式のメモリモデルからデータソースへの分子の接続を確立する。設計によって、入力ハンドラは任意のメモリモデルのデータソースに対するインターフェイスを可能にする。入力ハンドラは、選んだ入力ハンドラの形式によって決められるが、キューが可能とする程度まで到来データソースを待ち行列に入れ、次に、データを持ち行列に入れたインターフェイスハンドラ４４に登録する。入力ハンドラは、また、分子の他のサブコンポーネントがそのデータを待ち行列にいれるのを可能にする。図示の例では、メモリに対する到来ポインタ、すなわち、データ“Ａ”が入力ハンドラ４２ａおよび入力ハンドラ４２ｂのキュー“Ｂ”に受け入れられ、到来ポインタ／データ“Ｄ”が入力ハンドラ４２ｃのキュー“Ｅ”に受け入れられる。

【００２７】

任意の入力ハンドラに入ってくるデータソースの形式は分子内に組み立てられた入力ハンドラの形式によって決まる。入力ハンドラの各形式はデータをいかに受け入れるかに対して所定のメモリモデルのある特定のデータチャンネルを特定できる。例えば、カリフォルニア州マウンテンビュー所在のサンマイクロシステムズ社のＪａｖａ Ｓｈａｒｅｄ Ｄａｔａ Ｔｏｏｌｋｉｔ（ＪＳＤＴ：商標）のような従来のデータ共有環境のメモリモデルおよびデータ取り扱い機能が用いられてもよい。入力ハンドラが形成されるとき、特定のデータチャンネルがシステムのメモリ環境に対して登録される。特定のデータチャンネル上でソース（他の分子またはシステム内の第３者のクライアント）から入ってくるデー

タはその入力ハンドラに送られる。特定のデータ形式に対して入力ハンドラを選択することによって、ランタイム（実行時）におけるデータ形式のチェックを行う必要性が回避される。種々の形式の入力ハンドラが取り扱う種々のファイル形式に適合するように書かれる。好ましい構成では、一旦入力ハンドラがデータソースを受け入れてデータを処理し始めると、出力ハンドラが所定の環境に対するインターフェイスに合うようにデータを書式設定する、すなわち形式化するまで、データは“無形式”である。多数の入力ハンドラを用いる利点は、同一の分子に集められて処理される多数のメモリモデルからのデータソースへ同時にアクセス可能であることである。

【００２８】

インターフェイスハンドラ４４は、いつ分子への入力が１つまたはそれ以上の条件を満たすかを決定することによって分子に対する中心機能を実行する。命令によって指示されて集中処理管理によって監視されるのではなく、分子が所定の条件を満たす入力を待つことができることによって、分子は待機し、監視し、継続する能力を持ち、その結果、分子は自律的に作動する特性を持つ。これは、特定のシーケンスで特定の時間に特定のタスクを実行するように書かれた従来の線形プログラミングのコード命令から区別される。所定の条件は入力ハンドラへの所要のデータ入力の全てが存在してもよい。その他、所定の条件は信号ハンドラによって受け取った信号が存在してもよい。インターフェイスハンドラは、手法を呼び出す所定の条件、条件の所定の組み合わせのための手法、または結果を取り扱うように呼び込まれた出力ハンドラを変更するために、信号ハンドラによって受け取った信号によってダイナミックに再構成されてもよい。

【００２９】

インターフェイスハンドラは、どの入力データソースが存在するかによって、複数の異なった手法ハンドラおよび関連する手法の各々に対する所定の入力条件を持つことができる。分子は、インターフェイスハンドラ内に、１つまたはそれ以上の入力ハンドラからの指示を待つ複数のルックアップテーブルを持つ。インターフェイスハンドラは、入力ハンドラと手法ハンドラとの間の通信を進行させる前に、ルックアップテーブル内で特定されたすべての入力が集まるまで、待機する。

【００３０】

インターフェイスハンドラはいつ所定の条件が満たされるかを決定するために多数の入力条件を監視する任意の適切な手段を用いてもよい。この機能はビットアレー、バーテーブル、スイッチアレーと同等なソフトウエアによって行われてもよい。すべての必要とされる入力条件が満たされたとき、インターフェイスハンドラ４４はそれぞれの手法ハンドラ４６を呼び出し、関連する手法４７を呼び出すための適切なレフェレンスを手法ハンドラに送る。図３に示す例では、入力ハンドラ４２ａと４２ｂは、正しいポインター／データソースが待ち行列に入れられたことをインターフェイスハンドラ４４に通知し、その結果、ビットアレー内の特定のフラッグが入力条件“Ｃ”を示すために“１”に設定され、入力ハンドラ４２ｃは入力条件に対するフラッグビットを“Ｆ”に設定する。ビットアレーの第３列内の４つのフラッグビットが他の入力条件の存在、例えば、信号ハンドラ４０に送られた信号によって“１”に設定される。すべての所要のフラッグビットがビットアレーの第３列で“１”に設定されると、インターフェイスハンドラ４４は経路“Ｇ”を開き、適切な手法ハンドラ４６を呼び出し、そして、手法ハンドラ４６は、出力ハンドラ４８への入力を処理するために関連する手法４７を呼び出す。

【００３１】

図４には、インターフェイスハンドラのロジック演算の一層詳細な例が描かれている。インターフェイスハンドラ４４は“ルックアップ＃１”と示された第１条件に必要な所定の入力を格納している。ルックアップ＃１は、どのような事象（“全てのデータ存在”、“時間切れ”、“信号”を含む）がインターフェイスハンドラ４４に対する所定の条件の下で発生するかを特定する。インターフェイスハンドラ４４は１つまたはそれ以上のルックアップを有してもよい。この例では、ルックアップ＃１は３つのデータソース、すなわち、共有したメモリデータＡ、ハードウエアメモリデータＢ、およびネットワークデータＣ

の存在に基づいたインターフェイスハンドラを描いている。ルックアップ＃１では、インデックス１は共有したメモリデータＡが存在するか否かに依存してオン／オフに設定され、同様に、インデックス２はハードウエアメモリデータＢに対して設定され、インデックス３はネットワークデータＣに対して設定される。すべてのデータが存在すると、インターフェイスハンドラは適切な手法ハンドラ４６ａのために“全てのデータ存在”を呼び出す。デフォルト条件として、“時間切れ”のためのインデックスは、すべてのデータ入力条件が満たされないで特定の時間間隔が経過したならば、手法ハンドラ４６ｂを呼び出すために“オン”に設定される。他のデフォルト条件として、“信号”に対するインデックスは、すべてのデータ入力条件が満たされる前に特定の信号が信号ハンドラ４０によって受信されたならば、手法ハンドラ４６ｃを呼び出すために“オン”に設定される。ルックアップ＃２は２つのデータソース、すなわちハードウエアメモリデータＢとネットワークメモリデータＣに依存するものとして示されている。インターフェイスハンドラは常時どの入力ハンドラ４２がデータを持っているかを知っている。なぜならば、入力ハンドラが新しいデータを受信したとき、またはデータがもはやないとき、そのことがインターフェイスハンドラに知らされるからである。

【００３２】

再び図３を参照すると、所定の入力条件が満たされたとき、それぞれの手法ハンドラが呼び出され（所定の条件の３つの組み合わせに対して形式“Ｈ”、“Ｋ”および“Ｎ”）、関連する手法４７または入力データ上で実行される処理ステップを呼び出し、入力ハンドラ４２から入力データを手法４７に与える。入力データは経路“Ｇ”を介してブロックとして、またはストリーミングデータとして取り扱われる。もし入力データがストリームされるべきならば、手法ハンドラは手法ハンドラからクエリー（照会）“Ｉ”によって取り扱われるストリーミング用の入力経路を設定し、入力ハンドラから回答“Ｊ”を入力する。データのブロックに対しては、手法４７は出力Ｍを手法ハンドラ４６に返し、手法ハンドラ４６はそれを出力ハンドラ４８に送り、次に、その後の一掃の面倒をみる。ストリーミングのデータに対しては、出力Ｍは、手法の終了までに、出力ハンドラ４８に直接送られる。手法の例は算術加算の計算、データの分類、ブール演算等であってもよく、第３者のアプリケーションの全体を含んでもよい。

【００３３】

図５および図６は２つの手法ハンドリング（取扱）形式、すなわち、ブロック手法ハンドリングおよびストリーミング手法ハンドリングの例をそれぞれ示す。ブロック手法ハンドリングでは、手法ハンドラ４６は、インターフェイスハンドラ４４から、手法４７を呼び出すために必要とされる到来データへの照会を受け取る。次に、手法ハンドラ４６は入力ハンドラ４２からの到来データを照会し、ブロック手法４７を呼び出し、回答を手法４７に送る。次に、手法ハンドラ４６は手法４７がその結果を返すのを待つ。そして、手法ハンドラ４６は出力ハンドラ照会を出力ハンドラ４８から得て、結果を出力ハンドラに送る。もし照会が存在しないならば、手法ハンドラは出力ハンドラを形成（呼び出す）ことができる。次に、データは出力ハンドラ４８に送られ、手法ハンドラ４６は停止する。

【００３４】

ストリーミング手法ハンドラでは、手法ハンドラ４６は、インターフェイスハンドラ４４から、手法４７を呼び出すのに必要とされる到来データへの照会を受け取る。データ送信用の出力ハンドラ４８の照会が同様に手法ハンドラ４６によって得られ、または形成される。所定の条件が満たされると、ストリーミング手法４７が呼び出され、入力および出力ハンドラ照会がストリーミング手法４７に送られる。ストリーミング手法４７は、次に、入力ハンドラ４２からデータを受け取り、それを処理し、それを出力ハンドラ４８に直接送る。このストリーミング形態で手法ハンドラを利用することにより、分子は、ロジックウエブが手法４７のダイナミックな鎖を一列にすることを可能にする“接続エンジン”として働く。

【００３５】

出力ハンドラ４８は、プログラマによって決められるが、出力が与えられる形式のメモリ

環境内でデータチャンネルへの適切な接続を達成する。もし出力用のデータチャンネルが出力ハンドラに対して構成ファイル内で特定されていないならば、出力ハンドラは存在する条件に応答して実時間においてデータチャンネルを形成できる。かくして、設計によって、出力ハンドラは任意の形式のメモリモデルに対するインターフェイスとなる。出力ハンドラは、選んだ出力ハンドラの形式に従ってプログラマによって決められるが、キューが許容する程度まで送信データソースを待ち行列に入れる。

【００３６】

出力ハンドラを用いる利点は、同一の分子に集められて処理された入力データに対する多数のメモリモデルへの同時分散を可能にすることである。多数の形式の出力ハンドラがあってもよい。共有メモリコンピュータシステムに対して入力および出力ハンドラを用いることは非常な利点である。実際のデータを移動させる代わりに、データに対するポインタや処理を、１つの分子内で、または１つの分子から他の分子に、作動しようとしている他の分子、すなわち、次の分子のために、移動させることができる。このことにより、分子には従来のオブジェクト指向プログラミング（ＯＯＰ）手法に優る速度上の利点が与えられる。ＯＯＰ手法はポインタを送ることができるが、ポインタはポインタを探すためにＯＯＰプログラムを介して処理されねばならない。対照的に、分子は他の分子から未処理のポインタを受けることができ、直接次の作業に進むことができる。

【００３７】

プログラマによって決められるが、各分子にはマイクロコンポーネントが設けられており、マイクロコンポーネントにより分子は自律的に働いて定義した手法により所要のデータを意図した出力に処理する。データはロジックウエブが構成されたどのような共有環境においても特定のデータチャンネルを通して分子へまたは分子から移動される。もしデータが利用できないならば、分子は単に待機し、次に何を行うかを指示されない。各分子は固有の名前を与えられてもよく、それは名前レジストリに保持される。任意の分子はその分子を固有の名前で呼び出すことができ、同一の名前の他の分子は同一のレジストリには形成されない。

【００３８】

分子の処理は終了信号を信号ハンドラ４０に送ることによって何時でも終了される。分子は、存在する任意の個所で、停止され、信号ハンドラ４０に送られた再構成信号によって異なってモードまたは手法に再構成される。分子には、また、その現状の構成（状態情報）を書き改めるために信号が送られる。その結果、処理の状態が即時に任意の個所で分析できる。対照的に、従来のプログラムは状態情報をダイナミックに変化させること保持することができず、何がある時刻に実行されているかを明らかにすることができない。

【００３９】

分子の５つのサブコンポーネントの任意のものまたはすべてが“消去”信号を受け、かつ選択した形式のハンドラは処理の状態に関する保持情報を支持している場合には、ハンドラはリロードできる分子の構成ファイルに状態情報を書き込む。かくして、もしシステムが何らかの理由により機能しなくなると、状態情報を分子から得ることができ、各分子およびロジックウエブ全体の状態を再形成するのに用いられる。このことにより、システムの管理者はどこにまたどのような故障が生じたかを突き止めることができ、また、故障が直された後処理を再開するために、その構成を他の分子にロードできる。処理（ロジックエウブ）の“状態”を再構築する能力は他のプログラミングパラダイムとは異なっている。システムの故障の場合、一般には、処理全体が最初から再開されねばならなく、そして故障に到達するまで、一度に１つの実行を通してステップを進めなければならない。本発明のプログラミング方法では、ロジックウエブを用いて、処理は故障の直前で再形成され、判断速度が著しく速くなる。

【００４０】

分子の構成ファイルは、例えば、“１度の実行”、“データがなくなるまで実行”、“消去の信号が送られるまで実行”のような実行状態を分子が実行するかを特定する。分子は、再度の実行を完了したとき、入力および出力ハンドラにもはやデータが存在しないとき

、または信号ハンドラが信号環境から分子を消去する信号を受け取ったとき、自分自体を消去してもよい。このことはソフトウエアファイルを消去することおによって、またはその実行可能なコードを不能にすることによって達成できる。ロジックウエブの一体性は分子毎に消去を行うことによって維持される。他のハンドラを破損させる恐れのある正しくないデータや状態情報は終了後は残されない。分子レベルでの消去により、効率的なメモリの使用を可能し、ルーチン全体を周期的に消去する必要性をなくすことが可能である。分子は所要の実時間の間にロードされ、それによって、分子の終了の際には機械リソースを切り離す。ストリーミング手法に対しては、機械リソースが切り離された個所において、データが与えられる限度に、利用できるリソースを用いてデータを流す分子が形成できる。もし分子の必要な寿命がダイナミックに決定されるならば、プログラマは、消去信号を受け取るために、信号ハンドラに対して適切な信号環境を傍受することを指示できる。消去信号は分子自体を消去するために分子のサブコンポーネントの各々に送られる。なお、消去信号は将来の参照のための状態を書き出すことを含んでいてもよい。各分子に関して消去する（各データ処理のサブステップ）このプログラミング方法の固有の能力により機械リソースの効率的な使用がなされる。他のプログラミングパラダイムでは、いくつかの消去が実時間中に生じるが、機械リソースはアプリケーションが存在する間は切り離せない。

ロジックウエブプログラミング方法

本発明のプログラミング方法は処理タスクの全体のインクレメント（増分）ステップを実行するために１つまたはそれ以上のロジックウエブの構成を要求するものである。１組の増分処理ステップを通して処理タスクを達成するプログラムの設計は、プログラミング分野における技術、工夫、芸術の問題であり、本明細書の記載の範囲を超えたものである。新しいプログラミング方法の基本的な特徴は、データ処理ステップを自律的に実行する複数の分子から成るコンピューティングリソースに接続されたロジックウエブの形成である。ロジックウエブの構成は実行が必要とされる処理タスクに依存する（一例が以下に題“データ同期例”として記載されている）。ロジックウエブの形成における基本の要素は前述した基本の５つのサブコンポーネントを持つ少なくとも１つの分子の形成である。

【００４１】

分子をプログラムするために、プログラマには分子を形成するためのハンドラ形成のライブラリが与えられる。プログラマは、最初データが得られるデータソース、例えば、格納されたファイル、インターネットデレクトリ、ＪＳＤＴ（商標）スペース（ヴぁ里ふぉル二ア州、マウンテンビュー所在のサンマイクロシステムの商標）、ＩＢＭ　Ｔｕｂｕｌｅ（商標）（ニューヨーク州、アーモンク所在のＩＢＭコーポーレーションの商標）を決定し、データを処理するためにどの方法を用いるかを決定する。プログラマは、データが存在するまたは存在することになるデータ環境に適合する適切な入力および出力ハンドラを取り上げ、次に適切なインターフェイスハンドラを選ぶ。プログラマは、手法ハンドラを呼び出すために満足される入力条件を決定することによってインターフェイスハンドラを構成する。次に、プログラマは呼び出される手法の名前を選び、処理されたデータが送られる環境にどの出力ハンドラが適合するかを選ぶ。プログラマは、分子が一度だけ実行する、データがなくなるまで実行する、または消去の信号が送られるまで実行するか否かを決定する。また、プログラマは、時間切れ期間を用いるか否か、時間切れ手法はどのようなものか、ある形式の信号を受信したときの呼出手法を決定する。最後に、プログラマは、どのような追加、または次の分子がロードされるか、この分子または他の分子をリマップするためにどんな信号が用いられるかを定義する。

【００４２】

この結果、１つの分子ユニットが形成される。プログラマは、１つの初期化ファイルとしてそれをセーブ（待避）し、それに名前レジストリにログ（記録）される固有の名前を与える。プログラマは、各分子がロジックウエブに形成されるように同様な処理を継続する。プログラミング方法のこのパラダイムがさらに進展されるにつれて、最適化した分子およびロジックウエブ全体に対する標準の初期化ファイルがプログラミング利用のためのラ

イブラリに格納できる。ハンドラのライブラリまたはリストの進展により、コードの列を書くことなしに、素人でも十分に機能する分子やアップリケーション全体を形成できることが期待できる。

