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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
08 February 2018 (08.02.2018)

WIPO | PCT

(10) International Publication Number
WO 2018/024348 A1

(51) International Patent Classification:
G06F 9/52 (2006.01)

(21) International Application Number:
PCT/EP2016/077180

(22) International Filing Date:
09 November 2016 (09.11.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/370,961 04 August 2016 (04.08.2016) US

(71) Applicant: **TELEFONAKTIEBOLAGET LM ERICSSON (PUBL)** [SE/SE]; SE-164 83 Stockholm (SE).

(72) Inventors: **FALLON, Liam**; Ericsson Software Campus, Athlone, Westmeath (IE). **KEENEY, John**; Ericsson Software Campus, Athlone, Westmeath (IE). **VAN DER MEER, Sven**; Ericsson Software Campus, Athlone, Westmeath (IE).

(74) Agent: **ERICSSON**; Torshamnsgatan 21-23, 164 80 Stockholm (SE).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report (Art. 21(3))*

(54) Title: METHOD AND APPARATUS FOR DISTRIBUTED NETWORK MANAGEMENT

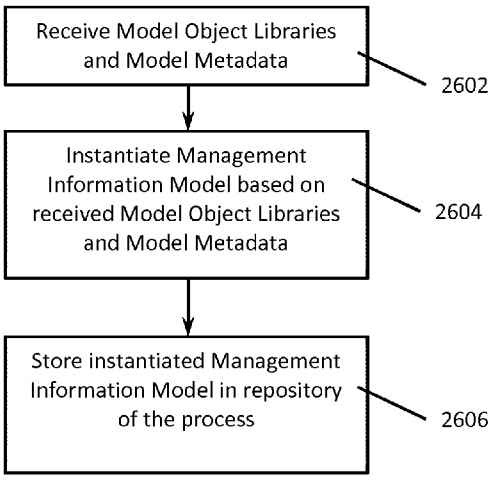

Fig. 26

(57) Abstract: Method and Apparatus for Distributed Network Management
A method of managing a network. The network comprises a plurality of distributed hosts, wherein at least one process is being run on a host. Said process provides a function related to operation of the network by executing at least one application instance, wherein the at least one application instance requires for execution at least one Management Information Model. The method comprises receiving (2602) Model Object Libraries representing the at least one Management Information Model and Model Metadata associated with the Model Object Libraries and instantiating (2604) the at least one Management Information Model based on the received Model Object Libraries and Model Metadata. The method further comprises storing (2606) the instantiated Management Information Model in a repository of the process for use by the at least one application instance. Also disclosed is an apparatus configured to carry the above method.

WO 2018/024348 A1

Method and Apparatus for Distributed Network Management

Technical Field

The present invention relates to network management, in general, and in particular to network management based on distributed Management Information Model.

Background

As Telecommunication networks are becoming increasingly distributed and virtualized due to the advent of technologies such as Software Defined Networking (SDN) and Network Function Virtualization (NFV), network management applications must evolve to manage such distributed virtualized networks. Although specifications for managing NFV such as the NFV-MANO specification [ETSI, 2014] do not specify a distributed architecture for management per se, using a distributed approach in the design of a management system is an obvious way of leveraging the power of today's cloud infrastructure with its underlying multi-core and memory rich hosts. It is not surprising, therefore, that modern management systems such as Ericsson Network Manager, ENM, [Ericsson, 2016] are inherently distributed.

Network management has long used information models to describe information that is being managed in network elements and management systems [ITU-T, 2005b], [3GPP, 2011a], [IETF, 1991]. Such models provide many advantages: the modelled information is unambiguously described, the model acts as an interface to the managed domain, the protocol for accessing the model is available, and any application can use and manipulate the information in the model once it complies with the management protocol, and usage of the model can be logged. The paradigm of a hierarchy of models is ubiquitous in network management [ITU-T, 2000b], with managers in a given level of management using models published by agents in the level below.

A number of interesting technologies have emerged in the domain of computer science that can be used to share information between processes in a distributed system. Indeed, Memory Driven Computing [Bresniker et al., 2015] has been proposed as a way of harnessing the large banks of cheap memory on the hosts of modern distributed systems. Implementations of Distributed Hash Tables [Coulouris et al., 2012] that allow exchange of unstructured maps of information between processes have been available for some time [Infinispan, 2016], [Hazelcast, 2016]. A map is an object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value. In parallel, frameworks have emerged that support distributed locking across distributed processes. Some frameworks take a centralized approach, where a central server holds a record of locks [Curator, 2015] while others take a fully distributed approach, with lock information being exchanged between processes at run time [Hazelcast, 2016]. In addition, distributed technologies for monitoring such as syslog [IETF, 2009] and Log4j [Log4j, 2015] and persistence such as relational databases and distributed file systems are mature and are very suitable for application to distributed model management.

There is no existing method for defining, scoping, and using a distributed information model, where the model itself has a scope wider than a single network element, NE. Although hierarchies of models are very common, each model is a separate information model instance with responsibility for that model instance resting with the NE in which the information model resides. The problem of information model coordination is magnified because a different approach to model coordination is taken in each domain. For example, a given cell *C1* is modelled in a base station model as cell *C1* and is modelled separately in the management system as cell *C1*. The models are different, the cell is modelled twice, and the management system and base station NEs must interact with each other to keep those models consistent, in this case using 3GPP Integration Reference Points, IRPs, [3GPP, 2011a]. Likewise, an interface is modelled in a computer using MIB-II [IETF, 1991] as *eth0* and in an element manager separately as

eth0. Again, the element manager and computer must interact with each other to keep their models consistent, in this case using Simple Network Management Protocol, SNMP.

Information model coordination has been a challenge in management systems for very many years, but the advent of Software Defined Networking exasperates the problem. All current solutions rely on a stable hierarchy being present, with NEs occasionally entering and leaving the hierarchy in a controlled manner. When resources are managed across transient (and often virtual) NEs, where NEs, element managers and even network managers transient, spun up and torn down as required, a hierarchical approach to model coordination makes maintaining a consistent distributed view of those models a very difficult task. The approach of having independent static hierarchical models envisaged for the telecommunication networks of the 1980s [ITU-T, 2000b] does not scale well in today's dynamic networks because of the overhead and latency introduced by model coordination.

The lack of a common distributed management of information models is a serious drawback for distributed management application development. Not only does each management application have to manage its own distributed information, there is no coordination when different distributed management applications share management information.

Although Distributed Shared Memory [Protic et al., 1996], Memory Driven Computing [Bresniker et al., 2015] and distributed cache frameworks [Infinispan, 2016], [Hazelcast, 2016] allow memory to be shared between systems, the data that is shared is unstructured and uncontrolled, usually as distributed maps. Such frameworks are too unstructured and uncoordinated to be used safely by disparate management applications. Each process has the freedom to add, modify, or remove entire distributed maps or instances in the distributed maps. For example, process P1 can add the instance named cell *C1* as an object called *Cell* with parameters representing the various attributes that a cell might have. Process P2 can then simply overwrite the instance named cell *C1* with a

string value or even delete the instance. Even more concerning, a process can place a completely incompatible *Customer* object as a value on the *C1* instance on the map called *CellMap*, turning cell *C1* into customer *C1*.

Data sharing using databases is common, but a centralised database used by multiple distributed applications can introduce significant latency. There are numerous approaches to use distributed databases to support distributed data access. For example, CouchBase (http://www.couchbase.com) is a distributed NoSQL database, but like most other NoSQL databases it does not use schemas to enforce a particular data model. Many traditional relational databases support replication (for example MySQL) using either master-master or master-slave replication strategies. However, most such database replication strategies are only loosely consistent, i.e. lazy and asynchronous, violating ACID (i.e. Atomicity, Consistency, Isolation, Durability) properties, or where eager replication is used then updates introduce high latency.

The ability to lock a particular element in a model so that a management application knows that it can perform a safe read or write is a fundamental requirement for any distributed management model. Distributed locking mechanisms [Curator, 2015], [Hazelcast, 2016] support distributed synchronized locking of named distributed locks, but they do not provide a mechanism to associate or bind a named lock to a particular distributed information model instance. Controlled distributed locking of particular named instances across distributed processes is not supported.

A straightforward locking approach is preferred to a transactional approach because transaction frameworks such as those that implement the Java Transaction API (JTA) [Oracle, 1999] [Little et al., 2016] introduce a high degree of complexity and coordination that cannot be hidden from management applications. Explicit support for starting, joining, committing, and aborting transactions must be provided. Further, transactional approaches do not scale well [Cecchet et al., 2002], [Femminella et al.,

2011] as the number of application instances increase the speed of transaction execution diminishes rapidly.