【００４３】

本発明のプログラミング方法によってデータ処理タスクをプログラミングすることで得られるものは、集中処理管理の必要性なしで、割り当てられる任意のコンピューティングリソースにおけるデータを自律的に処理できる分子のロジックウエブを形成することによって実行するプログラムである。さらに重要な特徴は、実時間の変更、プログラムの開発や拡大、またはプログラムの故障の修理のようなデータ処理機能を変更するためにロジックウエブの１つまたはそれ以上の分子をダイナミックに再構成することができることである。

【００４４】

処理タスクにおけるステップの実行は信号ハンドラの生成動作を通して達成される。信号ハンドラが形成されると、その構成ファイルを読み込む際に、それは分岐し、構成ファイル内で指示される任意の他の分子を形成する。信号ハンドラはその分子を他の分子の名前にリマップしてもよく、または他の手法が呼び出されてもよい。かくして、分子の各層は“それ自身のほかに”ロジックウエブを構築する。その結果、分子の連続的な層が形成され、利用できるデータソースが条件を満たすのを待つことになる。

【００４５】

図７Ａ～７Ｆは本発明のロジックウエブプログラミング方法と従来のオブジェクト指向プログラミングとの比較を示す。この例は所定の検索入力に従って自動車のデータを検索するデータ処理タスクの最も単純な比較を用いる。データ処理プログラムは、自動車のタイプ、型、スタイル、運転特性、操作特性、色、ブレーキ特性及び加速特性の１つまたはそれ以上のカテゴリー（分類）によるデータベースの検索を実行するように設定されている。ロジックウエブプログラミング方法の効果が図７Ａ～７Ｃに示されており、オブジェクト指向プログラミング方法が図７Ｄ～７Ｆに示されている。

【００４６】

図７Ａで、自動車の分子がロードされる。自動車の分子が検索入力Ａ、Ｃ、Ｄで充足されると、それは操作分子の形成を要求し、操作分子が運転分子の形成を要求する。入力が運転分子に伝達され、運転分子がそれらを処理し、その結果を戻す。図７Ｄで示すオブジェクト指向プログラミング方法では、自動車のオブジェクト全体がロードされ、入力Ａ、Ｃ、Ｄが自動車のオブジェクトに通される。自動車のオブジェクト、すなわち、自動車のタイプ、操作、運転の階層的要素、またはクラスが入力を処理し、結果を発生させる。

【００４７】

図７Ｂは、ロジックウエブプログラミングパラダイムでは、操作の機能特性を変化するためにはいかにして実行中の操作を停止する必要がないかを示す。ガソリンと電気（電力）の分子が追加され、操作にアクセス可能にされる。リマッピング信号は、加速が呼び出される場合に加えられる新しい分子に適合するように加速の分子に送られる。図７Ｅでは、操作の実行が終了されねばならない。自動車オブジェクトの全体、ガソリンおよび電気オブジェクトのために取得されるソースコードがヒエラルキー内の該当場所で編集され、加速オブジェクトがヒエラルキーの変化に適合するように編集される。次に、オブジェクトコードは再コンパイルされねばならない。その場合、操作で用いられる任意の他のソースコードまたはオブジェクトコードも編集され、再コンパイルされ、再リンクされねばならない。

【００４８】

図７Ｃの最後の例では、ロジックウエブプログラミング方法での加速の分子は現在実行状態にある。加速の分子に新たな入力Ｂと存在する入力Ｃが充足されると、ガソリン分子が呼び出され、データが伝達されて処理され、その結果が戻される。図７Ｆのオブジェクト指向の例では、操作が再開され、自動車オブジェクト全体が再ロードされる。入力ＢとＣが自動車オブジェクトに通され、自動車のタイプ、運転、加速、ガソリンの階層的要素、

またはクラスが入力を処理し、結果を発生する。このことは、ロジックウエブプログラミング方法では、システムがシステムの実行中に加えられる新しいパラメータにダイナミックに適応できることを示している。このことにより、実行操作を継続するためにリコード（再符号化）し、コンパイルする必要性をなくしている。

コンパイリングおよび実時間操作

ロジックウエブプログラミング方法では、処理タスクの全ては、最終の結果に至るデータソースの所要の処理を自律的に達成できる分子を相互作用するロジック“ウエブ”を設計することによって達成される。分子のウエブは、プログラマによって書かれた、処理タスク全体の達成に参加するコンピューティングリソースの間に分散された初期化ファイルから実時間において形成される。

【００４９】

初期化ファイルは、処理タスクを開始するために分子の初期コーホートを発生する。ローカルＣＰＵの利用できるコンピューティングリソースと、関連するデータベースを用いて、分子は、終了、データなしでの実行、再構成まで、操作が維持される。操作中、分子はいつでも他の分子を呼び出すのに用いられる。このように、分子はダイナミック環境中で自律的要素のように働く。しかしながら、でたらめというより、その操作は目的をもったものであり、即座に並列処理となるものである。

【００５０】

コンピューティング処理のこの新しいパラダイムの結果、極めて効率的な分散処理ばかりでなく忠実な並列プログラミングと処理が得られる。どのような形式または操作システム環境であっても、ネットワーク上の任意の機械を利用できることに加えて、異なったソースからのデータを即座に処理する。

【００５１】

好ましい実行では、ハンドラは固有の形式の名前を割り当てられ、オブジェクトまたはソースコードで書かれる。同一の形式のハンドラが、出会うことが予想されるプログラミング環境の多数の形式に対して書かれてもよく、意図するローカルコンピューティングリソース上で実行するように正しいコードの形式を呼び出すことができる。例えば、それらはローカルコンピューティングリソースのオペレーティングシステムに用いられるプログラム言語で書かれてもよく、オペレーティングシステム環境で実行可能なコンポーネント（ＪＡＶＡ仮想機械）上で実行するためにＪＡＶＡ（商標）（カリフォルニア州、マウンテンビュー所在のサンマイクロシステムズの商標）のような移植性の高い全てを網羅する言語で書かれてもよい。

【００５２】

ロジックウエブの分子を発生するための初期化ファイルは、ディスク上にまたはメモリ内に格納された、またはローカルコンピューティングリソースにネットワークを介して送信された比較的小さいファイルとして物理的に存在する。初期化ファイルはローカルコンピューティングリソースのＲＡＭに読み込まれ、他のコンパイルされたプログラムと同様に実行される実行可能なファイルとして動作する。もし分子がインターネット環境で展開されるならば、コードファイルはＵＲＬアドレスから取り出され、またはローカルデレクトリ、例えば／ｕｓｒ／ｌｉｂ／ｊａｖａから得られる。

【００５３】

初期化ファイルは、コンポーネントハンドラの名前を呼び出すことによって分子の初期コホートの各々をアセンブルするように動作する。ハンドラ名が呼び出されると、ソースまたはオブジェクトコードを含むファイルは格納されたハンドラライブラリから（内部または外部から）取り出され、分子を形成するようにアセンブルされる。このことにより、オブジェクト全体が完全にアセンブルされたブロックでまたは他のプログラム言語（コンパイルされねばならないソースコードを書く）で書かれるような従来のＯＯＰ方法よりもはるかに融通のきく実時間動作が得られる。

【００５４】

図８Ａ－８Ｄは３つの形式の分子、すなわち、ブロック処理、ストリーミング処理および

ブロック処理のストリーミングの実時間実行をステップ毎に示す。図８Ａでは、信号ハンドラ４０、入力ハンドラ４２、インターフェイスハンドラ４４、手法ハンドラ４６、手法（手法要素）４７、出力ハンドラ４８を含む分子の基本パーツが示されている。分子のための初期化ファイルは、最初、信号ハンドラ４０を構築する。信号ハンドラ４０が始動し、プログラム全体に対する名前レジストリに知らせる。もし名前が既に存在するならば、分子を形成する要求を無視するか、存在する分子をリマップする。集中形式の名前レジストリを維持する必要性をなくすためにダイナミックな名前コンベンション（協議体）が用いられてもよい。次に、信号ハンドラ４０は入力および出力ハンドラ４２と４８およびインターフェイスハンドラ４４を呼び出す。この時点で、入力を待っている実行中の分子がある。任意のステップにおいて、信号ハンドラは入力および出力ハンドラ、インターフェイスハンドラ、方法ハンドラの属性の全てのリマップを命令するまたは待ち号列に入れる、またはユニットを停止し、再開し、または終了することができることに留意すべきである。

【００５５】

図８Ｂで示すブロック処理に対して、分子は、ステップＡで、入力ハンドラ４２がインターフェイスハンドラ４４にデータを持っていることを登録するのを待っている。インターフェイスハンドラ４４が充足されると、手法ハンドラ４６を呼び出す。ステップＢで、手法ハンドラは入力ハンドラのレファレンス（参照）、出力ハンドラの名前、手法の名前をインターフェイスハンドラから受け取る。手法ハンドラはブロックとしての入力データを入力ハンドラに照会する。ステップＣでは、手法ハンドラは手法４７を呼び出し、入力データ回答を送り、手法が結果を戻すまで待つ。ステップＤで、結果が戻されると、出力ハンドラ参照が手法ハンドラによって取得される。もし出力ハンドラ参照が存在しないと、手法ハンドラは出力ハンドラ４８を形成する。ステップＥでは、出力ハンドラがデータをその出力環境に送り、分子は入力ハンドラがより多くのデータを待ち行列に入れるための待機に戻る。

【００５６】

図８Ｃに示すストリーミング処理に対しては、分子は、ステップＡで、入力ハンドラがインターフェイスハンドラ４４にデータを持っていることを登録するのを待っている。インターフェイスハンドラ４４が充足されると、手法ハンドラ４６を呼び出す。ステップＢで、手法ハンドラは入力ハンドラのレファレンス（参照）、出力ハンドラの名前、手法の名前をインターフェイスハンドラから受け取る。手法ハンドラ（ストリーミング処理のために選ばれている）は出力ハンドラ参照を取得し、または、もし出力ハンドラ参照が存在しないと、手法ハンドラは出力ハンドラ４８を形成する。ステップＣでは、手法ハンドラは手法４７を呼び出し、入力および出力ハンドラの参照を送る。手法は、入力データキューが空になる、または出力ハンドラがもはや存在しなくなるまで、入力および出力ハンドラ間を流れる。ステップＤでは、最後の手法の結果が出力ハンドラに送られる。手法が結果を戻すまで待つ。ステップＤで、最後の手法結果が出力ハンドラに送られる。ステップＥでは、出力ハンドラがデータをその出力環境に送り、分子は入力ハンドラがより多くのデータを待ち行列に入れるための待機に戻る。

【００５７】

図８Ｄに示すブロック処理のストリーミングに対しては、ステップＡで、分子は入力ハンドラがインターフェイスハンドラに登録するのを待っている。充足されると、手法ハンドラを呼び出す。ステップＢでは、手法ハンドラはインターフェイスハンドラから入力ハンドラ参照、出力ハンドラの名前、手法の名前を受け取り、出力ハンドラ参照を取得し、または出力ハンドラを形成する。ステップＣでは、選ばれた手法ハンドラが入力ハンドラを待ち行列に入れ、入力データを持つ手法を呼び出し、結果を待ち、結果を出力ハンドラに送る。もし入力ハンドラのキューが空でないならば、手法ハンドラは、キューが空になるまで、または出力ハンドラがもはや存在しなるまで、ステップを繰り返す。ステップＤでは、最後の手法の結果が出力ハンドラに送られる。ステップＥでは、出力ハンドラがデータをその出力環境に送り、分子は入力ハンドラがより多くのデータを待ち行列に入れるた

めの待機に戻る。

【００５８】

ロジックウエブの動作では、１つのコンピューティング機械で始められた処理は別のところに移し、ネットワーク上の他の利用できる機械で、または利用できるリソースを持つ多数の機械上で動作を継続する。ロジックウエブの部分だけが動作に不可欠なものである。

データ同期の例

いかにコンピュータ処理およびプログラミング方法がロジックウエブを形成するのに適用されるかを示すために、ある例が図９を参照して記載されている。ロジックウエブはマルチメディアのプレゼンテーションを取り扱うために一般に用いられているような異なったデータソースからのデータのストリーム（流れ）を同期する所定の処理タスクを達成できる。初期化ファイルは、入力構成ファイルを読み取り、新しい分子、すなわち、リソースマネージャー７１と属性ユーザインターフェイス７２を形成するオープン分子７０を形成する。リソースマネージャー７１は利用できるリソースについて機械に照会し、オープン分子が構成ファイルから決定した基準が与えられて、どれほど多くのデータソースを機械が取り扱うか（この例では、３）を定める。リソースマネージャー７１は３つのデータローダー７４ａ、７４ｂ、７４ｃとユーザ定義データローダー７４ｄを形成する。

【００５９】

属性ユーザインターフェイス７２により、入力構成ファイルから属性を選択できるばかりでなく、新しい属性の形成、ファイルのタイプおよびファイル名の選択が可能である。属性ユーザインターフェイスは、ユーザの要求に応じてカスタムデータローダを形成し、実行分子７３を形成する。実行分子７３はファイル名、ファイルのタイプおよびファイル属性を分散し、同期分子８０を含む新しい分子を形成する。

【００６０】

データローダー７４ａ、７４ｂ、７４ｃはデータを位置決めし、もし十分なメモリがあれば、データをメモリにロードし、または必要に応じて、データをセクションに分けてロードさせ、データを送る用意をし、データユーザインターフェイス分子７５ａ、７５ｂ、７５ｃを形成する。これらにより、ユーザは、各ストリームをいつ開始するかを決定するために、他のストリームの長さと整合するように特定のデータのストリームを計量するために、およびデータストリームの持続時間を決定するために、特定のデータのストリームの時間枠を補正することができる。データローダーは、また、与えられたパラメータに適合するデータストリームを発生し、新しい分子を形成するデータストリーマ分子７６ａ、７６ｂ、７６ｃを形成する。

【００６１】

ユーザ定義データローダー７４ｄは、ユーザがデータの圧縮の量を制御することを可能にする圧縮ユーザインターフェイス７７と与えられたパラメータに関してデータを圧縮するデータ圧縮器７８を形成する。データ圧縮器７８はデータユーザインターフェイスとデータストリーム７６ｄを形成する。

【００６２】

同期分子８０はパルス繰返し数の調整を可能にし、ストリーミングが開始されてからどれほど時間が経過したかのカウント（計数）を与える。ユーザは停止させ、再開させ、データストリーム内で前後に移動させるように、この時間計数を用いてもよい。同期分子８０は同期パルスを発生し、同期ユーザインターフェイスによって維持されたパラメータに従ってパルスを送るように利用可能なデータストリーマのリストを維持し、新しい分子を形成する。

【００６３】

同期分子８０は、また、多数の入力データストリームを特定の出力ストリームの数に組み合わせる層分子８１を形成し、表示分子８２とセーブ分子８３を含む新しい分子を形成する。表示分子８２はデータをスクリーン上に出力し、セーブ分子８３は、データをディスク、ネットワーク、テープ、または他の媒体に出力する。

【００６４】

意図した結果を得るためのコンポーネントの相互作用を説明する。ユーザはオープン分子７０のための初期化ファイルを選ぶ。オープン分子は属性ユーザインターフェイス７２の形成を要求する。また、オープン分子７０はリソースマネージャー７１を呼び出す。もし機械またはネットワークに使用中のリソースマネージャーが既にあるならば、リソースマネージャーがリソースを再分散するのに用いられる。ないならば、新しいリソースマネージャーが形成される。リソースマネージャー７１はどの程度多くのデータストリームを快適に取り扱うことができるかを計算し、データローダー７４（各データストリームに対して１つ：この例では３つ）の形成を要求する。入力データのファイルのタイプを調べ，次にそのタイプのファイルの包括的な大きさを調べ、さらに利用できるメモリリソースを調べることによってリソースマネージャーはデータローダーの形成を行い、これらのパラメータからメモリを破損することなく行うことができるローダーの数を作るかを決定する。

【００６５】

ユーザは、属性ユーザインターフェイス７２内の各利用できるデータローダー７４に対する１つの名前と１つのファイルのタイプを選ぶ。もしユーザが利用できるデータローダー７４より多いファイルのタイプを取り上げると、属性ユーザインターフェイス７２はリソースマネージャー７１から追加のデータローダー７４を要求する。データローダーの形成は、属性ユーザインターフェイスのリストにないファイルのタイプをユーザが要求できる程度に特注作成できる。この場合、名前とユーザ定義データローダー７４ｄへの経路が与えられる。これにより、システムに融通性が得られ、予定していないデータの形式を受け入れることができる。ユーザは、また、属性ユーザインターフェイス７２中の実行に送られる属性やパラメータを決定する。

【００６６】

実行分子７３は同期分子８０を呼び出し、同期分子は層分子８１の形成を要求する。層分子８１は、表示分子８２とセーブ分子８３の形成を要求する。同期分子８０は、データストリーマ分子によって知らされた、同期パルスを送る名前のリストを維持する。

【００６７】

実行において、ファイル名、タイプおよび属性がデータローダーに送られる。データローダーがファイル名、タイプおよび属性を得るやいなやデータローダーは充足され、データを見出しデータをメモリにロードする。もし十分なメモリが利用できないならば、データはセクションに分けてロードされる。次に、データローダーはデータユーザインターフェイス分子７５とデータストリーマ分子７６の形成を要求する。この処理は、ファイル名、ファイルのタイプおよび属性が満たされたデータローダーとユーザ定義データローダーの各々に対して生じる。

【００６８】

リソースが僅かになったことをリソースマネージャー７１が検出した場合、リソースマネージャー７１はデータ圧縮を呼び出すためにデータローダーに要求を送る。次に、データローダーは適切なデータ圧縮をデータ圧縮分子７８に送る。属性ユーザインターフェイスの１つが圧縮されたデータを要求する場合、データローダーはデータローダーはデータをデータ圧縮器に送る。圧縮ユーザインターフェイス７７は圧縮パラメータを調節するためにアクセス可能である。

【００６９】

次に、ユーザはデータユーザインターフェイス７５内の各データストリームの持続時間開始ポイントを選ぶ。ユーザは他のストリームの長さに適合するように特定のデータストリームの時間枠を補償する、すなわち、拡大または縮小してもよい。データストリーマにデータユーザインターフェイス７５から送られたパラメータが充足されたとき、データストリーマはその参照名を同期分子８０に送り、データがストリームする用意ができたことを知らせる。同期分子８０は、参照名を名前のリストに加える。リストは現状でのシステム内の任意のデータストリーマの参照名からなる。もしもはやデータストリーム内にストリームのために用意されるデータがないならば、参照名は同期分子８０のリストから除去される。名前のリストの参照の数は層分子８１のコンポーネントに送られる。

【００７０】
同期分子８０はそのパルスをデータストリーマ７６に伝達し始め、データストリーマがデータを層分子８１のコンポーネントに伝達する。もし層分子８１のコンポーネントがリスト上のすべてのデータストリームを受け取らないならば、層分子は、すべてのデータが同時に層分子８１に到達するまで、同期分子８０がパルスを遅らせるように要求する。層分子８１のコンポーネントはデータ表示分子８２に送り、セーブ分子８３を呼び出す。もし任意の時点で、ユーザが一時停止、停止、時間計数の後退のためにデータストリームを中断したいならば、ユーザは同期ユーザインターフェイス７９で行ってもよい。
【００７１】
データ同期例でロジックウエブプログラミング方法を用いることに対していくつかの利点がある。ダイナミック構成の能力に起因して、多数のユーザが同時に機械やネットワーク上で同期システムを用いることができ、同期システムの各々はリソースマネージャーによってデータ同期リソースに分散できる。このことを達成する能力は、通常は付加のロードバランスの特殊なソフトウエアアプリケーションを必要とする自動的なロードのバランスを達成する。ユーザは必要に応じて任意のユーザ定義データローダーを形成できるので、システムはどのようなデータフォーマットにも適合する。システムはデータを予めファーマット化し、予め処理する。また、システムは、必要に応じて、ユーザの中断なしに、いつ圧縮が必要であるかを決定する。また、データ圧縮器は充足条件に応答して自動的に圧縮する。データローダーのリストに挙げられたネットワークファイルのタイプに合うようにデータを再フォーマットする。ハードウエアのリソースを考慮すると、システムは分散され、プラットファームは独立である。コンポーネントのどのいずれもがＬＡＮ、ＷＡＮ、またはインターネット上の任意の機械に載せることができる。並列処理環境では、多数のＣＰＵ上に分散した多数の同期ロジックウエブが並列のデータストリームを共通の層に伝達するように調和される。
【００７２】
分子はロジックウエブプログラミング方法の基本的な要素である。それは、大きなアプリケーションに通常関連する必須の機能（メモリ管理、機能の呼出、消去を含む）を保持するものである。それは占有範囲が小さく、極めて融通性があり、ダイナミックに再構成できる。それはローカルコンポーネントネットワーキンばかりでなく異なったハードウエア環境上の遠隔のネットワーキングに対する接続として機能する。ロジックウエブプログラミングパラダイムによって、集中処理管理の必要性なしに、またはスケジューリングや調和を厳密に予想することなく、機能ユニット間のつなぎ目のない互換性を伴った真に自律的な処理が可能である。
【産業上の利用可能性】
【００７３】
本発明のコンピュータ処理およびプログラミング方法では、複雑な計算及びデータ処理タスクが任意のタイプのおよび任意のネットワーク上の任意の位置にある利用できるコンピューティングリソースを利用するコンポーネント分子からなるロジックウエブを設計することによって達成できる。したがって、分散コンピューティングおよび並列処理環境でタスク処理を実行するのに理想的に適したものである。ロジックウエブは、任意のタイプの古いシステム、異なった言語でプログラムされたアプリケーション、および任意のファーマットに格納された任意のタイプのデータと共に動作する分子で設計され、かくして、データ集約タスクばかりでなく全く新しいタイプのデータ集約システムの形成の極めて効率的なプログラミングが可能である。
【００７４】
ロジックウエブプログラミング方法は工業用の特定のインターフェイスにそのコンポーネントを適用する事によって任意の分野に適用できる。例えば、データ集約、モデリング、シミュレイション、データの視角化、アニメーション、翻訳およびモジュラー生成アプリケーションに適用できる。現在のインターネットに基づくアプリケーションに対しては、それは、簡単さ、これらの考慮を集約すること、サーバに基づくコンピューティングモデ

ル、固有の大きさの調整能力、ＪＡＶＡに基づくコア機能および拡張性、伝達のためのウエブブラウザー技術、テラバイトデータセットを処理する能力、集約機能を持つユビキタスデータモデル、特注で構築された機能、分散エンドユーザの指針および設計の確約、モジュラーシーケンスのデータフローおよび位相型理工アプローチを持って、現在の市場の要件に合わせることができる。

【００７５】

ロジックウエブプログラミング方法は、データセンターおよびシステム監視、可視化共同支配、決定支援システム、複雑なリレーショナル表示、および科学モデリング可視化のようなデータの可視化アプリケーションに特に適したものである。ロジックウエブソフトウエアは、並列、分散、同期、および追跡型オブジェクト指向シーケンスを伴った非線形シーングラフとして動作できる。ロジックウエブのモジュラーアプローチの利点は、可視インターフェイスと結合されたとき、特に明らかである。分子は手法が名前を付けるような単一の手法である。手法の接続または結合は分子に固有のものであるので、ロジックウエブは、インターネット操作とは殆ど無関係に、人間指向の機能名や可視キューによって迅速に組立てられる。ロジックウエブを単に実行することによって故障の時点で自己診断を呼び出す。最適の支援環境で可視的に監視されるので、プログラミング方法はコンピュータの素人にまでコンピュータを“プログラム”する能力を拡張する。適切な類推によって、必要な部品を認識し部品を寄せ集めることによるパイプと取付け具の組立てのレベルにまで複雑なデータ処理タスクのプログラミングを単純化できる。

【００７６】

【表１】

APPENDIX I

```
objectMolecule(String moleculeCallerName, String moleculeCalledName)
{
        /* Contributes to methods, methodTriggers, communication */
        objectNames[]                   = Object names ( session/client names )
        objectContexts[]                = Equivalent to package ( molecule is componate to
                                          a function
                                           collection of molecule, and is of relative Context)

        /* Socketing / Sessioning */
        objectContextKey{}              = controls scope of access to event, field, and method
                                           groups, names, locations, version labeling, establish
                                          status,

        {
                eventNeighbors{}                = List of comsuming molecules     (pass
                                                  Events/Data)
                eventNeighborsConsumeRoute{}    = List of which Local/Remote data sets the neighbor
                                                  molecules can consume (Routing);
                eventNeighborsProduceRoute{}    = List of where then consuming routing produces the
                                                  data sets local to the neighbor molecule;

                /* The objectMolecule.Deamon.thread should maintain a record of molecule channel
                IO*/
                fieldChannel{}                  = Table of which channels are active
                fieldChannelExposed{}           = Table defining channels inter/external exposablity
                fieldLocalStatic{}              = Table of static local memory routing
                fieldLocalStaticExposed{}       = Table of defining  static local memory routing
                                                  exposability
                fieldLocalDynamic{}             = Table of dynamic local channel routing
                fieldLocalDynamicExposed{}      = Table of defining local channel exposability
        }

        /* localRemoteSwitcher() */

        objectMethodAccessLabeling[]    = Determines Local or Remote access for methods
        objectMethod[]                  = Methods callable name (local to Java or via API)
        objectMethodMoleculeData{}      = Data from File.moleculeName that forfills
                                          Consuming/Producing Routing.
```

【００７７】
【表２】

```
        /*methodTrigger() data , Local/Remote flags*/

        objectWait[]                        = methodTrigger() lifespan, or lifespan dependancy
        objectMethodConsumingRouting[]      = Which Channel/Local data have methodTrigger()
                                              block on
        objectMethodConsumingFofillment[]   = Where a methodTrigger() can look for data local
                                              and or remote and if it can switch between the two
                                              and if so who takes priority.
        objectMethodProducingRouting[]      = Which Channel/Local space data is delivered to.

        a methodTrigger().thread is started for each objectMethod[] element. All interfacable
        molecule should have enough data to create a methodTrigger for objectDisplay().
        fieldChannel.LocalStatic.LocalDynamic{}. a localRemoteSwitcher().thread is started
        as a child to methodTrigger().thread as approriate, which is governed by
        objectMethodAccessLabeling[],objectMethodConsumingFofillment[]

}

objectDisplay()
{
        struct objectDisplayEnvi {

                displayEnviroment[]     = Reference to current interface environment, track
grounding, Screen tracking system
                displayState{}      = Switch modes Accept Events, Accept Args, Accept Pointer, .. etc
                                      Criteria relevant object/ Superscope/subscope


                displayAlphas[]         = zBuffersLayer's relative alpha (range 0.0 - 1.0) array allows
single object to multiple Depths
                                        or volume Depth assignment.
                displayZDepths[]        = zBufferLayer assignment, array allows single object to
multiple Depths
                                        or volume Depth assignment.
                displayZElements[]    = micro layering for a given zBufferLayer
                displayComposites[]   = Object accumulation ( add, subtract, dif, ..)
                displayLocations[]      = Global location when parent, relative location when child


                regPeriod[]       = Array to (x,y image) or (x,y,z volume)
                regDisplay{}        = Display scale as opposed to preObjectArray x,y,and/or z
dimensions and registration

                Array[]           = array from an ObjectDisplayEngine()
                EventArray[]          = array from an EventDisplayEngine()
                RegArray[]          = array from an RegistrationDisplayEngine()
        }
}

objectEnviroment()
{
```

【００７８】

【表３】

```
        Establish broadcasting pointer information channels and display through put channels.
}

-------Core Array Process Methods------

Type: Data/String

MultiArray  = array with N dimensions
SinglArray  = arrayX1,arrayX2,..,arrayXN
MixedArray  = arrayX1,arrayX2,array with 1-N dimensions,..,arrayXN

Algorithm:(API call, Java Class/Method Call)

        Encode:
                -consumer(type,MixedArray,format) produce(file)
                *raw to format (simple to many)
                *Multipl SinglArrays            to    aif,wav,txt
                *Multipl SinglArrays, or MixedArray to rgb,tif,jpg
        Decode:
                -consumer(file) produce(type,MixedArray)
                *format to raw (many to simple)
                *aif,wav,txt to Multipl SinglArrays
                *rgb,tif,jpg to Multipl SinglArrays, or MixedArray

        Cmd Line:
                -pass(String cmd,String argv[]) return(pid,status)

        Splitter:
                -consume(type,MixedArray[N],how) produce(type,MixedArray)

        Mixer:
                -consume(type,MixedArray[N],how) produce(type,MixedArray)

        Stream:
                -consume(type,SinglArray,how,rate) produce(channel VARstream)

        PacketStream:
                -consume(type,MixedArray,how,rate) produce(channel VAR[]stream)

        Packet:
                -consume(channel VARstream,how) produce(type,SinglArray)

        MultiPacket:
                -consume(channel VAR[]stream,how) produce(type,MultiArray)

        Formula w/ Logic controlling output:
                -consume(Elm & MixedArray,Sting equation) produce(Elm & MixedArray);
                *Composite Over,under,multiple,add,differance

        Range:
                -consume(type,MixedArray[N]) produce(type,min[X1->N],max[X1->N],avg[X1->N])

        Sort:
                -consume(type,SinglArray,key) produce(type,SinglArray)
```

【００７９】
【表４】

Element:
-consume(type,MixedArray[N],index[]) produce(type,MixedArray)

Crop:
-consume(type,MixedArray[N],indexStart[X1->N],indexEnd[X1->N])
produce(type,arrayX1->N)

Scale:
-consume(type,MixedArray[N],type,Factor) produce(type,MixedArray[N])

Sample:
-consume(type,MixedArray,step) produce(type,MixedArray)

Flip:
-consume(type,SinglArray,xdirection,ydirection) produce(type,SinglArray)

Filter:
-consume(type,SinglArray,) produce(type,SinglArray)

Rotate:
-consume(type,SinglArray,angle,direction) produce(type,SinglArray)

Interpolate:
-consume(type,MixedArray[N],interp_type) produce(type,MixedArray[N])

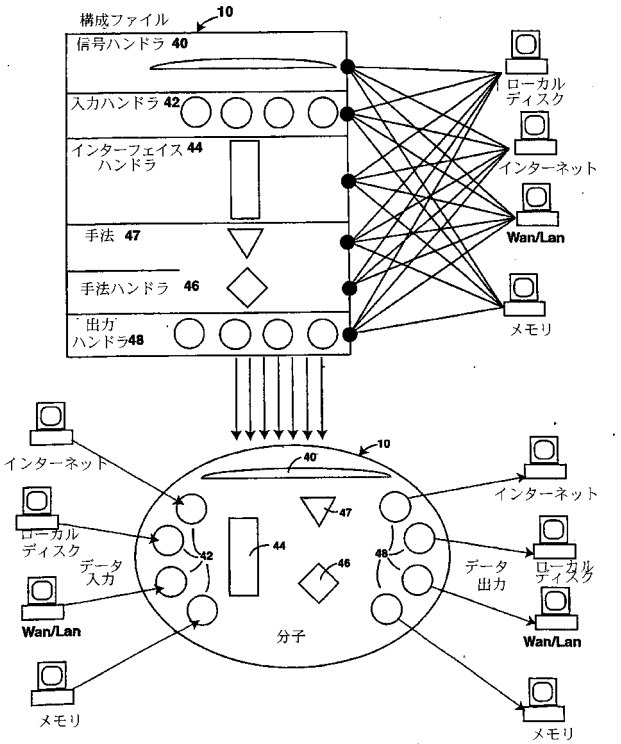
Tasks:

Data manipulation Process sequencing tool

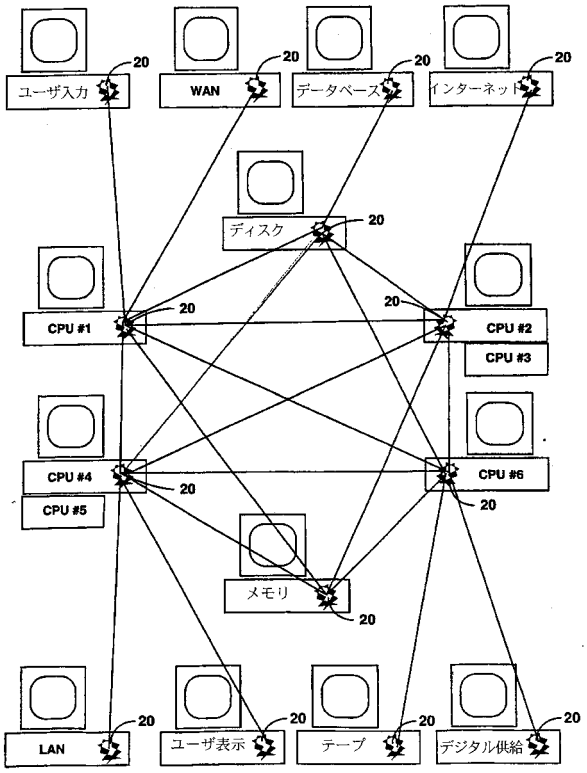
【図面の簡単な説明】
【００８０】
【図１】本発明のソフトウエアのエンティティ（ソフトウエアの構成要素）（“分子”）

の全体図である。
【図２】分散コンピューティングすなわち並列処理環境における分子のロジックウエブの配置を示す。
【図３】信号ハンドラ、入力ハンドラ、インターフェイスハンドラ、手法ハンドラ、および出力ハンドラのマイクロコンポーネント（小型構成）を持つ基本形式のソフトウエア分子用の構造体の好ましい例を示す。
【図４】ソフトウエア分子のインターフェイスハンドラの論理演算を詳細に示す。
【図５】２つの手法ハンドラ形式のうちのブロック手法ハンドラの例を示す。
【図６】２つの手法ハンドラ形式のうちのストリーミング手法ハンドラの例を示す。
【図７】本発明のプログラミング手法と従来のオブジェクト指向プログラミングとの比較を示す。
【図８】ブロック処理、ストリーミング処理およびブロック処理のストリーミングの３つの基本形式の分子のステップ毎のランタイム実行を示す。
【図９】いかに新規なコンピュータ処理およびプログラミング手法がデータ同期タスクを達成できるロジックウエブを発生させるために適用されるかを示す。
【符号の説明】
【００８１】
１０　ソフトウエアエンティティ（分子）
４０　信号ハンドラ
４２　入力ハンドラ
４４　インターフェイスハンドラ
４６　手法ハンドラ
４８　出力ハンドラ