The lack of common distributed models also means that common monitoring of operations on common items in shared models is difficult. If a manager requests a NE to change an attribute on cell *C1* or interface *eth0*, then both the manager and the NE must log the change using their representation of *C1* and *eth0* using their different logging mechanisms. A further system is required to aggregate and correlate the separate logs together to provide a complete view of when a model instance was initiated, written, read, and deleted.

Similarly, such a lack of a common view means that persistence of models must also be done separately, with each separate model being saved and managed in the persistence system separately. The separate models must then be mapped and joined to provide a common view.

Existing approaches have the following drawbacks.

- No existing method for defining, scoping, and using an information model with a scope wider than a single NE, concepts are re-modelled in hierarchical information models for various NE types and various management systems.
- Model coordination is required across the different information model instances, which is difficult especially in dynamic soft network management systems.
- Memory Driven Computing [Bresniker et al., 2015] and distribution frameworks [Infinispan, 2016][Hazelcast, 2016] allow memory to be shared between systems but the sharing is unstructured and unsafe.
- Integrated model aware distributed locking across models in different NEs is not possible.

- Integrated model aware distributed monitoring is not possible. Integrated model aware distributed persistence is not possible.

Summary

It is the object of the present invention to obviate at least some of the above disadvantages and provide an improved management of Management Information Model for use in network management.

Accordingly, the invention seeks to preferably mitigate, alleviate or eliminate one or more of the disadvantages mentioned above singly or in any combination.

According to a first aspect of the present invention there is provided a method of managing a network. The network comprises a plurality of distributed hosts, wherein at least one process is being run on a host. Said process provides a function related to operation of the network by executing at least one application instance, wherein the at least one application instance requires for execution at least one Management Information Model. The method comprises receiving Model Object Libraries representing the at least one Management Information Model and Model Metadata associated with the Model Object Libraries and then instantiating the at least one Management Information Model based on the received Model Object Libraries and Model Metadata. The method also comprises storing the instantiated Management Information Model in a repository of the process for use by the at least one application instance.

According to a second aspect of the present invention there is provided an apparatus for managing a network. The network comprises a plurality of distributed hosts, wherein at least one process is being run on a host. Said process provides a function related to operation of the network by executing at least one application instance, wherein the at least one application instance requires for execution at least one Management Information Model. The apparatus comprises a processor and a memory. The memory contains instructions executable by the processor such that the apparatus is operative to

receive Model Object Libraries representing the at least one Management Information Model and Model Metadata associated with the Model Object Libraries. The apparatus is also operative to instantiate the at least one Management Information Model based on the received Model Object Libraries and Model Metadata; and to store the instantiated Management Information Model in a repository of the process for use by the at least one application instance.

According to a third aspect of the present invention there is provided an apparatus for managing a network. The network comprises a plurality of distributed hosts, wherein at least one process is being run on a host. Said process provides a function related to operation of the network by executing at least one application instance. The at least one application instance requires for execution at least one Management Information Model. The apparatus comprises a receiver for receiving Model Object Libraries representing the at least one Management Information Model and Model Metadata associated with the Model Object Libraries and a creator for instantiating the at least one Management Information Model based on the received Model Object Libraries and Model Metadata. The apparatus also comprises a memory for storing the instantiated Management Information Model in a repository of the process for use by the at least one application instance.

Further features of the present invention are as claimed in the dependent claims.

Brief description of the drawings

The present invention will be understood and appreciated more fully from the following detailed description taken in conjunction with the drawings in which:

FIG. 1 is a diagram illustrating a distributed Management Information Model management in one embodiment of the present invention;

FIG. 2 is a diagram illustrating further details of the distributed Management Information Model management in one embodiment of the present invention;

FIG. 3 illustrates metadata and Model Objects for Distributed Management Information Model management in one embodiment of the present invention;

FIG. 4 illustrates an example of a distributed Management Information Model in use;

FIG. 5 illustrates an example of a process for creating metadata and Model Object;

FIG. 6 illustrates an example of a process for creating metadata for a MIM Model Map;

FIG. 7 illustrates an example of a process for creating metadata for MIM Model Usage by applications;

FIG. 8 illustrates an example of a process used by a Distributed Model Deployer for deploying and updating metadata and Model Object Libraries;

FIG. 9A and FIG. 9B illustrate examples of locking and unlocking sub-processes;

FIG. 10 illustrates an example of a process of initializing local copies of distributed MIM Model Maps;

FIG. 11 illustrates an example of a process of updating local copies of distributed MIM Model Maps;

FIG. 12 illustrates an example of a process of clearing a MIM Model Map;

FIG. 13 illustrates an example of a process of initializing a Distributor component;

FIG. 14 illustrates an example of a process of setting up and initializing a MIM Model Instance Object and its value;

FIG. 15 is a diagram illustrating an example of a distribution mechanism;

FIG. 16 illustrates an example of a process of reading a MIM instance from a MIM Model Map;

FIG. 17 illustrates an example of a process of writing a MIM instance to a MIM Model Map;

FIG. 18 is a diagram illustrating an example of a locking mechanism;

FIG. 19 illustrates an example of a process of acquiring and releasing read or write lock;

FIG. 20 is a diagram illustrating an example of a monitoring mechanism;

FIG. 21 is a diagram illustrating an example of a persistence mechanism;

FIG. 22 illustrates an example of a process of persisting MIM Model Maps;

FIG. 23 is a diagram illustrating a policy engine running instances of adaptive policies;

FIG. 24 is a diagram illustrating how MIM Model Maps manage context in the policy engine of FIG. 23;

FIG. 25 is a diagram illustrating an embodiment of an apparatus for managing a network;

FIG. 26 – FIG. 28 are flowcharts illustrating a method of managing a network in embodiments of the present invention;

FIG. 29 is a diagram illustrating an embodiment of an apparatus for managing a network.

Detailed description

In the following description, for purposes of explanation and not limitation, specific details are set forth such as particular architectures, interfaces, techniques, etc. in order to provide a thorough understanding of the invention. However, it will be apparent to those skilled in the art that the invention may be practiced in other embodiments that depart from these specific details. In other instances, detailed descriptions of well-known devices, circuits, and methods are omitted so as not to obscure the description of the invention with unnecessary details.

Reference throughout the specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with an embodiment is included in at least one embodiment of the present invention. Thus, the appearance of the phrases "in one embodiment" or "in an embodiment" in various places throughout the specification are not necessarily all referring to the same embodiment. Further, the particular features, structures or characteristics may be combined in any suitable manner in one or more embodiments.

This document presents Distributed Management Information Model Management, an overview of which is shown in Figure 1. The distributed Management Information Models (MIMs) are authored by an Information Model Authoring component, 102, which stores Model Metadata, 108, describing the MIMs in a Model Knowledge Base, 104. The Information Model Authoring component, 102, also creates a Model Object for each concept in each MIM, which may be instantiated at run time and stores it in a Model Object Repository, 106.

Once the Model Metadata, 108, is stored, a Distributed Model Deployer, 110, distributes the Model Metadata and Model Object Libraries to processes, 112 - 116, on real or virtual hosts. These processes, 112 - 116, may be in any geographic location. The Deployer, 110, ensures that only the required set of models and Object Libraries are deployed to a given process, ensures that the model metadata and objects on each host is

consistent, upgrades the metadata and objects on processes as the metadata changes, and removes metadata and objects when they are no longer required on a process.

The Model Manager, 118 – 122, on a process, 112 - 116, uses the Model Metadata to build local copies of the MIMs required by the applications running on that process, instantiating and initialising the model instances on each MIM using the Model Object Library, 124 – 128. The Model Manager controls reading and writing of instances on MIMs by applications, 130 – 134, on the process, ensuring that updates of MIM instances are transferred to other processes using a distribution mechanism. It also ensures that applications can only access instances on MIMs that are specified in the model metadata. Applications may not add or remove instances to or from a MIM, and applications can only access instances that have been defined as accessible to those applications.

If an application requires a read lock or write lock on an instance in a MIM, the Model Manager, 118 – 122, uses a locking mechanism to ensure that the model instance is locked or unlocked over all processes that are using the distributed MIM that contains that instance.