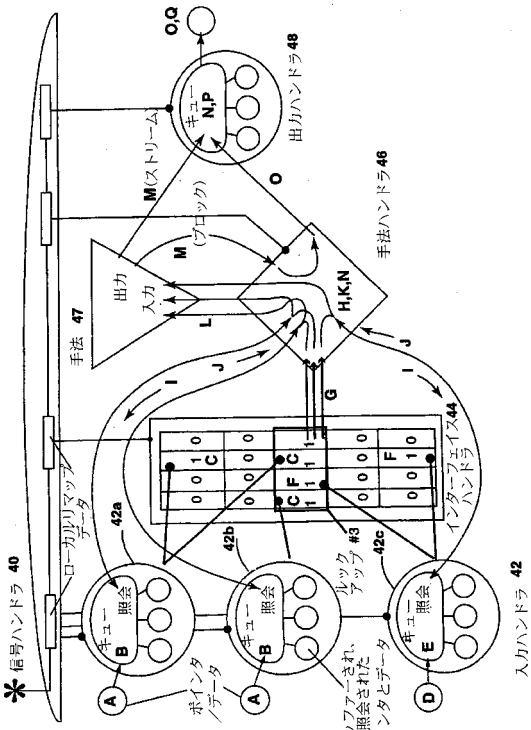
【図１】



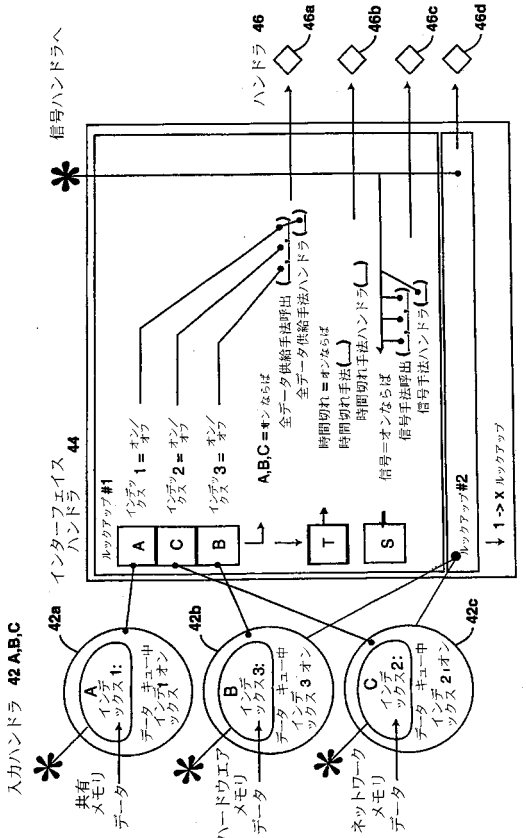
【図２】



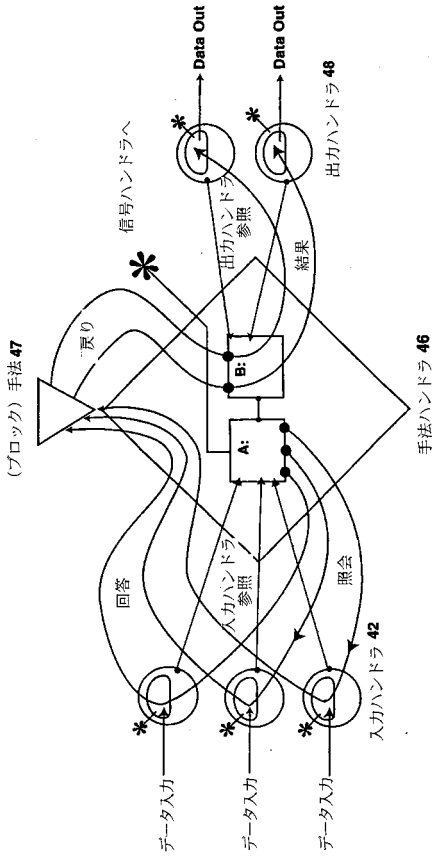
【図３】


信号ハンドラ 40
ローカルリマップデータ
手法 47
出力
入力
M(ストリーム)
M(ブロック)
O,Q
キュー N,P
出力ハンドラ 48
O
手法ハンドラ 46
H,K,N
J
I
G
L
インターフェイスハンドラ 44
42a
42b
42c
キュー照会
ルックアップ #3
ポインタ/データ
バッファーされ、照会されたポインタとデータ
入力ハンドラ 42

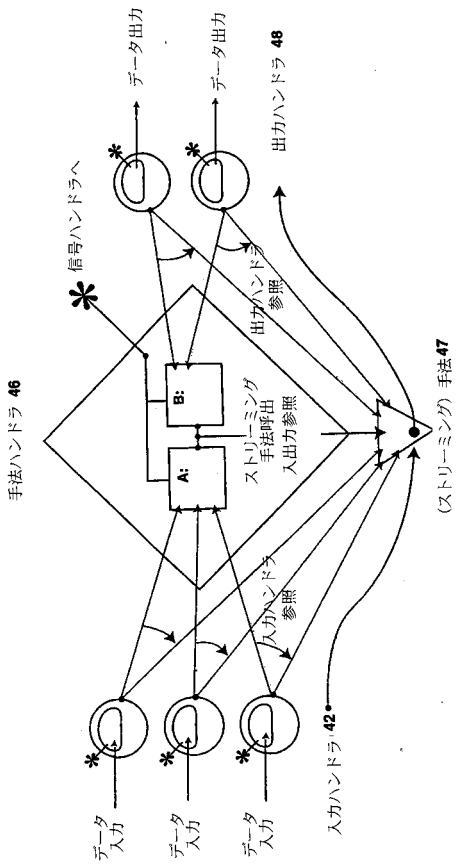
【図４】


信号ハンドラへ
ハンドラ 46
46a
46b
46c
46d
インターフェイスハンドラ 44
ルックアップ #1
インデックス 1 = オン/オフ
インデックス 2 = オン/オフ
インデックス 3 = オン/オフ
A,B,C = オンならば 全データ供給手法呼出
全データ供給手法ハンドラ
時間切れ = オンならば 時間切れ手法
時間切れ手法ハンドラ
信号 = オンならば 信号手法呼出
信号手法ハンドラ
ルックアップ #2
1 -> X ルックアップ
入力ハンドラ 42 A,B,C
42a
42b
42c
インデックス 1: データ キュー中 インデックス 1 オン
インデックス 3: データ キュー中 インデックス 3 オン
インデックス 2: データ キュー中 インデックス 2 オン
共有メモリデータ
ハードウエアメモリデータ
ネットワークメモリデータ

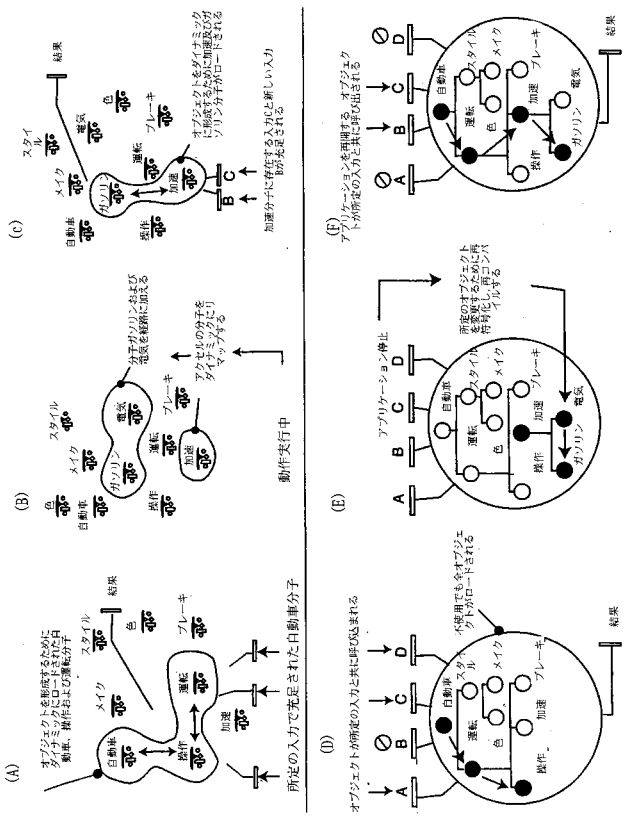
【図５】


Data Out
Data Out
信号ハンドラへ
出力ハンドラ 48
出力ハンドラ参照
結果
戻り
(ブロック) 手法 47
B:
A:
手法ハンドラ 46
回答
入力ハンドラ参照
照会
入力ハンドラ 42
データ入力
データ入力
データ入力

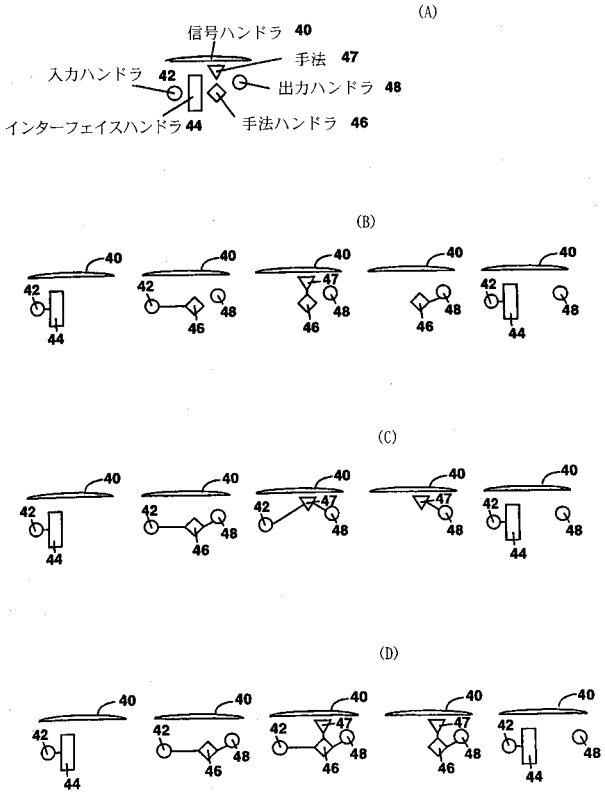
【図６】


データ出力
データ出力
信号ハンドラへ
出力ハンドラ 48
出力ハンドラ参照
B:
A:
ストリーミング手法呼出 入出力参照
手法ハンドラ 46
入力ハンドラ参照
入力ハンドラ 42
(ストリーミング) 手法 47
データ入力
データ入力
データ入力

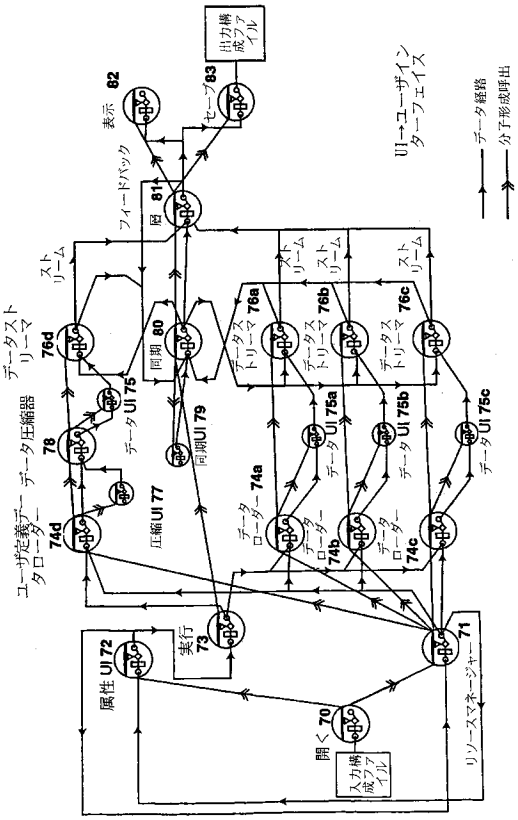
【図７】



【図８】


(A)
信号ハンドラ 40
手法 47
入力ハンドラ 42
出力ハンドラ 48
インターフェイスハンドラ 44
手法ハンドラ 46
(B)
(C)
(D)

【図９】


データストリーマ
ユーザ定義データローダー
データ圧縮器
ストリーム
フィードバック
表示 82
セーブ 83
出力構成ファイル
UI→ユーザインターフェイス
データ経路
分子形成呼出
属性 UI 72
実行 73
開く 70
入力構成ファイル
リソースマネージャー 71
圧縮UI 77
同期UI 79
同期 80
データUI 75
データUI 75a
データUI 75b
データUI 75c
データローダー 74a
データローダー 74b
データローダー 74c
データローダー 74d
データストリーマ 76a
データストリーマ 76b
データストリーマ 76c
データストリーマ 76d

【国際公開パンフレット】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
6 February 2003 (06.02.2003)

PCT

(10) International Publication Number
WO 03/010659 A1

(51) International Patent Classification⁷: G06F 9/44

(21) International Application Number: PCT/US01/41389

(22) International Filing Date: 23 July 2001 (23.07.2001)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicants and
(72) Inventors: **FABLES, Wylci** [US/US]; 214 Kawehi Place, Kula, Maui, HI 96790 (US). **PARK, Jore** [US/US]; 214 Kawehi Place, Kula, Maui, HI 96790 (US).

(74) Agent: **CHONG, Leighton, K.**; Ostrager, Chong & Flaherty (Hawaii), Suite 1200, 841 Bishop Street, Honolulu, HI 96813 (US).

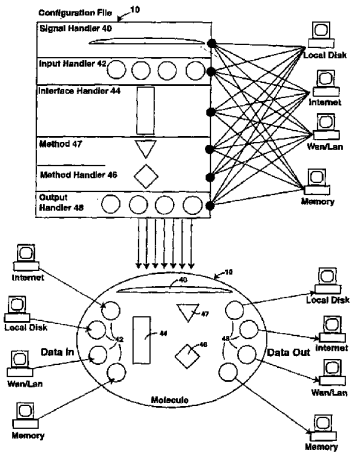
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) **Title:** COMPUTER PROCESSING AND PROGRAMMING METHOD USING AUTONOMOUS DATA HANDLERS



(57) **Abstract:** A computer processing and programming method calls for creating a plurality of software entities ("molecules 10") which can be dynamically configured to process data autonomously. The molecules can send and receive signals indicating the state of a molecule, whether or not a processing task is fulfilled, the results of a processing task, and whether or not the operation of a molecule is to be terminated, interrupted, reconfigured, or continued by creation of one or more "next" molecules. The molecules are created from a common set of software micro-components (40, 42, 44, 46, 48), which may be programmed in any programming language to run in any operating system environment. The molecules may reside with a single computing resource, however, they are ideally suited to be deployed with distributed computing resources of different types on a network or in a parallel processing environment. An overall data processing task is performed by creating molecules in a "logic web" which can dynamically adapt to fulfill its task. Logic webs can be assigned to operate with different legacy systems, with applications programmed in different languages, and with data of any type stored in any format. As a result, data processing tasks in distributed or parallel processing environments can be performed much more efficiently, and entirely new types of network computing or parallel processing tasks can be undertaken.

WO 03/010659 A1

COMPUTER PROCESSING AND PROGRAMMING METHOD USING AUTONOMOUS DATA HANDLERS

SPECIFICATION

Technical Field

This invention generally relates to computer processing and programming methods, and more particularly, to methods that employ a new type of autonomous data handlers.

Background of Invention

Computer processing has been developed from its earliest theoretical foundations to the present as a machine-oriented paradigm in which machine instructions are executed by a central processing unit (CPU) to convert raw data and input in sequential steps to a useful output. To accomplish more and more complex tasks involving greater and greater amounts of data, the development of computer hardware has proceeded with exponential increases in speed, power, and capacity, now approaching the physical limits of the atomic scale and the speed and wavelength of light. The programming of complex tasks by linear programming methods increasingly requires intensive code writing which can take up millions of lines of code and is susceptible to errors and cross-code conflicts which can require intensive debugging efforts. It also inherently requires that the programming be accomplished in one chosen programming language for a selected operating system, leading to obsolescence and legacy systems.

Object oriented programming (OOP) methods were developed to break up large programming tasks in modules that are self-contained and self-executing. Modules that have been optimized for specific functions can thus be catalogued and re-used in modular form in a variety of larger programming tasks. However, OOP methods require the enforcement of class definition hierarchies over a programming environment as a whole, resulting in closed systems and libraries of OOP modules that must be recompiled to support other environments.

Parallel processing methods were developed over the past decade in the attempt to obtain increased capability by breaking up a processing task into parts executed by an array of CPUs in tandem. However, conventional approaches to parallel processing have required centralized

management and control of programmed tasks which imposes stringent requirements in the coordination and synchronization of tasks. Parallel processing methods to date have also continued to employ linear programming methods which are intensive and susceptible to fault, especially when operating in parallel, and are captive to the chosen programming language and operating system.

In recent years, with the rise of large-scale distributed networks, there have been proposals to perform very large or complex tasks by breaking them up into small pieces and distributing them among cooperating computers on a network, often referred to as "distributed computing" or "network computing". The proposals for distributed computing represent a shift in the computing paradigm from the processing of tasks of high computational intensity towards one in which data are widely distributed among resources on a network and processed locally with lower-intensity CPUs through distributed efforts. However, since the same linear programming methods and centralized management and control have been used, the distributed computing paradigm to date has also required stringent programming and coordination of efforts to achieve worthwhile results. The failure of one or more individual processing efforts or time delays associated with communicating or coordinating with centralized processing management can produce bottlenecks which bog down the effectiveness of this approach.

One method which has utilized multiple asynchronous computer processes is called "Flow Based Programming" ("FBP"), as explained in greater detail in Flow-Based Programming: A New Approach To Application Development, by J. Paul Morrison, published by Von Nostrand Reinhold, New York, 1994. The reference describes Flow Based Programming as being based on three principles, "asynchronous processes, data packets with a lifetime of their own, and external definition of connections". In FBP applications are defined as networks of "black box" processes, which exchange data across predefined conditions. FBP is a process of stepping tagged data packets through networks of "black box" processes and thus is a data-centric process. The FBP approach needs extreme predictive capability to work well.

Therefore, it is a primary objective of the present invention to provide a computer processing and related programming method which decentralizes the management of computer processing tasks and allows distributed or parallel processing of data to proceed autonomously with adaptive options such as taking the most efficient path available.

It is a further object of the invention to provide a new programming method which is optimally designed to facilitate parallel or distributed data processing by avoiding the need for captive programming environments such that simultaneous use of multiple programming languages can be supported and centralized processing management is eliminated. It is also intended to reduce the intensity and case specific nature of programming required for complex processing tasks and the susceptibility of such programming to fault.

Summary of Invention

In accordance with the present invention, a computer processing and related programming method calls for creating a plurality of software entities (referred to herein as "molecules") which can be dynamically configured to process data autonomously. The molecules can send and receive signals indicating the state of a molecule, whether or not a processing task is fulfilled, the results of a processing task, and whether or not the operation of a molecule is to be terminated, interrupted, reconfigured, or continued by creation of one or more "next" molecules. The molecules are created from a common set of software micro-components, which may be programmed in any programming languages to run in any operating system environments.

The molecules created for an overall processing task may reside with a single computing resource (CPU and database), however, they are ideally suited to be deployed with distributed computing resources of different types on a network or in parallel processing environments. Each molecule may be programmed in a programming language or languages suited to the computing resource where it is intended to operate. The molecules can function with a mixture of compiled or run-time compiled code, allowing them to run on a computing resource with only a portion of the process adapted to run on that computing resource. The molecules can signal each other and send and receive data using common network protocols. They operate to complete their respective data processing tasks autonomously until the overall data processing task is completed. An overall data processing task can be performed by creating molecules and structuring their operations and interactions in a "logic web" which can dynamically adapt to fulfill the task.

Adapted to a parallel processing environment, the molecules can be dynamically configured to run autonomously on parallel computing resources. A logic web is defined to allow efficient access of each molecule to a data object it is intended to process, and to any available processor of the parallel processor array as an interim CPU resource. The scheduling of processor access can be handled by a main controller program, or by a separate scheduler molecule or mini logic web. The molecules are programmed to signal their state, result, and termination, reconfiguration or generation of next molecules. The dispersed, autonomous operation of the molecules avoids the rigid scheduling structures and time delay conflicts in conventional parallel processing methods, in order to accomplish data processing tasks more efficiently and quickly.

As a software-configured entity, a molecule is comprised of the following basic software micro-components: a signal handler, an input handler, an interface handler, a method handler, an associated method, and an output handler. The signal handler is capable of sending and receiving signals for invoking other molecules and signaling other molecules and resources on the network. The input handler receives data input and puts the data in a queue. The interface handler

determines when the inputs to the molecule satisfy a predefined condition in order to open a path to a method handler. The method handler processes the data sent from the input handler's queue according to a predefined method and sends an output to the output handler.

Adapted as a software programming method, a logic web of molecules is created to run on any available computing resources of any type and to process data in any format and in any location. Initialization files are programmed to create an initial host of molecules on the intended computing resources. The initial molecules operate to generate the next molecules in successive layers or waves representing a progression of program steps. Each initialization file is structured to assemble the software micro-components of a molecule from a library of standard micro-component types. These standard types are chosen objectively in terms of their function, and are not rigid class types in an OOP hierarchy. New types of software micro-components can readily be created as needed and programmed in the programming language suited to the environment they are intended to operate in.

With the computer processing and programming method of the invention, complex computational and data processing tasks can be performed by designing logic webs constructed of component molecules that utilize any available computing resources of any type and in any location including on a network. Thus, for example, a logic web can be designed with molecules to operate with different legacy systems, with applications programmed in different languages, and with data of any type stored in any format. As a result, data processing tasks in distributed or parallel processing environments can be performed much more efficiently, and entirely new types of data processing tasks can be undertaken.

Other objects, features, and advantages of the present invention will be explained in the following detailed description of the invention having reference to the appended drawings:

Brief Description of Drawings

FIGS. 1A is a general illustration of a software entity ("molecule") in accordance with the invention, and **FIG. 1B** illustrates deployment of logic webs of molecules in distributed computing or parallel processing environments.

FIG. 2 shows a preferred example of the structure for a software molecule of a basic type having signal handler, input handler, interface handler, method handler, and output handler micro-components.

FIG. 3 shows the logic operation of the interface handler of the software molecule in greater detail.

FIGS. 4A and 4B show examples of two method handling types, block method handling and streaming method handling, respectively.

FIGS. 5A - 5F illustrate a comparison of the programming method of the invention with conventional object-oriented programming.

FIGS. 6A - 6D illustrate the step by step run-time execution of three basic types of molecules, the Block Process, the Streaming Process, and Streaming a Block Process.

FIG. 7 illustrates how the novel computer processing and programming methods are applied to generate a logic web that can accomplish a data synchronization task.

Detailed Description of the Invention

In the present invention, a new computer processing and programming method calls for creating a plurality of software entities ("molecules") which are programmed to process data autonomously in communication with other molecules and available computing resources. The molecules are created from a common set of software micro-components programmed in any programming languages to run in any operating system environments. They are configured in a logic web of molecules to perform a data processing task, which may be a component of an overall processing task. A logic web, or a web of logic webs, may reside in a single computer environment, but is particularly suited to be used in parallel processing and distributed processing environments.

Referring to **FIG. 1A**, a general illustration of a software entity ("molecule") 10 in accordance with the invention is illustrated. Each software molecule 10 is basically composed of its configuration (data) files, signal handler 40, input handler(s) 42, interface handler 44, method (47) and method handler 46, and output handler(s) 48 (to be described in further detail under the heading "Software Molecule" below). The molecule's components are gathered from local or remote sources and combined to create the molecule (described in further detail under the heading "Programming Method" below). Each molecule is programmed to process a defined data object autonomously, and can send and receive signals to and from other molecules with which it interacts in a logic web to accomplish a processing task. The logic web is assigned to run on any available computing resource, and communicates with other molecules in other logic webs via a communications bus within a computing environment and/or by network in a network environment (indicated by the connecting lines in the figure). Through a communications link, a molecule can send and receive signals to its signal

handler 40, access data through any input source to its input handler(s) 42, and output data through its output handler(s) 48.

Referring to **FIG. 1B**, deployment of logic webs 20 of molecules is illustrated for distributed computing environment. A multitude of distributed computing resources, in the form of local computers, LAN/WAN servers, CPUs, network servers, databases, I/O devices, etc., are connected by network of any type (indicated by connecting lines). The network may be a local area network (LAN), a wide area network, and/or a global network of networks (the Internet). While an entire web of logic webs may reside at a single computing resource and communicate over a network with other computing resources having the data to be processed by the molecules, the efficiency of the system is greatly enhanced if the logic webs intended to process data at particular computing resources reside and process data locally at those computing resources and instead communicate their state information and results to other molecules via the network. Thus, for example, a molecule for processing COBOL-formatted data may reside at a COBOL server at which the COBOL-formatted data are stored, or a molecule for processing data in a JAVA space may reside at a computing resource having the desired data and a locally resident JAVA virtual machine. In this distributed mode, the locally resident logic web of molecules may be programmed in the programming language used at the computing resource where it resides, so that it can readily be operated in that environment.

A logic web, or web of logic webs, may similarly be deployed in a parallel processing environment having an array of CPU resources (indicated by CPU#1, ..., CPU#6) and associated databases and I/O devices. The logic web is structured to allow efficient access of each molecule to a data object it is intended to process. Access of a molecule to CPU computing resources may be scheduled by a resource manager molecule or logic web. Alternatively, it may communicate with a CPU scheduler such as conventionally used in parallel processing environments. Since the molecules are designed to process data autonomously, they can operate without having to be monitored and controlled by a centralized processing management. The molecules signal their state information, results, and termination, reconfiguration or generation of next molecules to other molecules or logic webs set up in the system. However, it is also possible to have the logic webs communicate with a conventional controller program in a hybrid arrangement.

Software Molecule

Referring to **FIG. 2**, a preferred structure for a software molecule of a basic type is shown having a signal handler 40, input handlers 42, a data interface handler 44, method handlers 46, and output handlers 48. The signal handler 40 is capable of sending and receiving signals to and from other molecules and other resources on a communications bus or network, similar to decentralized broadcasting of signals on a CB or private radio network. The signals are broadcast in a common

protocol used on the communications bus or network, e.g., the TCP/IP protocol used on for Internet communications.

The signal handler 40 can perform the following basic routines in the system:

1. Read the configuration file from the data source. The configuration file contains names, attributes and type of the molecule's sub-components.

2. Build the molecule by assembling the sub-components as defined in the configuration file. Loading or invoking a sub-component proceeds by calling the name and type for the component from a local or remote source having a library of files for the handler types stored. The component file is passed to the signal handler and added to the molecule.

3. Select the signal environment. The signal handler listens to the signal environment specified in the configuration file.

4. Kill, suspend or restart the molecule.

5. Send signals to and from the signal environment specified.

6. Interpret signals it receives, and deliver the signals to the appropriate sub-components of the molecule, including the capability to reconfigure. The signal handler can send requests to the sub-components. Examples of signal handler requests include: (a) to the input handler or output handler to reduce queue size; (b) to any of the five sub-components to write out their state and then die; (c) to cue or mandate a re-mapping of any of the five sub-components; (d) to modify buffer size or memory use.

7. Call to load the "next" or other molecule. The signal handler looks at specified name registries to see if the "next" molecule already exists. If it does exist, then the signal handler will cause that molecule to remap, or alternatively, if it is specified, it will not call the "next" molecule. If it does not exist, it calls for the creation of a new signal handler which will load the "next" molecule. The new signal handler is not a child thread, but rather a completely separate and unique new entity. This is differentiated from other programming paradigms, in which all threads are children to a primary parent thread, which together form an application.

The sending and receiving of signals of the signal handler can be implemented in any manner appropriate to the communications or "signal" environment of the computing resources where the molecule resides. In a conventional computer or network environment, the signal handler can issue or receive a "signal" as a data packet addressed to a specified molecule in the system or on the network. A signal from a signal handlers may be routed to a target molecule based upon its address as maintained in a name registry. The name registry may be maintained by another molecule or logic web, or a conventional resource register. For example, conventional data sharing environments, such as the JAVA™ Shared Data environment, have registers for servers, clients, and data channels set up for given applications. A signal handler can also "listen" to the signal environment for a global broadcast to all molecules, or it may tune into a specified channel as is listed in the configuration file,

or wait for any signal from any specified signal environment, or may be tuned along with a group of other molecules in a logic web to the same channel.

The handler components of a molecule are structured as a common set of types that can be invoked to create or dynamically configure a molecule. The molecule can be assembled by signaling the names of the desired handler types in order to retrieve the corresponding section of code from stored memory. In this manner, any number of types of input and output handlers can be accessed depending on the correct type of the data source or memory model. Similarly, a molecule can be configured with many different types of method handlers. When an interface handler is supplied with new configuration information, the new information as to which types of method handlers to invoke for given conditions is stored, so that the molecule can execute the new method when these conditions exist. The appropriate section of source or object code for any of the five component handlers of a molecule can be accessed from stored files via local memory, LAN, WAN, or Internet.

A rudimentary example of a range of handler types and attributes, such as may be referenced in the molecule configuration files, is shown in the listing provided in Appendix I. It is to be understood, however, that the number, types, and attributes of handlers that may be created are unlimited, and can be expanded commensurately with the needs of programmers for accommodating different types of memory models, data sharing environments, and computing resources.

The input handler 42 establishes the molecule's connection to data sources from specified types of memory models, as determined by the programmer. By design, the input handler thus allows interface to data sources of any memory model. It queues incoming data sources to the extent the queue will allow, as is determined by the type of input handler chosen, then registers with the interface handler 44 that it has queued data. The input handler also allows other sub-components of the molecule to query for its data. In the example shown, incoming pointers to memory or data "A" are accepted in the queue "B" of input handler 42a, and also of input handler 42b, and incoming pointers/data "D" are accepted in the queue "E" of input handler 42c.

The type of data source that may come in to any input handler is determined by the type of input handler assembled in the molecule. Each type of input handler can specify a particular data channel of a given memory model for how it will accept data. For example, the memory model and data handling functions of a conventional data sharing environment such as Java Shared Data Toolkit (JSDT™), of Sun Microsystems, Mountain View, CA, may be used. When the input handler is generated, the specified data channel is registered for the memory environment of the system. Data coming from a source (another molecule or a third party client in the system) on the specified data channel can then be routed to that input handler. By making a choice of an input handler for a specific data type, the need to do data type checking at run-time is avoided. Various types of input handlers are written to accommodate various file types to be handled. In the preferred structure, once an input

handler has accepted a data source and begins to process data, the data will be "typeless" until the output handler formats or types the data to interface to a given environment. The advantage of using multiple input handlers is to allow simultaneous access to data sources from multiple memory models that are to converge and be processed in the same molecule.

The interface handler 44 performs a central function for the molecule by determining when the inputs to the molecule satisfy one or more predefined conditions. By allowing the molecule to await the inputs fulfilling a predefined condition, rather than having to be directed by instructions and monitored by a centralized processing management, the molecule has the ability to wait, look, and proceed, so that it has the characteristic of acting autonomously. This is differentiated from a code instruction of conventional linear programming, which is written to execute a specific task at a specific time in a specific sequence. The predefined condition may be the presence of all of the required data inputs to the input handler. Alternatively, it may include the presence of a signal received by the signal handler. The interface handler may also be dynamically reconfigured by signals received by the signal handler, in order to change the predefined conditions to invoke a method, the method for a given set of conditions, or the output handlers to be invoked to handle the results

The interface handler can have predefined input conditions for each of a plurality of different method handlers and associated methods, depending upon which input data sources are present. The molecule has a plurality of lookup tables within the interface handler awaiting indications from one or more input handlers. An interface handler will wait until all inputs specified in the lookup table are met before facilitating communication between the input handler and the method handler.

The interface handler may use any suitable means for monitoring the multiple input conditions to determine when the requirements of a predefined condition have been met. This function may be implemented by the software equivalent of a bit array, bar table, or switch array. When all required input conditions are met, the interface handler 44 calls the respective method handler 46 and passes it the appropriate references for invoking the associated method 47. In the example shown in **FIG. 2**, the input handlers 42a and 42b communicate to the interface handler 44 that the correct pointers/data sources are queued, resulting in specified flags in the bit array being set to "1" to indicate the input condition "C", and the input handler 42c sets the flag bits for input condition "F". The fourth flag bit in the third row of the bit array is set to "1" by the presence of another input condition, for example, a signal sent to the signal handler 40. With all required flag bits set to "1" in the bit array's third row, the interface handler 44 opens a path "G" to invoke the appropriate method handler 46, which then calls the associated method 47 for processing the inputs to the output handler 48.

In **FIG. 3**, a more detailed example of the logic operation of the interface handler is depicted. The interface handler 44 stores the predefined inputs necessary to a first condition indicated

as "Lookup #1". Lookup #1 specifies what events will occur under the predetermined conditions for the interface handler 44, including "all data present", "timeout", and "signal". The interface handler 44 may have one or more lookups. In this example, Lookup #1 depicts an interface handler dependent upon the presence of three data sources, i.e., Shared Memory Data A, Hardware Memory Data B, and Network Memory Data C. In Lookup #1, index 1 is set on/off depending if Shared Memory Data A is present or not, and, similarly, index 2 for Hardware Memory Data B, and index 3 for Network Memory Data C. When all data are present, the interface handler calls "all data present" for the appropriate method handler 46a. As a default condition, the index for "TimeOut" is set to "on" to invoke method handler 46b, if a specified time interval has passed without all data input conditions being met. As another default condition, the index for "Signal" is set to "on" to invoke method handler 46c, if a specified signal is received by the signal handler 40 before all data input conditions are met. Lookup #2 is shown as dependent upon two data sources, Hardware Memory Data B and Network Memory Data C. The interface handler knows at all times which of the associated input handlers 42 have data, because when an input handler receives new data, or no longer has data, it notifies the interface handler.

Referring again to **FIG. 2**, when the predefined input conditions are met, the respective method handler is invoked (types "H", "K", and "N" for the three sets of predefined conditions), calls the associated method 47 or processing step to be performed on the input data, and provides the input data from the input handler 42 to the method 47. The input data can be handled as a block, via path "G", or as streaming data. If the input data is to be streamed, method handler sets up input paths for streaming handled by queries "I" from the method handler and input responses "J" from the input handler. For a block of data, the method 47 returns an output M to the method handler 46, which sends it on to the output handler 48 and then attends to cleaning up after itself. For streaming data, the output M is sent directly to the output handler 48 until termination of the method. Examples of a method may be computing an arithmetic sum, sorting data, performing a Boolean operation, etc., and may include an entire third party application.

FIG. 4A and **4B** show examples of the two method handling types, block method handling and streaming method handling, respectively. In block method handling, the method handler 46 has received from the interface handler 44 the references to the incoming data that it will need to call a method 47. The method handler 46 then queries the incoming data from the input handlers 42, calls the block method 47, and passes the responses to the method 47. The method handler 46 then waits for the method 47 to return its results. It then acquires output handler references from the output handlers 48, and sends the results to them. If the references do not exist, the method handler can create (invoke) the output handler. The data then is passed to the output handler 48, and the method handler 46 exits.

In streaming method handling, the method handler 46 has received from the interface handler 44 references to the incoming data that it will need to call a method 47. Output handler 48 references for outgoing data are also acquired or created by the method handler 46. When the predefined conditions are met, the streaming method 47 is called, and the input and output handler references are passed to the streaming method 47. The streaming method 47 can then receive data from the input handler 42, process it, and produce it directly to the output handler 48. Utilization of the method handler 46 in this streaming fashion allows the molecule to act as a "connection engine" which allows a logic web to string together dynamic chains of methods 47.

The output handler 48 establishes the appropriate connection to a data channel in the type of memory environment the output will be provided to, as determined by the programmer. If a data channel for output is not specified in the configuration file for the output handler, the output handler can be enabled to create a data channel at run time in response to the conditions that exist. By design, the output handler thus allows interface to any type of memory model. It queues outgoing data sources to the extent to which the queue will allow, as determined by the programmer in accordance with the type of output handler chosen.