The Model Manager, 118 – 122, in each process, 112 - 116, has a copy of the MIM metadata. Each Model Manager can monitor model usage information such as initialization, reads, writes, locks, and unlocks on model instances and send that information to a common Model Usage Collector, 136, for storage in the Model Knowledge base, 104, 138. The monitored information gives the usage of every instance in every MIM on all NEs, allowing the consistency of each MIM individual to be verified and its consistency to be checked. Consider the case where information for cell *C1* is changed by a coverage optimization application and an energy saving application. Changes to the *PowerLevel* property of cell *C1* by all instances of both applications can be easily monitored. In this case, any overlap or interference between applications that read or write the properties from the same instance can be easily detected. All accesses to instances (and their properties) can then be audited. In addition, because the usage

information of all MIM instances is available, machine learning and semantic techniques can be applied to the usage information to identify less obvious conflicts and side effects.

The MIM metadata enables consistent management of persistence of information in MIMs. MIM metadata can describe how the information can be persisted. For example, the MIM metadata can be used to generate tables in a database management system. Persistence can then be enabled on one or more processes, ensuring that the data in the MIMs is always saved. MIM metadata can also assist in defining how the instance state can be serialised, a key requirement for synchronising state across different model object libraries (some fields might be transient or constant). MIM metadata can also be used to assist in defining locking strategies for MIM instances or their subordinate parts. For example, a property can be defined as "read-only", thus supporting relaxed locking, or "write-seldom-read-often" thus supporting optimistic locking strategies.

The solution disclosed herein describes Distributed Management Information Model Management and has the following advantages:

- The method manages the distribution of formally modelled management information in a way that allows distributed management applications to use that context in a safe, controlled, and monitored manner.

- Distributed applications using this method have their information models inherently defined, distributed and controlled.

- The method formalizes the modelling of information as metadata and distributes the information models across multiple applications, with the required set of models being distributed to the appropriate processes.

- The applications have a unified view of the modelled information, and use the information as constrained and controlled by a distributor on each process.

- The distributor enforces controlled reading, writing, locking, and monitoring of information in the distributed management information models.

- The approach prevents the information in MIMs from becoming inconsistent because applications work towards common distributed synchronized MIMs.

- Applications cannot corrupt the structure of information in the MIMs because the definition of the MIMs and their instances is controlled by metadata supplied by the Distributed Model Deployer.

- Distributed Management Application design is more straightforward because the requirement to coordinate shared information across application instances is delegated to Distributed Management Information Model Management.

- Each operation on a MIM instance by every process that uses the MIM instance is logged, allowing the consistency of MIM instance usage to be verified and conflicts and side effects to be identified.

In the following description, for purposes of explanation and not limitation, specific details are set forth such as particular architectures, interfaces, techniques, etc. in order to provide a thorough understanding of the described solution. However, it will be apparent to those skilled in the art that the described solution may be practiced in other embodiments that depart from these specific details. In other instances, detailed descriptions of well-known devices, circuits, and methods are omitted so as not to obscure the description of the solution with unnecessary details.

Reference throughout the specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with an embodiment is included in at least one embodiment of the described solution. Thus, the appearance of the phrases "in one embodiment" or "in an embodiment" in various places throughout the specification are not necessarily all referring to the same

embodiment. Further, the particular features, structures or characteristics may be combined in any suitable manner in one or more embodiments.

With the advent of cloud computing, virtualization, and hosts with multiple cores and hundreds of gigabytes of memory, it is very common for management applications to be distributed on multiple processes running on distributed real and virtual hosts. The use of structured and controlled information models has a very long history in network management and the value of having such structured models to constrain the managed information is widely recognized. However, current management information models are defined with the scope of a single Network Element (NE) or host with co-ordination of models across multiple hosts being left as a task for each management application. The lack of common distributed management of information models is a serious drawback for distributed management application development because, not only does each management application have to manage its own distributed information, there is no coordination when different distributed management applications share management information.

We present a method and apparatus that manages the distribution of formally modelled management information in a way that allows distributed management applications to use that context in a safe, controlled, and monitored manner. The method formalizes the modelling of information as metadata and distributes the information models across multiple applications, with the required set of models being distributed to the appropriate processes. The applications then have a unified view of the modelled information, and use the information as constrained and controlled by a Model Manager on each process. The Model Manager enforces controlled reading, writing, locking, and monitoring of information in the distributed management information models. Distributed applications using this method have their information models inherently defined, distributed and controlled.

Figure 2 is a detailed view of Distributed Management Information Model Management. In Distributed MIM Management, all distributed MIMs are defined in Metadata by a Model Authoring component, 102. The Metadata is stored in the Model Knowledge base, 104. The Distributed Model Deployer, 110, distributes the MIMs to a Model Manager, 118 – 122, in each process, 112 – 116, that is using Distributed MIM management, where each process may host one or more instances, 202, of one or more applications 130. The Model Manager, 118 – 122, manages its local copies of MIMs held in Model Maps, 204, provides access to the MIMs for applications, 130, distributes the MIM contents to other processes, as well as managing locking, monitoring, and persisting of MIM Model Instances. A Model Map, 204, is a collection of named MIM instances that have been defined to have a particular common scope, and provides a consistent interface to access named MIM instances. The references to an operation of persisting described in embodiments of the solution disclosed in this document refer to storing.

With reference to Figure 26 an embodiment of a method of managing a network is now to be described. The network comprises a plurality of distributed hosts, wherein at least one process, 112, is being run on a host. In one embodiment, the hosts are distributed in various geographical locations. Said process, 112, provides a function related to operation of the network by executing at least one application instance, 202. The at least one application instance, 202, requires for execution at least one Management Information Model. In this embodiment the method comprises: receiving, 2602, Model Object Libraries representing the at least one Management Information Model and Model Metadata associated with the Model Object Libraries and instantiating, 2604, the at least one Management Information Model based on the received Model Object Libraries and Model Metadata. The method also comprises storing, 2606 the instantiated Management Information Model in a repository of the process for use by the at least one application instance, 202. Preferably said at least one Management Information Model is used by other instances of said application as well as by instances of other applications.

In a preferred embodiment the operations of receiving, instantiating and storing are performed by the distributor, 206, shown in Figure 2 as part of a process, 112, operating on a host.

Further, in a preferred embodiment, illustrated in Figure 27, the method comprises receiving, 2702, from a first application instance, 202, a request for a write access to the Management Information Model stored in the repository. In response to this write request the method comprises locking, 2708, said Management Information Model if the Management Information Model is available for modification, 2704 - *no*. In a locked state other application instances, 130, in this host as well as applications instances, 132 and 134, in other hosts are not allowed to read said locked Management Information Model and are not allowed to write in said locked Management Information Model. The locking operation, 2708, is carried out by a Locker, 208, illustrated in Figure 2 as part of a process, 112, operating on a host. If the MIM is in a locked state, 2704 – *yes*, the method goes to "wait" state, 2706, and waits until the MIM is unlocked. In a preferred embodiment the distributor, 206, carries out the checking of the state of the MIM. The method further comprises modifying, 2710, said Management Information Model, in a write operation, based on the request received from the first application instance, 202, and storing, 2712, the modified Management Information Model. These operations are preferably carried out by the persistor, 210, illustrated in Figure 2 as part of a process, 112, operating on a host. Alternatively, the operation of modifying, 2710, may be carried out by the distributor, 206. The method further comprises unlocking, 2714, said Management Information Model to make said Management Information Model available for reading and writing operations for applications, 130 – 134, in the or another host.

Preferably the checking if the Management Information Model is available for modification comprises checking if a locked state is imposed on said Management Information Model by any process running on any host in the network and wherein said Management Information Model is available for modification only if it is not in a locked state.

In another embodiment, illustrated in Figure 28, the method further comprises monitoring, 2802, operations of: instantiating a MIM, reading a MIM, writing to a MIM, locking and unlocking a MIM; and sending, 2804, usage logs entries containing monitoring data to a Model Usage Collector, 136, which centrally records, 138, MIM usage data for the network. The operations of monitoring and sending usage logs are preferably carried out by a monitor, 212, illustrated in Figure 2 as part of a process, 112, operating on a host.

In one embodiment the method comprises imposing a write lock state on a Management Information Model if said Management Information Model contains data from a system external to the network. This is a special case in which the data from external sources is protected against modification by the applications running on the hosts, but which data can be read by these applications. It is important that, for example, weather data received from external supplier are not modified, but it is important that these data can be read so that, for example, adequate radio transmission parameters are set.