The advantage of using output handlers is to allow simultaneous distribution to multiple memory models for input data which have converged and processed in the same molecule. There can be many types of output handlers. Particularly advantageous is the use of input and output handlers for shared-memory computer systems. Instead of moving actual data, pointers to the data or processes can be moved within a molecule or from one molecule to another, for another or the next molecule to operate on. This gives the molecule a speed advantage over conventional object-oriented programming (OOP) methods. OOP methods can pass a pointer too, but the pointer has to be processed through the OOP program to find the pointer. In contrast, a molecule can accept a raw pointer from another molecule and proceed directly with the next operation.

As determined by the programmer, each molecule is provided with the micro-components that allow it to act autonomously to process the required data by a defined method to the intended output. Data are moved to and from molecules on specified data channels in whatever data sharing environment the logic web has been contructed for. If any data are not available, the molecule will simply wait, and does not have to be instructed what to do next. Each molecule may be given a unique name, which is maintained in a name registry. Any molecule can thus signal that molecule by its unique name, and an additional molecule of the same name cannot be created in the same registry.

The processing of a molecule may be terminated at any time by sending a termination signal to the signal handler 40. At any point in their existence, a molecule can be stopped and reconfigured for a different mode or method by a reconfigure signal sent to the signal handler 40. The molecule can also be signaled to write out its current configuration (state information), so that the state

of processing can be analyzed for any point in time. In contrast, a conventional program does not maintain dynamically changing state information, and therefore cannot explain what is running at any particular time.

In the event that any or all of the five sub-components of a molecule receive a "kill" signal, and the handler types selected support retaining information as to the state of the processing, the handlers will write their state information to a molecule configuration file that can be reloaded. Thus, if the system fails for any reason, the state information can be obtained from the molecule, and used to re-create the state of each molecule and the entire logic web. This allows a system administrator to locate where or how a failure occurred, and also to load that configuration file into another molecule in order to resume processing once the failure has been corrected. The ability to rebuild the "state" of a process (logic web) is different than in other programming paradigms. In the event of system failure, typically the entire process would have to be restarted from the beginning, and stepped through one execution at a time until failure is reached. With the programming method of the invention, using the logic web, the process can be recreated just prior to the point of failure, allowing for greatly increased speed of diagnosis.

The configuration file of a molecule specifies what run-state the molecule will run as, e.g., "run once", "run until out of data", or "run until signaled to die". A molecule may clean itself up when it has completed running once, when there is no longer data in its input handler and output handler, or when the signal handler receives a signal to kill the molecule from the signal environment. This can be accomplished by erasing the software file, or disabling its executable code. The integrity of the logic web can thus be maintained by taking care of "clean up" on a per molecule basis. Incorrect data or state information are not left behind after termination where they may corrupt other handlers. Clean up at the molecular level also allows for efficient memory usage and removes the need for periodic global clean up routines. Molecules are only loaded for the required run-time, thereby freeing machine resources immediately upon termination of the molecule. For streaming methods, a molecule can be created that will use the available resources to stream data so long as provided, at which point the machine resources are freed. If the necessary life span of the molecule is to be determined dynamically, the programmer can instruct the signal handler to listen to the appropriate signal environment to receive kill signals. The kill signal can be sent to each of the sub-components of the molecule for cleaning itself up, which may involve writing out their state for future reference. The unique ability of this programming method to clean itself up with each molecule (each data processing substep) makes for efficient use of machine resources. In other programming paradigms, although some cleaning occurs during run-time, the bulk of the machine resources are not freed until the application exits.

Logic Web Programming Method

The programming method of the present invention calls for the construction of one or more logic webs to perform incremental processing steps of an overall processing task. The design of a program to accomplish a processing task through a set of incremental processing steps is a matter of skill, ingenuity and artistry in the programming field, and is beyond the scope of this description. The fundamental characteristic of the new programming method is the creation of a logic web operatively connected to a computing resource which is composed of a plurality of molecules which can perform data processing steps autonomously. The particular structure of a logic web depends upon the processing task it is required to perform (a specific example is described under "Data Synchronization Example" below). The basic element in the creation of a logic web is the creation of at least one molecule having the basic five sub-components described above.

To program molecules, a programmer is provided with a library of handler types from which to create molecules. The programmer first determines what data sources data are to be obtained from, such as stored files, Internet directory, JSDT™ space (trademark of Sun Microsystems, Mountain View, CA), IBM Tubule™ space (trademark of IBM Corp., Armonk, NY), etc., and what method is to be used to process the data. The programmer picks the appropriate input and output handler types which match the data environments where the data sources are located or will reside, and then selects the appropriate interface handler. The programmer configures the interface handler by deciding which input conditions need to be satisfied in order to call a method handler. The programmer then selects the name of the method that will be invoked, and which output handler(s) match the environment(s) the processed data will be sent to. The programmer determines whether the molecule will run once, run until out of data, or run until signaled to die. The programmer also determines whether to use a timeout period, what the timeout method is, and what method to call when a signal of a certain type is received. Finally, the programmer defines which additional or next molecules will be loaded, and which signals are to be used in order to remap this or other molecules.

This results in one molecule unit. The programmer saves it as one initialization file and gives it a unique name that is logged in a name registry. The programmer follows a similar process for each molecule to be created in the logic web. As this paradigm of programming method is further developed, standard initialization files for optimized molecules and even whole logic webs can be stored in libraries for programming use. The development of libraries or lists of handlers is expected to allow a layperson to create fully operating molecules or whole applications without writing a line of code.

The result of programming a data processing task by the programming method of the present invention is a program which executes by creating logic webs of molecules which can autonomously process data at any computing resources to which they are assigned without the need for centralized processing management. A further important feature is the ability to dynamically reconfigure one or more molecules of a logic web to change its data processing function, either as a

run-time change, or as a development or extension of the program, or as a correction of program failure.

The execution of the steps in a processing task are accomplished through the generative actions of the signal handlers. As a signal handler is generated, upon the reading of its configuration file, it forks and creates any other molecules indicated in the configuration files. The signal handler may also remap its molecule to another molecule name, or another method may be called. Thus, each layer of molecules builds the logic web "out ahead of itself". As a result, the successive layers of molecules will be generated, then will sit and wait for their input conditions of available data sources to be fulfilled.

FIGS. 5A - 5F illustrate a comparison of the logic web programming method of the invention with conventional object-oriented programming. This example uses a simplistic comparison of the data processing task of searching automobile data according to given search inputs. The data processing program is set up to perform searches of the database by one or more categories of car type, make, style, steering characteristics, handling characteristics, color, braking characteristics, and acceleration characteristics. The effects of the logic web programming method are shown in **FIGS. 5A - 5C**, and the object-oriented programming method in **FIGS. 5D - 5F**.

In **FIG. 5A**, the Car molecule is loaded. When the Car molecule is fulfilled with the search inputs A, C, D, it calls for the creation of the Handling molecule, which in turn calls for the creation of Steering molecule. The inputs are delivered to the Steering molecule, which processes them and returns a result. In the object oriented programming method, shown in **FIG. 5D**, the entire Car object is loaded, and inputs A, C, and D are passed to the Car object. The hierarchical elements, or classes, of the Car object, i.e., Car type, Handling, and Steering process the inputs, producing a result.

FIG. 5B shows how in the logic web programming paradigm, it is not necessary to quit the running operation in order to change the functional characteristics of that operation. Gas and Electric (power) molecules are added and made accessible to the operation. A remapping signal is sent to the Acceleration molecule so as to accommodate the new molecules added in the event that acceleration is called. In the object oriented programming method, as shown in **FIG. 5E**, the running of the operation must be exited, the source code acquired for the entire Car object, the Gas and Electric objects edited for their place in the hierarchy, and the Acceleration object edited to accommodate the hierarchy change. The object code must then be recompiled, in which event any other source code or object code to be used in the operation must also be edited, recompiled and re-linked.

In the last example in **FIG. 5C**, the Acceleration molecule in the logic web programming method is now running. When the Acceleration molecule is fulfilled by new input B and existing input C, the Gas molecule is called, and the data is delivered and processed, returning a result. In the object oriented example in **FIG. 5F**, the operation is restarted, and the entire Car object is reloaded. Inputs B and C are passed to Car object, and the hierarchical elements, or classes, of Car type, Handling, Acceleration, and Gas process the inputs, producing a result. This shows that in the logic web programming method a system can dynamically adapt to new parameters which are applied while the system is running. This eliminates have to re-code and compile to continue run operations.

Compiling and Run Time Operation

In the logic web programming method, an overall processing task is accomplished by designing a logic "web" of interacting molecules which can autonomously perform the required processing of data sources to a final result. The web of molecules is generated at run time from initialization files written by the programmer which are distributed among the computing resources participating in accomplishment of the overall processing task.

The initialization files generate an initial cohort of molecules to start the processing task. Utilizing the available computing resources of a local CPU and associated databases, the molecules remain in operation until they are terminated, run out of data, or are reconfigured. During operation they may be used to invoke other molecules at any time. In this manner, they act like autonomous elements in a dynamic environment. However, rather than chaotic, their operation is purposeful and in parallel everywhere at once.

This new paradigm of computer processing results in optimally efficient distributed processing, as well as the capacity to do true parallel programming and processing. It will process data from disparate sources at once, in addition to being able to make use of any machine on a network no matter what its type or operating system environment.

In a preferred implementation, the handlers are assigned unique type names and are written in object or source code. Handlers of the same type may be written for many types of programming environments expected to be encountered, so that the correct code type can be invoked to run on the intended local computing resource. For example, they may be written in the program language used for the operating system of the local computing resource they will run on, or they may be written in a portable meta language such as JAVA™ script (trademark of Sun Microsystems, Mountain View, CA) to run on an executable component (JAVA virtual machine) within that operating system environment.

The initialization files for generating the molecules of the logic web exist physically as relatively small files stored on disk or in memory or transmitted via network to the local computing resources. They are run as executable files that are read into RAM of the local computing resource and executed like other compiled programs. If the molecule is intended to be deployed in the Internet environment, the code files can be retrieved from a URL address, or obtained from a local directory, e..g., /usr/lib/java.

The initialization files operate to assemble each of the initial cohort of molecules by invoking the component handler names. When a handler name is invoked, the file containing either source or object code is retrieved from stored handler libraries (either locally or remotely), and assembled to form the molecule. This results in much more flexible run-time operation than in conventional OOP methods where entire objects are written in completely assembled blocks, or other programming languages which write source code that first have to be compiled to run.

FIGS. 6A - 6D illustrates the step by step run-time execution of three basic types of molecules, the Block Process, the Streaming Process, and Streaming a Block Process. In **FIG. 6A**, the basic parts of a molecule are shown, including signal handler 40, input handler 42, interface handler 44, method handler 46, method 47, and output handler 48. The initialization file for the molecule builds the signal handler 40 first. The signal handler 40 starts up and notifies the name registry for the overall program. If the name already exists, it will either ignore the request to create the molecule, or remap the existing molecule. Dynamic naming conventions may also be used to eliminate the need to maintain a centralized name registry. The signal handler 40 then invokes the input and output handlers 42 and 48 and the interface handler 44. At this point, there is a running molecule that is waiting for input. It should be noted that at any step the signal handler can mandate or queue a remap of all the attributes of the input and output handlers, interface handler, and method handler, or it can suspend, resume, or terminate a unit.

For a Block Process as illustrated in **FIG. 6B**, the molecule waits in Step A for the input handlers 42 to register with the interface handler 44 that they have data. When the interface handler 44 is fulfilled, it proceeds to call the method handler 46. In Step B the method handler receives the input handler references, output handler name(s), and the method name from the interface handler. The method handler queries the input handler for the input data as a block. In Step C the method handler calls the method 47 and passes the input data response, then waits for the method to return a result. When a result is returned in Step D, the output handler references are acquired by the method handler. If it does not exist, the method handler creates an output handler 48. It then proceeds to pass the result to the output handler. In Step E the output handler 48 sends the data to its output environment, and the molecule returns to waiting for the input handlers to queue more data.

For a Streaming Process as illustrated in **FIG. 6C**, the molecule waits in Step A for the input handlers to register with the interface handler that they have data. When the interface handler is fulfilled, it proceeds to call the method handler. In Step B the method handler receives the input handler references, output handler name(s), and the method name from the interface handler. The method handler, which is chosen for the Streaming Process, acquires the output handler references, or, if it does not exist, it creates an output handler. In Step C the method handler calls the method 47 and passes the input and output handler references. The method now streams across the input and output handlers until the input data queue is empty, or the output handler no longer exists. In Step D, the last method result is passed to the output handler. In Step E the output handler sends the data to its output environment, and the molecule returns to waiting for the input handlers to queue more data.

For Streaming a Block Process as illustrated in **FIG. 6D**, the molecule waits in Step A for the input handlers to register with the interface handler and, when fulfilled, proceeds to call the method handler. In Step B the method handler receives the input handler references, output handler name(s), and the method name from the interface handler, and acquires the output handler references or creates an output handler. In Step C the chosen method handler queries the input handler, calls the method with the input data, waits for the result, and passes the result to the output handler. If the input handler queue is not empty, the method handler repeats the steps until the queue is empty or the output handler no longer exists. In Step D, the last method result is passed to the output handler. In Step E the output handler sends the data to its output environment, and the molecule returns to waiting for the input handlers to queue more data.

In the operation of the logic web, a process begun on one computing machine may migrate elsewhere, continuing the operation on another available machine on the network, or on multiple machines with available resources. Only the part of the logic web that is active is essential to the operation.

Data Synchronization Example

A specific example will now be described with reference to **FIG. 7** to illustrate how the computer processing and programming methods are applied to generate a logic web that can accomplish a given processing task, e.g., synchronizing data streams from different data sources, such as is commonly used for handling multimedia presentations. The initialization files create an Open molecule 70, which reads an input configuration file, and creates new molecules, Resource Manager 71 and Attribute User Interface 72. Resource Manager 71 queries the machine for available resources and establishes how many data sources the machine can handle (three in this example) given the criteria that the Open molecule has determined from the configuration file. Resource Manager 71 creates three Data Loaders 74a, 74b, 74c, and a User Defined Data Loader 74d.

Attribute User Interface 72 allows selection of attributes from the input configuration files, as well the creation of new attributes, and selection of file types and file names. It creates new custom data loaders upon user request, and an Execution molecule 73. Execution molecule 73 distributes file name, file type and file attributes, and creates new molecules including Sync molecule 80.

Data Loaders 74a, 74b, 74c locate the data, load the data into memory if there is enough memory, or stage the data to load in sections if necessary, prepare to pass the data, and create Data User Interface molecules 75a, 75b, 75c. These allow the user to offset the time frame of particular data streams to determine when each stream should begin, to scale a particular data stream to match the length of other streams, and to determine the duration of the data stream. Data Loaders also create Data Streamer molecules 76a, 76b, 76c which produce data streams which conform to the parameters provided, and creates new molecules.

User Defined Data Loader 74d creates Compression User Interface 77, which allows the user to control the amount of data compression, and Data Compressor 78 which compresses the data as per parameters provided. Data Compressor 78 creates Date User Interface 75 and Data Streamer 76d.

Sync molecule 80 creates Sync User Interface 79 which enables adjustment of a pulse rate and provides a count of how much time has elapsed since streaming began. The user may use this time count to stop, begin again, or move forward and backward in the data stream. Sync molecule 80 produces a synchronization pulse, maintains the list of available Data Streamers to send the pulse to, according to the parameters maintained by the Sync User Interface, and creates new molecules.

Sync molecule 80 also creates Layer molecule 81 which combines a number of input data streams into the number of specified output streams, and creates new molecules including Display molecule 82 and Save molecule 83. Display molecule 82 outputs the data on the screen, and Save molecule 83 outputs the data to disk, network, tape or other media.

The interaction of the components to produce the intended result will now be described. The user selects the initialization file for the Open molecule 70, which calls for the creation of the Attribute User Interface 72. Open 70 calls the Resource Manager 71. If there is already a resource manager in use on the machine or network, then that resource manager is used to redistribute the resources, otherwise a new resource manager is created. Resource Manager 71 calculates how many data streams can be handled comfortably, and calls to create the Data Loaders 74, one for each data stream (three in this example). It does this by looking at the input data files

types, then at the generic size of a file of that type, then at the memory resources available, and determines from those parameters how many loaders it can make without crashing memory.

The user selects one name and one file type for each available Data Loader 74 in the Attribute User Interface 72. If the user has picked more file types than the available Data Loaders 74, then the Attribute User Interface 72 requests an additional Data Loader 74 from Resource Manager 71. The creation of data loaders is customizable to the extent that the user may request a file type that is not in the Attribute User Interface list. In this case, the name and path to User Defined Data Loader 74d would be given. This gives the system flexibility to accept data types which have not been predetermined. The user also determines the attributes, or parameters to be passed to execution in Attribute User Interface 72.

Execution 73 calls for creation of the Sync 80 molecule, which calls for creation of the Layer 81 molecule. The Layer 81 molecule in turn calls for creation of Display 82 and Save 83 molecules. The Sync 80 maintains a list of names it will send a synchronization pulse to, as informed by the Data Streamers.

At execution, file names, types and attributes are passed to the Data Loaders. As soon as the Data Loaders get the file names, types and attributes, they are fulfilled and will locate the data and begin to load data into memory. If there is not enough memory available, it will stage the data to load in sections. Data Loaders then call to create Data User Interface 75 and Data Streamer 76. This process occurs for each of the Data Loaders and User Defined Data Loader that are fulfilled with file names, file types, and attributes.

In the event that Resource Manager 71 detects that the resources are becoming sparse, it will pass a request to the Data Loaders to invoke data compression. The Data Loaders will then request appropriate data compression to Data Compressor 78. In the event one of the attributes calls for compressed data, the Data Loaders will send the data to Data Compressor 78. The Compression User Interface 77 may be accessed to adjust compression parameters.