With reference to Figure 25 an embodiment of an apparatus, 2500, for managing a network is now to be described. The network comprising a plurality of distributed hosts, wherein at least one process is being run on a host. In one embodiment the hosts are distributed in various geographical locations. The process provides a function related to operation of the network by executing at least one application instance. The at least one application instance requires for execution at least one Management Information Model. The apparatus, 2500, comprises a processor, 2502, and a memory, 2504. The memory, 2504, contains instructions executable by the processor, 2502, such that the apparatus, 2500, is operative to receive Model Object Libraries representing the at least one Management Information Model and Model Metadata associated with the Model Object Libraries and instantiate the at least one Management Information Model based on the received Model Object Libraries and Model Metadata. The apparatus, 2500, is also operative to store the instantiated Management Information Model in a repository of the

process for use by the at least one application instance. In a preferred embodiment said at least one Management Information Model is used by other instances of said application as well as by instances of other applications.

In a preferred embodiment the apparatus, 2500, comprises further an interface 2506 for communication with other elements of the system.

The apparatus, 2500, is further operative to receive from a first application instance a request for a write access to the Management Information Model stored in the repository. If the Management Information Model is available for modification the apparatus is further operative to lock said Management Information Model so that the first application instance requesting the write access can modify the model. If the MIM is in a locked state other application instances, in this or another host, are not allowed to read said locked Management Information Model and are not allowed to write in said locked Management Information Model. The apparatus is further operative to modify said Management Information Model, in a write operation, based on the request received from the first application instance and to store the modified Management Information Model. Further the apparatus, 2500, is operative to unlock said Management Information Model to make said Management Information Model available for reading and writing operations for applications in the or another host.

In one embodiment, in checking if the management Information Model is available for modification the apparatus, 2500, is operative to check if a locked state is imposed on said Management Information Model by any process running on any host in the network. Said Management Information Model is available for modification only if it is not in a locked state.

In yet another embodiment the apparatus according to any one of embodiments is operative to monitor operations of: instantiating a MIM, reading a MIM, writing to a MIM, locking and unlocking a MIM; and send usage logs entries containing monitoring data to a Model Usage Collector which centrally records MIM usage data for the network.

The apparatus, 2500, in one embodiment is operative to impose a write lock state on a Management Information Model if said Management Information Model contains data from a system external to the network. This is a special case in which the data from external sources is protected against modification by the applications running on the hosts, but which data can be read by these applications. It is important that, for example, weather data received from external supplier are not modified, but it is important that these data can be read so that, for example, adequate radio transmission parameters are set.

With reference to Figure 29 another embodiment of an apparatus, 2900, for managing a network is now to be described. The network comprises a plurality of distributed hosts. In one embodiment the hosts are distributed in various geographical locations. At least one process is being run on a host, said process providing a function related to operation of the network by executing at least one application instance. The at least one application instance requires for execution at least one Management Information Model. The apparatus, 2900, comprises a receiver, 2902, for receiving Model Object Libraries representing the at least one Management Information Model and Model Metadata associated with the Model Object Libraries and also comprises a creator, 2904, for instantiating the at least one Management Information Model based on the received Model Object Libraries and Model Metadata. The apparatus further comprises a memory, 2906, for storing the instantiated Management Information Model in a repository of the process for use by the at least one application instance.

In a preferred embodiment at least one Management Information Model is used by other instances of said application as well as by instances of other applications.

Preferably, the receiver, 2902, is further operative to receive from a first application instance a request for a write access to the Management Information Model stored in the repository. The apparatus also comprises a locker, 2908, for locking and unlocking said Management Information Model In a locked state other application

instances in the or another host are not allowed to read said locked Management Information Model and are not allowed to write in said locked Management Information Model. Only unlocked Management Information Model is available for reading and writing operations for applications in the or another host. The creator, 2904, is further operative to modify said Management Information Model, in a write operation, based on the request received from the first application instance. The memory, 2906, is operative to store the modified Management Information Model.

Preferably, in checking if the Management Information Model is available for modification the apparatus, 2900, is operative to check if a locked state is imposed on said Management Information Model by any process running on any host in the network and wherein said Management Information Model is available for modification only if it is not in a locked state.

In a preferred embodiment the apparatus, 2900, also comprises a monitor, 2910, for monitoring operations of: instantiating a Management Information Model, reading a Management Information Model, writing to a Management Information Model, locking and unlocking a Management Information Model. Additionally, in this embodiment the apparatus, 2900, comprises a reporter, 2912, for sending usage logs entries containing monitoring data to a Model Usage Collector which centrally records Management Information Model usage data for the network.

In a preferred embodiment the apparatus, 2900, is operative to impose a write lock state on a Management Information Model if said Management Information Model contains data from a system external to the network.

In a preferred embodiment once model object libraries are in a process, 112, and the distributor, 206, has the associated metadata it creates a MIM and stores it. The application instances, 130 – 134, then may use the models. When a first application instance, 202, wants to write to the model (modify the MIM) then the Distributor, 206, activates a Locker, 208, ad imposes a lock on the model. This means that all remaining

application instances running on this process, 112, and on other processes, 114 and 116, cannot perform operations of writing and reading on this model. The distributors, 206, 214, 216, check if models are available for writing/reading (they are not if there is a lock on them). Once the lock is removed the modified model is persisted and is then available to processes (application instances) in this modified form. Based on the above one skilled in the art would recognize that the distributor, 206, plays a dual role:

- inside a process – constructing models based on Model Object Libraries and associated metadata and controlling access to these models.
- across all processes – to exchange status of models and updates of the models.

The Metadata and Model Objects for Distributed MIM management are shown in Figure 3. All MIMs are composed of MIM instances, which must be of a Model Type. The Model Type Definition, 308, holds the type name and version of the type as well as a reference to a Model Object, an object that is instantiated at run time as the MIM instance value. The type name must be unique in the Distributed MIM Management system. Model Type Authoring, 254, in the Authoring component creates Model Type Definition, 308, metadata and stores the metadata in the Model Knowledge Base.

The Model Object is an object programmed in a language such as Java. All Model Objects are sub classes of an Abstract Model Object, 312; at run time the Model Maps that hold MIMs are maps of Model Objects or façades to Model Objects. Model Objects are referenced using the name and version of the map to which they belong as well as a local name, which is unique in a given MIM model map so any shared properties are defined in the Abstract Model Object, 312, and are set at run time. It must be possible to transfer Model Objects from one process to another (serializable in Java) so all Model Objects have this property. They may have other properties such as the ability to be persisted. The Concrete Model Object, 314, definition contains any logic that the Model Object requires to implement its domain logic, for example a *Cell* Model Object may

have a Cell Id property and a *NetworkInterface* Model Object may have an *ifPhysAddress* property. Model Type Authoring in the Authoring component creates Model Objects and stores them in the Model Object repository.

The MIM Model Map Definition, 302, metadata defines MIMs with a similar scope and the content they may have. Each MIM has a name that uniquely identifies it in the Distributed MIM Management system and a version that is used to control the versions of MIMs that are being used. Each MIM contains zero or more MIM Model Instances, also defined in metadata. Each MIM Model Instance, 306, is a specific instance of a Model Type in a particular MIM, with the model type identified by the Model Type reference. Its definition contains the name and version of the Model Map that owns it and a Local Name that uniquely identifies the instance in its Model Map. For example, a cell *C1* in the MIM Model Map *CELL* has a local name of *C1*, and a cell *C2* has a local name of *C2*. The Writable Flag of an instance specifies whether applications can write a MIM instance or not. For example, the instances in a MIM containing weather data read from an external system cannot be written by applications.

The Scope of a MIM and its instances defines the visibility of the MIM in applications. A scope of *Application* means that a MIM is visible only to the application that uses it. A scope of *Global* means that any application can read and modify the MIM, and a scope of *External* means that all applications can read the MIM but it is modified externally to the Distributed MIM Management system. Other more selective scopes can also be defined.

Model Map Authoring, 252, in the Authoring component creates the metadata for MIM Model Maps and MIM Model Instances and stores the metadata in the Model Knowledge Base.

The Model Knowledge Base, 104, also contains Application Metadata, 218, which is set during configuration of any application that is using Distributed MIM Management.

The Application Definition, 304, metadata contains the name and version of each application and references to each MIM each application uses.

The Model Monitoring information, 138, in the Knowledge Base, 104, provides usage information on MIM instances. Each time an instance is initialized, read, written, locked, or unlocked, a log is produced by the Monitor, 212, component in the Model Manager, 118, of a process, 112. These logs are sent to the Model Usage Collector, 136, component, which stores them in the Model Knowledge Base, 104. Each log entry contains the identity of the MIM instance (Map name and version and local name), the time stamp of the log entry, what operation triggered the log, the call stack of the application at the time the log was made, and the value of the instance before and after the operation in question completed. Applications may use the Call Stack field of the log to insert tracing information that indicates its execution status such as what application module was running and what state it was in at the at the time the log entry was made.