The user then selects the duration and start point of each data stream in the Data User Interface 75. The user may offset or scale the time frame of particular data streams to match the length of other streams. When a Data Streamer 76 is fulfilled with the parameters sent by the Data User Interface 75, it passes its reference name to Sync 80, informing it that the data are ready to stream. The Sync 80 will then add that reference name to the list of names it will pulse to. The list is comprised of the references names of any Data Streamer in the system at the time. If there is no longer data ready to stream in a Data Streamer, the reference name will be removed from the Sync 80 list. The number of references in the list of names will also be passed to the Layer 81 component.

The Sync 80 begins to deliver its pulse to the Data Streamers 76, which deliver the data to the Layer 81 component. If the Layer 81 component does not receive all the data streams on the list, it will request that the Sync 80 slow down the pulse until all the data arrives at the Layer 81 at the same time. The Layer 81 component passes the data to the Display 82, and calls the Save 83 component. If at any point the user wishes to intervene in the data stream to pause, stop, or move back in the time count, the user may do so in the Sync User Interface 79.

There are several advantages to using the logic web programming method in the data synchronization example. Due to the dynamic configuration capability, there can be multiple users using synchronization systems on a given machine or network at the same time and each will be distributed data synchronization resources by the resource manager. The ability to perform this accomplishes automatic load balancing, which normally takes an additional load-balancing specific software application. Since the user can create any user defined data loader as needed, the system can accommodate any data format. The system dynamically pre-formats and pre-processes data. The system will also determine when compression is needed without user intervention if desired. The data compressor will automatically compress in response to fulfilled conditions. It will also reformat data to the network file type as listed in the data loader. Taking into account hardware resources, the system will adaptively deliver a variety of data sources to the display in sync interactively. The system is also distributed and platform independent. Any one of the components can reside on any machine on a LAN, WAN, or Internet network. In a parallel processing environment, multiple synchronization logic webs distributed over multiple CPUs can be coordinated to deliver parallel data streams to a common layer.

The molecule is the fundamental element of the logic web programming method. It carries with it the essential functions normally associated with larger applications, including memory management, invoking functions, and cleanup. It is small in footprint, very flexible, and dynamically reconfigurable (adaptive). It functions as a connector for local component networking, as well as remote networking over dissimilar hardware environments. The logic web programming paradigm allows truly autonomous processes with seamless compatibility between functional units, without the need for centralized processing management or highly predictive scheduling and coordination.

Industrial Applicability

With the computer processing and programming method of the invention, complex computational and data processing tasks can be performed by designing logic webs constructed of component molecules that utilize any available computing resources of any type and in any location on any network. It is thus ideally suited for performing processing tasks in distributed computing and parallel processing environments. Logic webs can be designed with molecules that operate with any type of legacy system, applications programmed in different languages, and data of any type stored in

any format, thus enabling much more efficient programming of data integration tasks, as well as the creation of entirely new types of data integration systems.

The logic web programming method can be applied to any field by adapting its components to industry specific interfaces and functionality. For example, it can be applied to data integration, modeling, simulation, data visualization, animation, rendering, and modular production applications. For current Internet-based applications, it can be tailored to current market requirements with simplicity, integrating these considerations: server-based computing model; inherent scalability; JAVA™ based core functions and extensions; Web browser technology for delivery; ability to manage terabyte data sets; ubiquitous database model with integrated functionality; customized built-in functions; distributed end user precepts and design premise; modular sequence data flows; and phased implementation approach.

The logic web programming method is particularly suited for data visualization applications, such as data center and system monitoring, visual corporate helm, decision support systems, complex relational displays, and scientific modeling visualization. Logic web software can operate as a nonlinear scene graph with parallel, decentralized, synched and tracked object-oriented sequences. The advantages of the logic web's modular approach are especially apparent when coupled with a visual interface. The molecules are single method so that the method names them. Since the connection or binding of the methods is inherent to the molecule, the logic web can be assembled quickly by human-oriented function names or visual cues with almost no regard to internal operation. Simply running the logic web will invoke self-diagnostics at any point of failure. Monitored visually in an optimization support environment, this programming method will extend the capability to "program" computers to the computer-literate layperson. An appropriate analogy would be simplifying the programming of complex data processing tasks to the level of assembling pipes and fittings by identifying the needed parts and hooking the pieces together.

It is understood that many modifications and variations may be devised given the above description of the principles of the invention. It is intended that all such modifications and variations be considered as within the spirit and scope of this invention, as it is defined in the following claims.

APPENDIX I

```
objectMolecule(String moleculeCallerName, String moleculeCalledName)
{
    /* Contributes to methods, methodTriggers, communication */
    objectNames[]                       = Object names ( session/client names )
    objectContexts[]                    = Equivalent to package ( molecule is componate to
                                          a function
                                          collection of molecule, and is of relative Context)

    /* Socketing / Sessioning */
    objectContextKey{}                  = controls scope of access to event, field, and method
                                          groups, names, locations, version labeling, establish
                                          status,

    {
        eventNeighbors{}                = List of comsuming molecules (pass
                                          Events/Data)
        eventNeighborsConsumeRoute{}    = List of which Local/Remote data sets the neighbor
                                          molecules can consume (Routing);
        eventNeighborsProduceRoute{}    = List of where then consuming routing produces the
                                          data sets local to the neighbor molecule;

        /* The objectMolecule.Deamon.thread should maintain a record of molecule channel
        IO*/
        fieldChannel{}                  = Table of which channels are active
        fieldChannelExposed{}           = Table defining channels inter/external exposablity
        fieldLocalStatic{}              = Table of static local memory routing
        fieldLocalStaticExposed{}       = Table of defining static local memory routing
                                          exposability
        fieldLocalDynamic{}             = Table of dynamic local channel routing
        fieldLocalDynamicExposed{}      = Table of defining local channel exposability
    }

    /* localRemoteSwitcher() */

    objectMethodAccessLabeling[]        = Determines Local or Remote access for methods
    objectMethod[]                      = Methods callable name (local to Java or via API)
    objectMethodMoleculeData{}          = Data from File.moleculeName that forfills
                                          Consuming/Producing Routing.
```

```
        /*methodTrigger() data , Local/Remote flags*/

        objectWait[]                            = methodTrigger() lifespan, or lifespan dependancy
        objectMethodConsumingRouting[]          = Which Channel/Local data have methodTrigger()
                                                  block on
        objectMethodConsumingFofillment[]       = Where a methodTrigger() can look for data local
                                                  and or remote and if it can switch between the two
                                                  and if so who takes priority.
        objectMethodProducingRouting[]          = Which Channel/Local space data is delivered to.

        a methodTrigger().thread is started for each objectMethod[] element. All interfacable
        molecule should have enough data to create a methodTrigger for objectDisplay().
        fieldChannel.LocalStatic.LocalDynamic{}. a localRemoteSwitcher().thread is started
        as a child to methodTrigger().thread as approriate, which is governed by
        objectMethodAccessLabeling[],objectMethodConsumingFofillment[]

}

objectDisplay()
{
        struct objectDisplayEnvi {

                displayEnviroment[]     = Reference to current interface environment, track
grounding, Screen tracking system
                displayState{}          = Switch modes Accept Events, Accept Args, Accept Pointer, .. etc
                                          Criteria relevant object/ Superscope/subscope


                displayAlphas[]         = zBuffersLayer's relative alpha (range 0.0 - 1.0) array allows
single object to multiple Depths
                                          or volume Depth assignment.
                displayZDepths[]        = zBufferLayer assignment, array allows single object to
multiple Depths
                                          or volume Depth assignment.
                displayZElements[]      = micro layering for a given zBufferLayer
                displayComposites[]     = Object accumulation ( add, subtract, dif, ..)
                displayLocations[]      = Global location when parent, relative location when child


                regPeriod[]     = Array to (x,y image) or (x,y,z volume)
                regDisplay{}    = Display scale as opposed to preObjectArray x,y,and/or z
dimensions and registration

                Array[]         = array from an ObjectDisplayEngine()
                EventArray[]    = array from an EventDisplayEngine()
                RegArray[]      = array from an RegistrationDisplayEngine()
        }
}

objectEnviroment()
{
```

Establish broadcasting pointer information channels and display through put channels.
}

-------Core Array Process Methods------

Type: Data/String

MultiArray = array with N dimensions
SinglArray = arrayX1,arrayX2,..,arrayXN
MixedArray = arrayX1,arrayX2,array with 1-N dimensions,..,arrayXN

Algorithm:(API call, Java Class/Method Call)

Encode:
-consumer(type,MixedArray,format) produce(file)
*raw to format (simple to many)
*Multipl SinglArrays to aif,wav,txt
*Multipl SinglArrays, or MixedArray to rgb,tif,jpg

Decode:
-consumer(file) produce(type,MixedArray)
*format to raw (many to simple)
*aif,wav,txt to Multipl SinglArrays
*rgb,tif,jpg to Multipl SinglArrays, or MixedArray

Cmd Line:
-pass(String cmd,String argv[]) return(pid,status)

Splitter:
-consume(type,MixedArray[N],how) produce(type,MixedArray)

Mixer:
-consume(type,MixedArray[N],how) produce(type,MixedArray)

Stream:
-consume(type,SinglArray,how,rate) produce(channel VARstream)

PacketStream:
-consume(type,MixedArray,how,rate) produce(channel VAR[]stream)

Packet:
-consume(channel VARstream,how) produce(type,SinglArray)

MultiPacket:
-consume(channel VAR[]stream,how) produce(type,MultiArray)

Formula w/ Logic controlling output:
-consume(Elm & MixedArray,Sting equation) produce(Elm & MixedArray);
*Composite Over,under,multiple,add,differance

Range:
-consume(type,MixedArray[N]) produce(type,min[X1->N],max[X1->N],avg[X1->N])

Sort:
-consume(type,SinglArray,key) produce(type,SinglArray)

Element:
-consume(type,MixedArray[N],index[]) produce(type,MixedArray)

Crop:
-consume(type,MixedArray[N],indexStart[X1->N],indexEnd[X1->N])
produce(type,arrayX1->N)

Scale:
-consume(type,MixedArray[N],type,Factor) produce(type,MixedArray[N])

Sample:
-consume(type,MixedArray,step) produce(type,MixedArray)

Flip:
-consume(type,SinglArray,xdirection,ydirection) produce(type,SinglArray)

Filter:
-consume(type,SinglArray,) produce(type,SinglArray)

Rotate:
-consume(type,SinglArray,angle,direction) produce(type,SinglArray)

Interpolate:
-consume(type,MixedArray[N],interp_type) produce(type,MixedArray[N])

Tasks:

Data manipulation Process sequencing tool

CLAIMS:

1. A computer processing method comprising the steps of:

creating a plurality of software entities ("molecules 10") each of which is configured with software micro-components including a signal handler (40), at least one input handler (42), at least one output handler (48), an interface handler (44), and at least one method handler (46) for an associated method, said input handler, output handler, and signal handler being operative for sending and receiving communication signals to or from another molecule or logic web externally of the respective molecule and being operatively connected to the other micro-components, said at least one input handler being operative for queuing input data, said interface handler being operative for determining when a predefined input condition for required input data to be received by said input handler is fulfilled and then invoking said method handler, said method handler being operative for invoking said associated method for processing the input data, and said at least one output handler being operative for outputting a result of the processing of input data by said method;

storing the created molecules in a library for later run time use;

deploying on a given computing resource a logic web comprising a plurality of molecules selected to perform a given processing task by invoking a first molecule to be retrieved from the library and executed on the given computing resource; and

said first molecule invoking one or more other molecules to incrementally extend said logic web "on the fly" by said signal handler of said first molecule sending a communication signal to invoke said other molecule(s).

2. A computer processing method according to Claim 1, wherein said creating step includes creating a plurality of logic webs each having its web of software entities configured to perform a data processing task with a computing resource autonomously, and said deploying step includes deploying the plurality of logic webs with respective ones of a plurality of computing resources, respectively.

3. A computer processing method according to Claim 2, wherein said plurality of computing resources are distributed on a network in a distributed computing environment.

4. A computer processing method according to Claim 2, wherein said plurality of computing resources include an array of central processing units (CPUs) in parallel in a parallel processing environment.

5. A computer processing method according to Claim 1, wherein the at least one logic web is comprised of a plurality of molecules, and said logic web exists as initialization files for generating an initial host of molecules which invoke and generate other molecules in successive layers of incremental processing steps.

6. A computer processing method according to Claim 1, wherein said creating step includes creating molecules having a handler function for creating next molecules in successive layers of incremental processing steps.

7. A computer processing method according to Claim 1, wherein said creating step includes creating molecules having a built-in handler function for performing a clean-up of its functions when the molecule is to be terminated.

8. A computer processing method according to Claim 1, wherein said creating step includes creating molecules having a handler type for recording information on the state of its micro-component handlers and signaling such state information externally through said signal handler.

9. A computer processing method according to Claim 1, wherein said signal handler can receive signals for and has a handler type for dynamically reconfiguring the micro-component handlers of the molecule while it is in existence to perform a processing task.

10. A computer processing method according to Claim 1, wherein said interface handler includes a handler type for providing the molecule with the characteristic of autonomously waiting, looking, and proceeding with said associated method for processing the input data by waiting until said input handler indicates that the predefined input conditions are present before invoking said method handler for the associated method.

11. A computer processing method according to Claim 1, wherein said interface handler includes a plurality of handler types for determining when respective predefined input conditions for the presence of respectively required data is fulfilled and for invoking respective ones of a plurality of method handlers and associated methods.

12. A computer processing method according to Claim 1, wherein said input handler is selected from one of a plurality of input handler types corresponding respectively to a plurality of different data source types.

13. A distributed computing method comprising the steps of:

creating a plurality of software entities ("molecules 10") each of which is configured with software micro-components for sending and receiving communication signals to or from another molecule or logic web externally of the respective molecule, said micro-components of each molecule being operatively connected to each other for processing input data in a given computing environment in which said molecule is deployed and providing a resulting output of processing the input data;

deploying the plurality of molecules each on a respective one of a plurality of computing environments; and

initializing each molecule deployed in its respective computing environment to initiate a "logic web" of molecules of data processing functions in successive layers of incremental processing steps, with a first molecule invoking one or more other molecules to incrementally extend said logic web "on the fly".

14. A distributed computing method according to Claim 13, wherein each logic web in each computing environment performs its data processing functions in its respective computing environment autonomously, and returns an output which is desired to be obtained from that computing environment.

15. A distributed computing method according to Claim 14, wherein each logic web returns the output for its respective computing environment to an external monitoring entity, and said external monitoring entity combines the outputs from the other computing environments to obtain a combined output of distributed computing.

16. A distributed computing method according to Claim 15, wherein the computing environments are a plurality of computing sites distributed on a network, and the logic webs return their outputs by sending signals on the network.

17. A distributed computing method according to Claim 15, wherein the computing environments are a plurality of computing resources in an array of processing units (CPUs) operated in parallel in a parallel processing environment.

18. A network computing method comprising the steps of:

creating a plurality of software entities ("molecules 10") each of which is configured with software micro-components for sending and receiving communication signals to or from another molecule or logic web externally of the respective molecule, said micro-components of each molecule being operatively connected to each other for processing input data in a given computing environment in which said molecule is deployed and providing a resulting output of processing the input data;

deploying the plurality of molecules each on a respective one of a plurality of computing environments which are computing sites distributed on a network;

initializing each molecule deployed in its respective computing environment to initiate a "logic web" of molecules of data processing functions in successive layers of incremental processing steps, with a first molecule invoking one or more other molecules to incrementally extend said logic web "on the fly"; and

having each logic web at each network computing site perform its data processing functions in its respective computing environment autonomously, and returns an output which is desired to be obtained from that network computing site.

19. A network computing method according to Claim 18, wherein each logic web returns the output for its respective network computing site to a network monitoring entity, and said network monitoring entity combines the outputs from the network computing sites to obtain a combined output for the network.

20. A network computing method according to Claim 19, wherein the network is a network of networks ("the Internet"), and the logic webs are deployed at websites on the Internet to compute data autonomously from the websites and return their outputs to the network monitoring entity.