The Distributed Model Deployer, 110, shown in Figure 2 distributes MIB metadata and Model Object Libraries to processes, 112 – 116. When a process starts, the Distributed Model Deployer, 110, requests a list of the applications, 130, on the process, 112, and then sends the metadata for all MIMs used by that process, 112, to the Model Manager, 118, on that process, 112. Alternatively, the Distributed Model Deployer, 110, may have the information about MIMs used by processes cached locally from earlier operations (e.g. deployments, upgrades, etc.) It then stores the Model Object Libraries for those MIMs on the process. Finally, Distributed Model Deployer, 110, asks the Model Manager, 118, on the process to set up Distributed MIM Management. If a new application is installed on a process or an application is removed from a process, the Distributed Model Deployer adds or removes the MIM metadata and Model Object Libraries as appropriate and asks the Model Manager, 118, on the process to update its Distributed MIM Management. The Distributed Model Deployer, 110, also ensures that MIM models across processes, 112 - 116, are kept consistent. If a MIM model is updated in the Model Knowledge Base, 104, the Deployer, 110, coordinates the update of the

MIM model across the Model Managers in all processes that use the MIM. See Section *MIM Metadata and Model Object Deployment* for details of these processes.

The Distributor, 206, 214, 216, in each Model Manager, 118, 120, 122, in each process, 112, 114, 116, (See Figure 2) manages the Distributed MIMs for the applications in its process. When the Distributor, 206, receives metadata for a MIM from the Distributed Model Deployer, 110, it initializes a local Model Map for that MIM, 204, with the metadata using the process described in Section *MIM Metadata and Model Object Deployment*. The Distributor, 206, distributes each MIM map to other processes using known distribution mechanisms such as Infinispan or Hazelcast. Such unstructured distribution mechanisms can be used because the Distributor, 206, defines the structure of the distributed MIM Model Maps, 204, using the MIM metadata and controls access to those MIM Model Maps, 204. Applications, 130, can then read or write the MIMs using a map interface to the MIMs provided by the Distributor, 206. The Distributor, 206, has a Locker, 208, and a Monitor, 212, and may have a Persistor, 210. The Locker, 208, provides distributed locking of MIM instances by using the MIM instance keys from the metadata as keys to a locking mechanism such as Curator or Hazelcast locks. The Distributor, 206, uses the Monitor, 212, to send a MIM Model Instance Usage Log, 310, entry to the Model Usage Collector, 136, when a MIM Instance is initiated, read, written, locked or unlocked. The Monitor, 212, uses a mechanism such as Log4j [Log4j, 2015] or syslog [IETF, 2009] to send the log messages. A Distributor may use a Persistor to save the MIM information to persistent storage. The Persistor, 210, uses the MIM metadata to define the structure of the files or database tables used to save the MIM information. The Distributor, 206, calls the Persistor, 210, periodically to save its MIMs.

Figure 4 illustrates a typical usage of distributed MIMs. Instances of three applications, 130, namely *CP* (Cell Power), *BSHVAC* (Base Station Heating, Ventilation, and Air Conditioning), and *CQOS* (Customer QoS), are running on distributed processes *P1*, *P2*, and *P3*. Those processes are running on distributed real or virtual hosts. There are nine distributed MIMs in the system as shown in Figure 4. In the case of the *CELL*

MIM, it holds information on cells in the system, it has *GLOBAL* scope and it is used by the *CP* and CQOS applications. The *TRANSPORT* MIM contains timetable and status information for motorways, rail networks, and airlines and has *EXTERNAL* scope. Each application has its own internal MIM (*BS_HVAC* in the case of the *BSHVAC* application), which is used to hold internal state information for use by that application. The *BS_HVAC* internal MIM may hold information such as the current fan speeds (RPM) in a given base station.

Instances of all applications are running in Process P1 so the Distributed Model Deployer, 110, (see Figure 2) sends the metadata for all MIMs to that process and the Model Manager in that process instantiates MIM Model Maps for each MIM. Process P2 runs instances of *CP* and *BSHVAC* applications so the five required MIMs are deployed and initialized. Process P3 is running instances of application *CQOS*, so only the MIMs used by that application are deployed and used.

In one embodiment Process P1 may be the process 112 illustrated in Figure 2 and the application instances running in Process P1 will be the application instances 130 running in process 112, whereas the MIM Model Maps in Process P1 correspond to MIM Model Maps, 204, of process, 112, in Figure 2.

Note that, in execution, each application instance uses the MIM Model Maps as if they were local, and need not be aware that maps are distributed. Distributed MIM Management ensures that reads, writes, and locks executed on any MIM instance by any application are distributed correctly. If the *BSHVAC* instance on Process *P2* writes to an instance on the *BASE_STATION* MIM, those changes are visible to the *CP* and *BSHVAC* application instances on *P1* and *P2*. Likewise, if the *CP* application on *P2* acquires a write lock on a QoS profile instance in the *QOS_PROFILE* MIM, all instances of the CP and *CQOS* applications on *P1*, *P2*, and *P3* that wish to use that QoS profile instance must wait until the *CP* application instance on *P2* releases the write lock before they can access that QoS profile instance.

Below are described details of embodiment of various aspects related to practical implementation of the solution disclosed in this document.

Metadata Authoring

The flowchart in Figure 5 shows the process for creating the metadata and Model Object for a Model type definition by the Authoring component. The Authoring component defines, 502, the Model Object as a sub class of an Abstract Model Object and ensures that persistence and other properties required by Distributed MIM Management are implemented, 504. The Authoring Component allows domain specific properties of the Model Object to be defined, this is illustrated in step 506. Once the object is written, the Authoring component stores, 508, the Model Object in the Model Object Repository. The Authoring component then stores, 510, the metadata of the Model Type in the Model Knowledge base, 104.

The flowchart in Figure 6 shows the process for creating the metadata for a MIM Model Map. The Authoring component stores the Name, Version, and Scope of the Model Map as its metadata in the Model Knowledge Base, 602. It the iterates over every member instance in the Model Map, 604, and stores the Name, Version, Instance name, Scope, whether the instance can be written, and the Model Type of the instance as its metadata in the Model Knowledge Base, 606.

The flowchart in Figure 7 shows the process for creating the metadata for MIM Model Usage by applications. An Application Configuration, 250, component stores, 704, the Name, Version, and Scope of each application as its metadata in the Model Knowledge Base. It the iterates over every MIM Model Map, 706, used by each application and stores a reference to the MIM Model Map Definition in the application's metadata, 708, in the Model Knowledge Base. These operations are performed for each application, 702.

MIM Metadata and Model Object Deployment

The flowchart in Figure 8 shows the process used by the Distributed Model Deployer, 110, (See Figure 2) for deploying and updating the metadata and Model Object Libraries for distributed MIMs in a system that is using Distributed MIM Management. The Deployer, 110, runs this process for each distributed MIM Model Map (e.g. MIM Model Map 204) to be updated in the system. For each MIM Model Map, the Distributor gets a list of processes that are using the map, 802. In parallel it locks the MIM Model Map, 804, on each process using the locking sub-process shown in Figure 9A. The Model Manager on each process locks the MIM Model Map. Once the process acquires the lock, 806, access to all MIM instances on the MIM Model Map is blocked for all applications on that process.

The Distributed Model Deployer now checks if the MIM Model Map update is compatible with the previous version of the MIM Model Map, 808; that is the version that is currently deployed. Updates are compatible if the new version of the metadata specifies only additions of instances to maps and/or only additions or extensions of MIM Model Instance definitions to MIMs. Therefore, an update that adds cell *C1234* to the *CELL* MIM is compatible but an update that deletes cell *C1* is not. An update that adds a field Description to the *CELL* Model Object is compatible but an update that deletes the field Cell Id is not. If an update is deemed to be compatible, 808-yes, the Deployer, 110, calls the Update Model Map sub-process in Figure 11 to update, 810, the MIMs on each process; otherwise, 808-no, it calls the Clear Model Map sub-process in Figure 12 followed by the Initialize Model Map sub-process, 822, in Figure 10 on each process, 824, to clear down and recreate the MIMs, 812.

Once all processes have updated, 814, or cleared and recreated, 816, their MIMs, the Deployer, 110, calls the Unlock Model Map sub-process, 818, in Figure 9B to release the locks on the distributed MIMs in each process, 820, which allows the applications to recommence using the maps.

As shown in Figure 9A locking a Model Map, 902, starts with a request, 904, from an application instance to impose a lock on a Model Map in a process. In the next step the Model Map is Locked, 906. Figure 9B illustrates unlocking a Model Map, 908. The operation starts with a request, 910, from the application instance that earlier requested the lock, to release the lock on a Model Map. In the next step the Lock is released, 912.

The flowchart in Figure 10 shows the process used by a Model Manager, 118, in a process, 112, (See Figure 2) to initialize its local copies of distributed MIM Model Maps, 1002. Firstly, the Model Manager stores, 1004, the Object Library for the MIM to its Process Model Object Library. It then calls the sub-process in Figure 13 to initialize, 1008, its Distributor, 206, component if the Distributor component is not already initialized, 1006-no. It then uses the MIM metadata to create, 1010, a Model Map for the MIM in the distribution mechanism. It then iterates, 1012, over each MIM Model Instance Definition in the metadata for the MIM, creating a MIM Model Instance, 1014, using the process shown in Figure 14. It then stores the instance on the MIM Model Map, 1016.

The flowchart in Figure 11 shows the process used by a Model Manager, 118, in a process, 112, (See Figure 2) to update its local copies of distributed MIM Model Maps, 1102. Firstly, the Model Manager, 118, updates, 1104, the Object Library for the MIM in its Process Model Object Library. It then used the MIM metadata to iterate over each MIM Model Instance Definition, 1106, in the metadata for the MIM, creating a MIM model instance, 1108, using the process shown in Figure 14. It then checks to see if that instance is already on the MIM Model Map, 1110. If not, 1110-no, the MIM Model Instance is stored, 1112, on the MIM model map. If the MIM Model Instance is already on the MIM Model Map, 1110-yes, the Model Manager, 118, checks, 1114, if the version of the MIM Model Instance on the MIM Model Map is equal to the incoming MIM and, if not, 1114-no, it transfers, 1116, the data from the MIM Model Instance on the map to the incoming MIM Model Instance and stores, 1112, the incoming MIM Model Instance on the MIM Model Map. If the version of the MIM Model Instance on the MIM Model Map is equal to the incoming MIM, 1114-yes, the process goes back to step 1106.

The flowchart in Figure 12 shows the process used by a Model Manager, 118, in a process, 112, (See Figure 2) to clear a MIM Model Map on a process, 1202. The Model Manager, 118, iterates over each instance on the map, 1204. The Model Manager, 118, reads, 1206, Model Instance value from a Model Map. If persistence is active, 1208-yes, it saves, 1210, the value of the MIM model instance. It then deletes, 1212, the MIM Model Instance from the MIM Model Map. If persistence is not active, 1208-no, the Model Manager, 118, deletes, 1212, the MIM Model Instance from the MIM Model Map. When all instances have been deleted, 1204-yes, the Model Manager, 118, deletes, 1214, the Model Map for the MIM in the distribution mechanism. Finally, it removes, 1216, the Model Object Library for the MIM from the Process Model Object Library.

The flowchart in Figure 13 shows the process used by a Model Manager, 118, in a process, 112, (See Figure 2) to initialize its Distributor, 206, component, 1302. The Model Manager initializes, 1304, the distribution mechanism of model maps between processes using this Distributed MIM Management system. In this step, a mechanism such as Infinispan or Hazelcast is initialized; using whatever specific method is required by the mechanism in question. The Model Manager then initializes, 1306, a Locker, 208, and its underlying locking mechanism. In this step, a mechanism such as Curator or Hazelcast Locks is initialized; using whatever specific method is required by the mechanism in question.

The Model Manager, 118, then initializes, 1308, a Monitor, 212, and its underlying monitoring mechanism. In this step, a mechanism such as Syslog or Java Logging is initialized; using whatever specific method is required by the mechanism in question. The Model Manager, 118, now checks if persistence is configured for activation, 1310. If so, 1310-yes, the Model Manager, 118, initializes, 1312, a Persistor, 210, and its underlying persistence mechanism using whatever specific method is required by the mechanism in question. The Model Manager, 118, then uses the MIM metadata for each MIM Model Map to create tables, 1314, in the persistence mechanism for storage of MIM information. Finally, the Model Manager, 118, initializes periodic

flushing, 1316, of MIM information to persistent storage using whatever interval for flushing that is configured in the system. If persistence is not configured for activation, 1310-no, the process stops.

The flowchart in Figure 14 shows the process used by a Model Manager, 118, in a process, 112, (See Figure 2) to set up and initialize a MIM Model Instance Object and its value, 1402. Firstly, the Model Manager, 118, uses the MIM Model Instance Definition metadata, 1404, to get a reference to the object in the Model Object Library that represents the Model Instance. It then creates an instance, 1406, of the Model Object and logs, 1408, initiation of the instance to the Monitor, 212. The Model Manager, 118, now uses the Distributor, 206, to check if the MIM Model Instance already exists in the distributed MIM Model Map, 1410, having being put there by another process. If so, 1410-yes, the Model Manager, 118, reads, 1412, the value of the MIM Model Instance from the MIM Model Map and stores, 1414, it into the Model Object.

If the value of the MIM Model Instance is not yet on the map, 1410-no, the Model Manager, 118, checks if persistence is active, 1416. If persistence is active, 1416-yes, the Model Manager attempts to read, 1418, the MIM Model Instance value from persistent store. If the value exists in the persistent store, 1420-yes, that value is stored, 1414, into the Model Object. Otherwise, 1420-no, a default value is stored into the Model Object, 1422. If persistence is not active, 1416-no, a default value is stored into the Model Object, 1422.

Reading from and Writing to Model Maps for MIMs in Processes

Figure 15 shows how Distributed MIM Management uses an underlying distribution mechanism such as Infinispan [Infinispan, 2016] or Hazelcast [Hazelcast, 2016] to pass changes in MIM Model Maps between processes. Each process has a local copy of each MIM Model Map. When an application instance changes a MIM Model instance on a MIM Model Map, the Distributor propagates that change to the distribution

mechanism, which updates the instance on all local copies of the MIM Model Map in processes *D1* to *D5* of the Distributed MIM Management system.

The flowchart in Figure 16 shows how an application reads a MIM instance from a MIM Model Map. The application requests, 1602, a MIM Model instance, supplying its key; the Distributor returns, 1604, a clone of the MIM Model instance from the MIM Model Map to the application and logs, 1606, the read operation to the Monitor. Each process has a local copy of its MIM Model Maps so a read operation simply reads from the local copy without using the underlying distribution mechanism. The flowchart in Figure 19 shows reading in the case where locking is applied.

The flowchart in Figure 17 shows how an application writes a MIM instance to a MIM Model Map. The application requests, 1702, a MIM Model instance using its key; the Distributor returns, 1704, a clone of the MIM Model instance from the MIM Model Map to the application and logs, 1706, the read operation to the Monitor. The Application then updates, 1708, the cloned MIM Model instance, making whatever changes it requires. The application now requests, 1710, a write of the MIM Model Map, supplying the key and the updated MIM Model instance. The Distributor, 206, writes, 1712, the updated Model instance to the MIM Model Map, 204. The underlying distribution mechanism (e.g. Infinispan or Hazelcast) propagates, 1714, the change to all other local copies of the MIM Model Map in other processes, as shown in Figure 15. The Distributor, 206, then logs, 1716, the write operation to the Monitor. The flowchart in Figure 19 shows writing in the case where locking is applied.

Locking and Unlocking Instances on Model Maps for MIMs in Processes

Figure 18 shows how Distributed MIM Management uses an underlying locking mechanism such as Curator [Curator, 2015] or Hazelcast Locking [Hazelcast, 2016] to lock and unlock MIM Model Map instances across processes. When an application requests a lock on a MIM Model map instance, the Distributor, 206, in the Model Manger, 118, passes the request to its Locker, 208. The Locker, 208, uses the key from the

metadata of the MIM Model Map instance as a key to a Lock Manager, 1802, in the underlying locking mechanism. Because the Lock Manager, 1802, of the underlying locking mechanism is distributed, only one application in a process can hold a lock at any one time. If two application instances request a lock, the request that reaches the Lock Manager first acquires the lock. The request from the second application instance is queued, with that application instance left waiting until the first application instance releases the lock. The second application instance then acquires the lock. The Lock Manager, 1802, in the distributed locking mechanism maintains a record of all locks that are active.