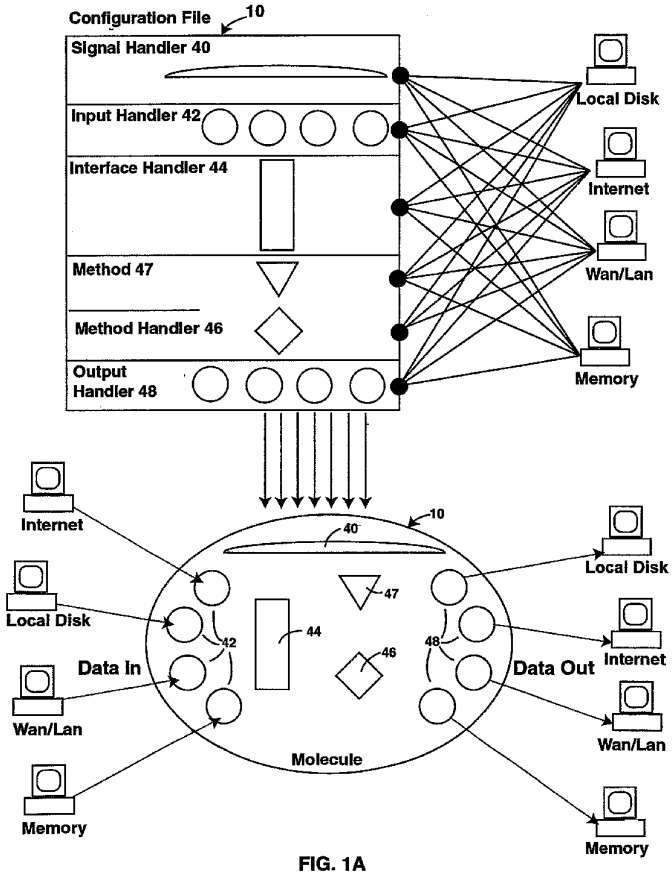

Configuration File
10
Signal Handler 40
Input Handler 42
Interface Handler 44
Method 47
Method Handler 46
Output Handler 48
Local Disk
Internet
Wan/Lan
Memory
Internet
Local Disk
Data In
Wan/Lan
Memory
10
40
47
44
46
42
48
Molecule
Local Disk
Internet
Data Out
Wan/Lan
Memory

FIG. 1A

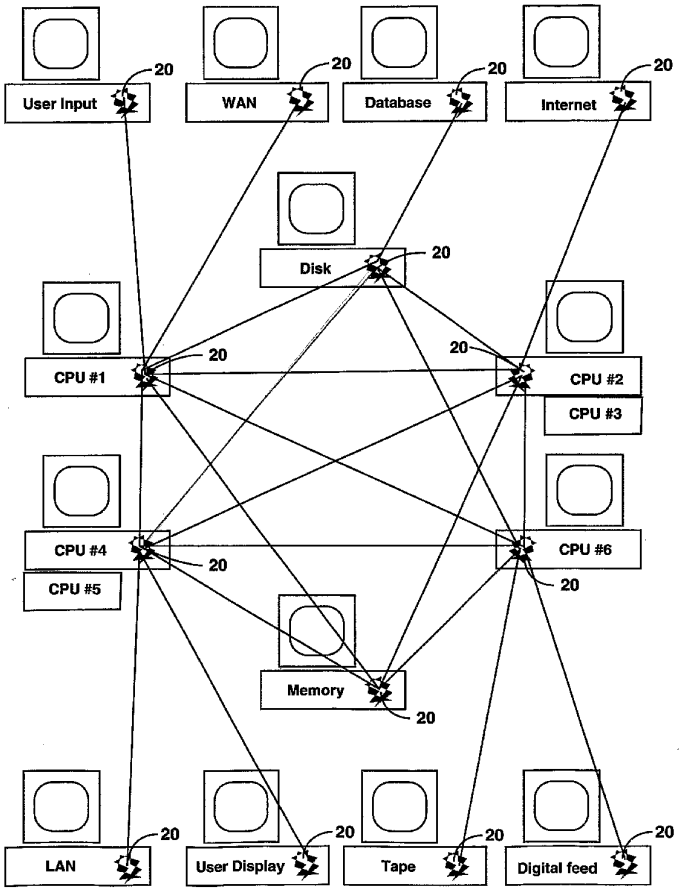

User Input
WAN
Database
Internet
Disk
CPU #1
CPU #2
CPU #3
CPU #4
CPU #5
CPU #6
Memory
LAN
User Display
Tape
Digital feed
20

FIG. 1B

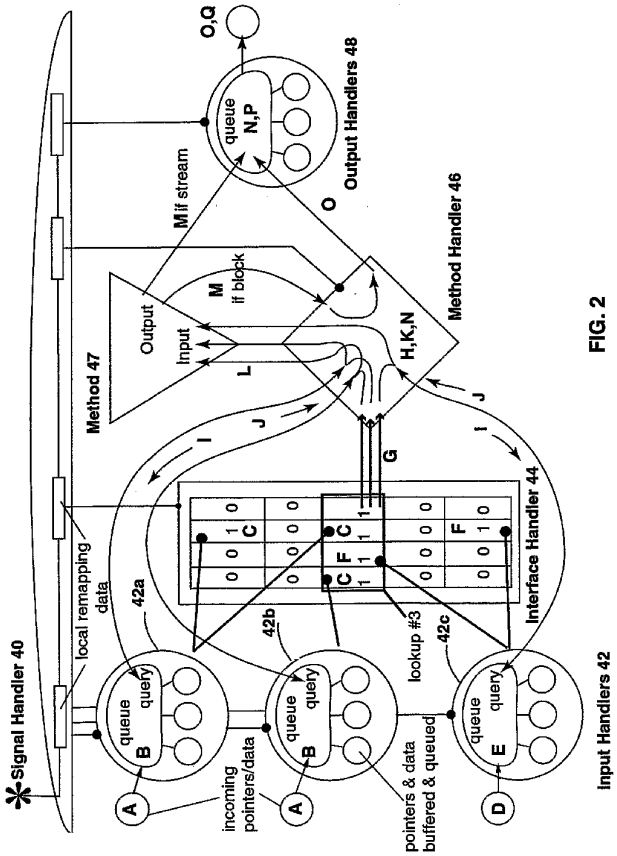

O,Q
Output Handlers 48
queue
N,P
O
M if stream
Method Handler 46
H,K,N
M
if block
Output
Input
L
Method 47
J
I
G
Interface Handler 44
C
F
local remapping data
42a
42b
42c
lookup #3
Signal Handler 40
queue
query
B
A
incoming pointers/data
pointers & data buffered & queued
E
D
Input Handlers 42

FIG. 2

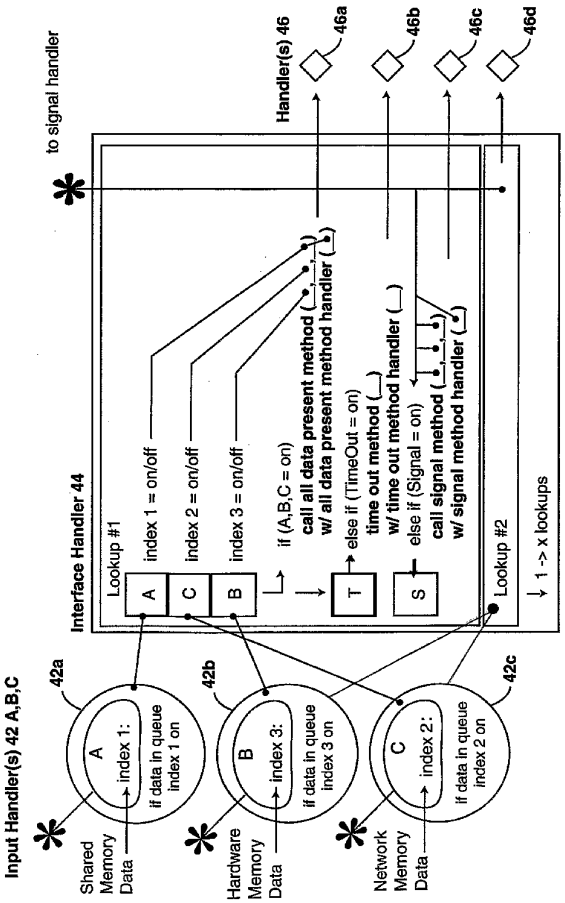

to signal handler
Handler(s) 46
46a
46b
46c
46d
Interface Handler 44
Lookup #1
index 1 = on/off
index 2 = on/off
index 3 = on/off
A
C
B
if (A,B,C = on)
call all data present method (_,_,_)
w/ all data present method handler (_)
else if (TimeOut = on)
time out method (_)
w/ time out method handler (_)
else if (Signal = on)
call signal method (_,_,_)
w/ signal method handler (_)
T
S
Lookup #2
1 -> x lookups
Input Handler(s) 42 A,B,C
42a
42b
42c
A
index 1:
if data in queue index 1 on
B
index 3:
if data in queue index 3 on
C
index 2:
if data in queue index 2 on
Shared Memory Data
Hardware Memory Data
Network Memory Data

FIG. 3

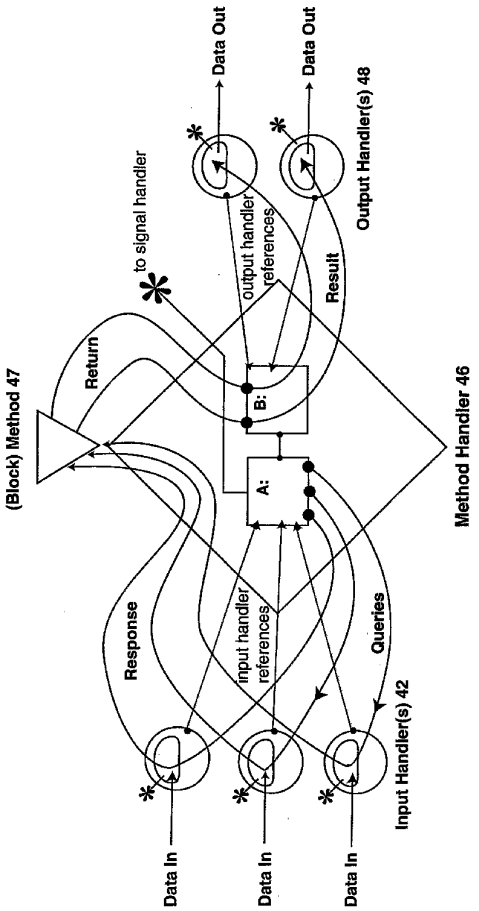

Data Out
Data Out
Output Handler(s) 48
to signal handler
output handler references
Result
Return
(Block) Method 47
B:
A:
Method Handler 46
Response
input handler references
Queries
Input Handler(s) 42
Data In
Data In
Data In

FIG. 4A

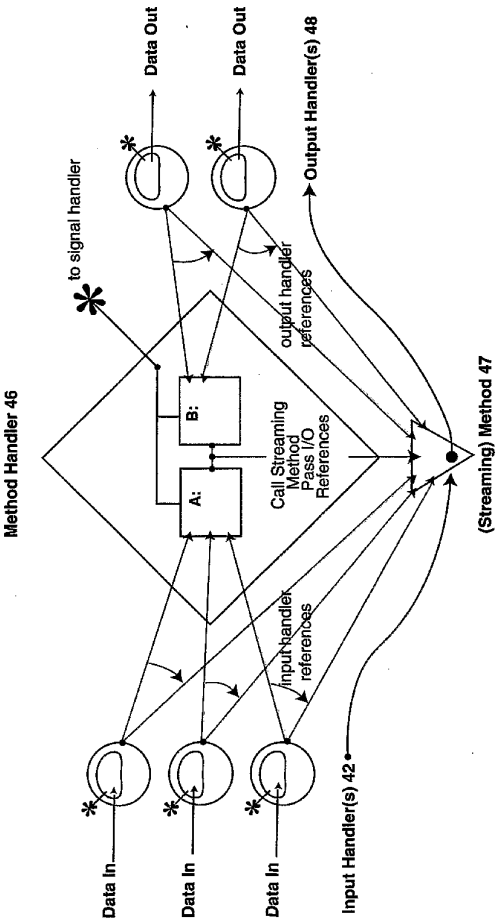

Data Out
Data Out
Output Handler(s) 48
to signal handler
output handler references
Method Handler 46
B:
A:
Call Streaming Method Pass I/O References
(Streaming) Method 47
input handler references
Data In
Data In
Data In
Input Handler(s) 42

FIG. 4B

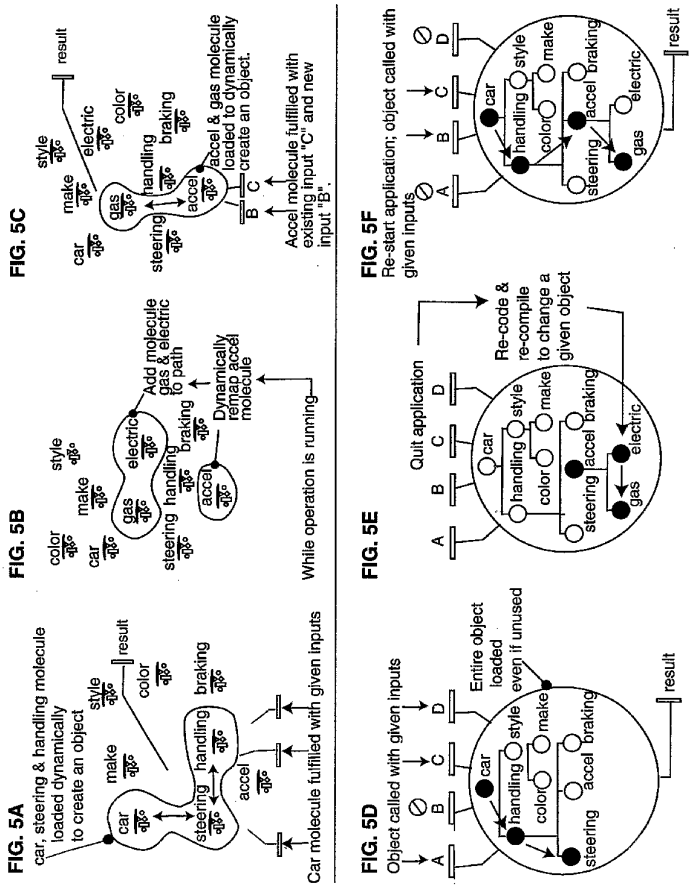


FIG. 5A
car, steering & handling molecule loaded dynamically to create an object
make
car
steering
handling
accel
braking
style
color
result
Car molecule fulfilled with given inputs
FIG. 5B
color
car
make
style
gas
electric
steering
handling
braking
accel
Add molecule gas & electric to path
Dynamically remap accel molecule
While operation is running
FIG. 5C
style
make
car
gas
electric
color
steering
handling
braking
accel
result
accel & gas molecule loaded to dynamically create an object.
Accel molecule fulfilled with existing input "C" and new input "B".
FIG. 5D
Object called with given inputs
A
B
C
D
car
handling
style
make
color
steering
accel
braking
Entire object loaded even if unused
result
FIG. 5E
Quit application
Re-code & re-compile to change a given object
FIG. 5F
Re-start application; object called with given inputs

FIG. 6A

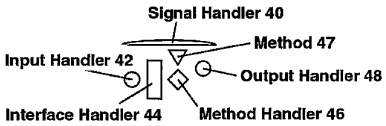


FIG. 6B

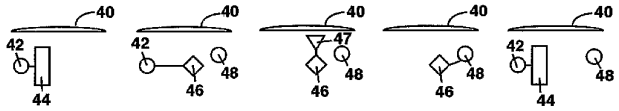


FIG. 6C

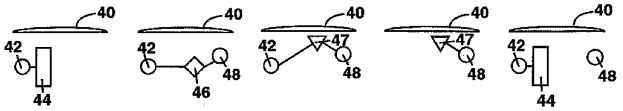
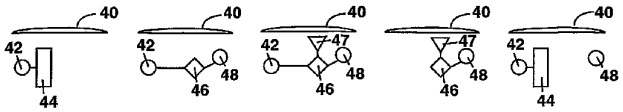


FIG. 6D



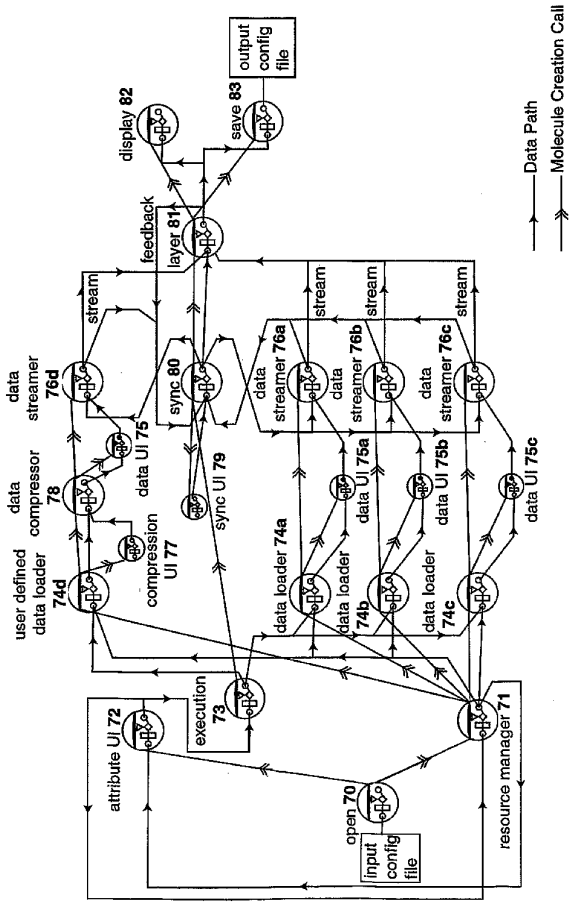

output config file
save 83
display 82
feedback
layer 81
stream
data streamer 76d
sync 80
data streamer 76a
data streamer 76b
data streamer 76c
data compressor 78
data UI 75
sync UI 79
data UI 75a
data UI 75b
data UI 75c
compression UI 77
user defined data loader 74d
data loader 74a
data loader 74b
data loader 74c
attribute UI 72
execution 73
resource manager 71
open 70
input config file
Data Path
Molecule Creation Call

FIG. 7

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/41389

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : G06F 9/44
US CL : 717/107
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 717/107, 1; 707/10, 102; 709/201; 714/38

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,701,439 A (JAMES et al) 23 December 1997 (23.12.1997), the whole document.	1-20
Y	US 5,937,189 A (BRANSON et al) 10 August 1999 (10.08.1999), the whole document.	1-20

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
01 April 2002 (01.04.2002)	09 MAY 2002
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. 703 305-3230	Authorized officer ST. JOHN COURTENAY III Telephone No. 703 305-3665

Form PCT/ISA/210 (second sheet) (July 1998)

フロントページの続き

(72)発明者　ファブレス　ウイレイ
アメリカ合衆国　ハワイ州　９６７９０　マウイ，クラ，カウエヒ　プレイス　２１４
(72)発明者　パーク　ジョア
アメリカ合衆国　ハワイ州　９６７９０　マウイ，クラ，カウエヒ　プレイス　２１４

Ｆターム(参考) 5B076 DD05
5B098 AA10 GA04 GC01 GC16 GD02

【要約の続き】

古いシステム、異なった言語にプログラムされたアプリケーション、任意のファーマットで格納された任のタイプのデータと共に動作するように割り当てることができる。その結果、分散型、すなわち、並列処理環境内でのデータ処理タスクは非常に効率的であり、全く新しいタイプのネットワークまたは並列処理タスクが行われる。