The flowchart in Figure 19 shows the process of an application acquiring and releasing a read or write lock. The application requests, 1902, a lock on a MIM Model instance from the Distributor using the instance key from the MIM Model Instance metadata; the Distributor requests the lock for this instance from the Locker, 1904, which forwards the request to the Lock Manager in the distributed locking mechanism. The Lock Manager, 1802, returns the lock if the MIM Model Instance key is not already locked. If the MOM Model Instance key is locked, the request is queued in the Lock Manager, 1802, and the request is answered when the lock becomes available. The distributed lock mechanism returns the lock to the Locker, which returns, 1906, it to the Distributor. The Distributor logs the lock operation to the Monitor, 1910, and the lock request returns, 1908, to the application.

The application performs whatever work it requires to perform when it has locked access to the MIM Model Instance, 1912. Once the application has completed this work, it requests release, 1914, of the lock on the MIM Model instance to the Distributor using the MIM Model instance key from the MIM Model Instance metadata. The Distributor requests release of the lock, 1916, for this instance from the Locker, which forwards the request to the Lock Manager, 1802, in the distributed locking mechanism. The Lock Manager, 1802, releases the lock and returns to the Locker, which returns to the

Distributor. The Distributor logs, 1920, the lock release operation to the Monitor and the lock release request returns to the application, 1918.

Monitoring Operations on Instances on Model Maps for MIMs in Processes

Figure 20 shows how Distributed MIM Management uses an underlying monitoring mechanism such as syslog or Java Logging to log operations. The Distributor, 206, 214, 216, in each process, 112, 114, 116, logs MIM Model Instance initiations when it creates MIM Model Maps with the key read from the MIM Model Instance metadata and logs read, write, lock, and unlock operations as they are requested by applications. The Model Usage Collector, 136, receives these logs and stores them, 138, in the Model Knowledge base, 104, in the format shown in Figure 3.

Persisting Model Maps for MIMs in Processes

Figure 21 shows how Distributed MIM Management uses an underlying persistence mechanism such as a Relational Database or distributed storage system to store the information in MIMs persistently. It is not normally necessary to activate persistence on all processes because it is adequate to save a single copy of the MIM Model information to the persistent store. However, for redundancy, a two or more process may be used. In the scenario shown in Figure 21, persistence is active only on processes *PE1* and *PE2*.

The tables used by the persistence mechanism may be created using the MIM metadata in the manner explained in Figure 13 and Section *MIM Metadata and Model Object Deployment*. Once created, the MIM Model Maps are stored to the persistence mechanism using the process described in Figure 22.

The flowchart in Figure 22 shows the process of periodically persisting MIM Model Maps. The process waits for a certain configurable length of time, 2202. It then iterates, 2204, 2206, over each MIM Model Map in the process, reading, 2208 each MIM

Model Instance in turn. If the Instance has been modified, 2210-yes, since the last persistence operation, the MIM Model Instance is persisted, 2212 to the persistence mechanism using its MIM Model Instance metadata.

Distributed MIM Management for Context in the Apex Policy System

The Apex Policy System [Van Der Meer et al., 2015] runs Adaptive Policies. Policies can be run in parallel. Figure 23 shows an Apex policy engine running three instances of the policy in a process. Each policy uses Policy Context C_p to hold its internal variables. Global Context C_g holds variables that are available to all policies.

The MIM Model Maps used to manage context in Apex are shown in Figure 24. Each policy has its own individual Policy Context MIM Model Map C_pi, and all policies use and set the Global Context MIM Model Map C_g.

The use of MIM Model Maps to share Policy and Global context has been implemented in Apex and has been demonstrated to work on processes running on multiple hosts.

Tracking of Denial of Service State Information for Distributed Applications

Consider a distributed management system. The management system is managing a large network and is itself distributed, with many cooperating instances of the management system running on different physical (possibly virtualized) hosts in different physical locations. Many instances of a Denial of Service attack mitigation application run in the different instances of the management system. Any of the instances of the Denial of Service application may receive information about any subscriber and may have to take action at any time.

In this application, four possible states exist for each subscriber, namely:

1. No suspicious activity detected.

2. Suspicious signaling activity, for example an unusual number of attaches to the network or an unusual number of session creations may be reported by the Radio Access Network, causing the DoS mitigation application to order an examination of the user plane traffic on this subscriber.

3. The user plane traffic indicates traffic from suspicious sources such as blacklisted IP addresses, or suspicious traffic patterns. In this case the DoS mitigation application orders that the user's traffic be directed into special Software Defined Network used for quarantining traffic.

4. The user is quarantined.

The DoS mitigation application is fully distributed so any instance of the application on any management system may require controlled, safe, in-memory access to the structured information on a subscriber that is defined in metadata.

Take, for example, the case where three management system instances, A, B, and C exist, perhaps running on three different distributed computers. Each instance of the management system is running the DoS mitigation application, with perhaps hundreds of threads running in each DoS mitigation application to handle the load of the millions of subscribers using the network.

The radio access network reports suspicious signaling activity on Subscriber Number 98765432 (See 2 above) to the management system and instance x on management system A receives the message. A thread in A_x acquires a read lock and reads the subscriber data for 98765432 (if any) from the Subscriber Model Map and sees that the subscriber has no previous suspicious activity logged, triggering the system to record that a read of the Subscriber Model Map was carried out. A_x now orders user plane traffic examination on subscriber 98765432 and records that information to the Subscriber Model Map record for 98765432. It does this by acquiring a write lock on record 98765432, recording the details of the suspicious signaling activity and setting the "user

plane traffic examination active" flag on the record together with the time of activation and any other relevant information, triggering the system to record that a write of the Subscriber Model Map was carried out. The system persists the record if persistence is active.

User plane traffic examination reports the user plane activity on Subscriber Number 98765432 (See 3 above) to the management system and instance z on management system C receives the message. A thread in C_z acquires a read lock and reads the subscriber data for 98765432 from the Subscriber Model Map and sees the previous suspicious signaling activity on this subscriber and the indication that user plane examination was ordered, triggering the system to record that a read of the Subscriber Model Map was carried out. C_z now orders quarantine of traffic on subscriber 98765432 and records that information to the Subscriber Model Map record for 98765432. It does this by acquiring a write lock on record 98765432, recording the user plane activity, clearing the "user plane traffic examination active" flag, and setting the "user traffic quarantine ordered" flag on the record together with the time of ordering and any other relevant information, triggering the system to record that a write of the Subscriber Model Map was carried out. The system persists the record if persistence is active.

The Traffic Quarantine application reports that traffic on Subscriber Number 98765432 has been quarantined (See 4 above) to the management system and instance Y on management system B receives the message. A thread in B_y acquires a read lock and reads the subscriber data for 98765432 from the Subscriber Model Map and sees that quarantine of traffic on this subscriber was indeed ordered. This triggers the system to record that a read of the Subscriber Model Map was carried out. By now records that traffic on subscriber 98765432 has been quarantined to the Subscriber Model Map record for 98765432. It does this by acquiring a write lock on record 98765432, clearing the "user traffic quarantine ordered" flag, and setting the "user traffic quarantine active" flag on the record together with the time of activation and any other relevant information,

triggering the system to record that a write of the Subscriber Model Map was carried out. The system persists the record if persistence is active.

Note that three fully independent instances of the DoS mitigation application (A_x, C_z, and B_y) accessed and set the subscriber information in the Subscriber Model Map, and it was declared that the DoS mitigation application may access the Subscriber Model Map. Note also that the DoS mitigation application can write and read only the information that is defined on the Subscriber Model Map and may not itself change the structure or schema of that information.

Of course, the radio network analytics application that detects suspicious signaling, the application that probes user plane data, and the application for traffic quarantining may also read and/or set record 98765432 of the Subscriber Model Map in a controlled, synchronized and audited manner. However, they must also have declared that they will access the Subscriber Model Map, and again they may not change the structure of the Subscriber Model Map.

Abbreviations:

ACID	Atomicity, Consistency, Isolation, Durability
ADMF	Administration Function
API	Application Programming Interface
CSP	Cloud Service Provider
DF	Delivery function
E2E	End to End
EM	Element Management
ETSI	European Telecommunications Standards Institute
FB	Functional Block
FCAPS	Fault Configuration Accounting Performance and Security
FEP	Front End Processor
FG	Forwarding Graph
HDFS	Hadoop Distributed File System
HW	Hardware
ICEs	Intercepting Control Elements
IRP	Integration Reference Point
ISG	Industry Specification Group
ISP	In service Performance
IT	Information Technology
JTA	Java Transaction API
KPI	Key Performance indicator
LEMF	Law Enforcement Monitoring Facilities
MANO	Management and Orchestration
MIB	Management Information Base
MIB II	Management Information Base version 2
MIM	Management Information Model
NE	Network Element
NF	Network Function
NFV	Network Functions Virtualisation
NFVO	Network Functions Virtualisation Orchestrator
NRM	Network Resource Model
NS	Network Service
NSD	Network Service Descriptor
OS	Operating System
PI	Performance indicator
PM	Performance Management
PNF	Physical Network Function
QoS	Quality of Service
RBAC	Role Based Access Control
RDBMS	Relational Database Management System
SDN	Software Defined Networking

SNMP	Simple Network Management Protocol
SOD	Separation of duty
SW	Software
TAP	TAP linux device interface
vCPU	virtual Central Processing Unit
VDU	Virtualisation Deployment Unit
VI	Virtual Infrastructure
VIM	Virtual Infrastructure Manager
VL	Virtual Link
VLAN	Virtual Local Area Network
VLD	Virtual Link Descriptor
VM	Virtual Machine
VNF	Virtualised Network Function
VNFC	Virtual Network Function Component
VNFCI	Virtual Network Function Component Instance
VNFD	Virtual Network Function Descriptor
VS	Virtual Storage

References:

3GPP, 2011a	3GPP, "Bulk CM Integration Reference Point (IRP): Information Service (IS)", 3GPP, no. 3GPP TS 32:612, Jan 2011
Bresniker et al., 2015	Bresniker, K.M., Singhal, S. & Williams, R.S., "Adapting to Thrive in a New Economy of Memory Abundance", Computer, vol. 48, no. 12, pp. 44-53, Dec 2015
Cecchet et al., 2002	Cecchet, E., Marguerite, J. & Zwaenepoel, W., "Performance and Scalability of EJB Applications", SIGPLAN Not., ACM, vol. 37, no. 11, pp. 246-261, Nov 2002
Coulouris et al., 2012	Coulouris, G., Dollimore, J., Kindberg, T. & Blair, G., "Distributed Systems: Concepts and Design", Addison-Wesley, ISBN 978-0-13-214301-1, 2012
Curator, 2015	Curator, A., "Apache Curator Getting Started Guide", Apache Curator, Oct 2015
ETSI, 2014	ETSI, "Network Functions Virtualisation (NFV); Management and Orchestration", ETSI, no. GS NFV-MAN 001, Dec 2014
Ericsson, 2016	Ericsson, "Ericsson Network Manager (ENM)", http://www.ericsson.com/ourportfolio/products/network-manager, Feb 2016
Femminella et al., 2011	Femminella, M., Maccherani, E. & Reali, G., "Performance Management of Java-based SIP Application Servers", Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on, pp. 493-500, May 2011
Hazelcast, 2016	Hazelcast, "Hazelcast Documentation", Hazelcast, May 2016
IETF, 1991	IETF, "Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II", IETF, RFC Editor, United States, no. RFC 1213, Mar 1991
IETF, 2009	IETF, "The Syslog Protocol", IETF, no. RFC 5424, Mar 2009
ITU-T, 2000b	ITU-T, "Principles for a Telecommunications Management Network", ITU-T, ITU, no. M.3010, Feb 2000
ITU-T, 2005b	ITU-T, "Generic Network Information Model", ITU-T, ITU, no. M.3100, Apr 2005
Infinispan, 2016	Infinispan, "Infinispan Documentation", Infinispan, Feb 2016
Little et al., 2016	Little, M., Halliday, J., Dinn, A., Connor, K., Musgrove, M., Robinson, P., Trikleris, G. & Feng, A., "Narayana Transaction Manager Project Documentation", jboss.org, Jan 2016
Log4j, 2015	Log4j, A., "Apache Log4j User's Guide", Apache Log4j, Dec 2015
Oracle, 1999	Oracle, "Java Transaction API (JTA)", Oracle, Apr 1999
Protic et al., 1996	Protic, J., Tomasevic, M. & Milutinovic, V., "Distributed Shared Memory: Concepts and Systems", Parallel Distributed Technology: Systems Applications, IEEE, vol. 4, no. 2, pp. 63-71, Sum 1996
Van Der Meer et al., 2015	WO2016/066438

CLAIMS

1. A method of managing a network, the network comprising a plurality of distributed hosts, wherein at least one process is being run on a host, said process providing a function related to operation of the network by executing at least one application instance, wherein the at least one application instance requires for execution at least one Management Information Model, the method comprising:

- receiving Model Object Libraries representing the at least one Management Information Model and Model Metadata associated with the Model Object Libraries;
- instantiating the at least one Management Information Model based on the received Model Object Libraries and Model Metadata;
- storing the instantiated Management Information Model in a repository of the process for use by the at least one application instance.

2. The method according to claim 1, wherein said at least one Management Information Model is used by other instances of said application as well as by instances of other applications.

3. The method according to claim 2 comprising:

- receiving from a first application instance a request for a write access to the Management Information Model stored in the repository;
- locking said Management Information Model if the Management Information Model is available for modification, whereas in a locked state other application instances in the or another host are not allowed to read said locked Management Information Model and are not allowed to write in said locked Management Information Model;
- modifying said Management Information Model, in a write operation, based on the request received from the first application instance;
- storing the modified Management Information Model;

- unlocking said Management Information Model to make said Management Information Model available for reading and writing operations for applications in the or another host.

4. The method according to claim 3, wherein checking if the Management Information Model is available for modification comprises checking if a locked state is imposed on said Management Information Model by any process running on any host in the network and wherein said Management Information Model is available for modification only if it is not in a locked state.

5. The method according to any one of preceding claims comprising:

- monitoring operations of: instantiating a Management Information Model, reading a Management Information Model, writing to a Management Information Model, locking and unlocking a Management Information Model; and
- sending usage logs entries containing monitoring data to a Model Usage Collector which centrally records Management Information Model usage data for the network.

6. The method according to claim 1 comprising:

- imposing a write lock state on a Management Information Model if said Management Information Model contains data from a system external to the network.

7. The method according to any one of preceding claims, wherein the distributed hosts are distributed in various geographical locations.

8. The method according to any one of preceding claims, wherein the at least one Management Information Model represented by the received Model Object Libraries and Model Metadata associated with the Model Object Libraries has been identified as being used by the at least one process.

9. An apparatus for managing a network, the network comprising a plurality of distributed hosts, wherein at least one process is being run on a host, said process providing a function related to operation of the network by executing at least one application instance, wherein the at least one application instance requires for execution at least one Management Information Model, the apparatus comprising a processor and a memory, the memory containing instructions executable by the processor such that the apparatus is operative to:

- receive Model Object Libraries representing the at least one Management Information Model and Model Metadata associated with the Model Object Libraries;
- instantiate the at least one Management Information Model based on the received Model Object Libraries and Model Metadata; and
- store the instantiated Management Information Model in a repository of the process for use by the at least one application instance.

10. The apparatus according to claim 9, wherein said at least one Management Information Model is used by other instances of said application as well as by instances of other applications.

11. The apparatus according to claim 10 operative to:

- receive from a first application instance a request for a write access to the Management Information Model stored in the repository;
- lock said Management Information Model if the Management Information Model is available for modification, whereas in a locked state other application instances in the or another host are not allowed to read said locked Management Information Model and are not allowed to write in said locked Management Information Model;
- modify said Management Information Model, in a write operation, based on the request received from the first application instance;
- store the modified Management Information Model;

- unlock said Management Information Model to make said Management Information Model available for reading and writing operations for applications in the or another host.

12. The apparatus according to claim 11, wherein in checking if the Management Information Model is available for modification the apparatus is operative to check if a locked state is imposed on said Management Information Model by any process running on any host in the network and wherein said Management Information Model is available for modification only if it is not in a locked state.

13. The apparatus according to any one of claims 9 - 12 operative to:

- monitor operations of: instantiating a Management Information Model, reading a Management Information Model, writing to a Management Information Model, locking and unlocking a Management Information Model; and
- send usage logs entries containing monitoring data to a Model Usage Collector which centrally records Management Information Model usage data for the network.

14. The apparatus according to claim 9 operative to impose a write lock state on a Management Information Model if said Management Information Model contains data from a system external to the network.

15. The apparatus according to any one of claims 9 - 14, wherein the distributed hosts are distributed in various geographical locations.

16. The apparatus according to any one of claims 9 - 15, wherein the at least one Management Information Model represented by the received Model Object Libraries and Model Metadata associated with the Model Object Libraries has been identified as being used by the at least one process.

17. An apparatus for managing a network, the network comprising a plurality of distributed hosts, wherein at least one process is being run on a host, said process providing

a function related to operation of the network by executing at least one application instance, wherein the at least one application instance requires for execution at least one Management Information Model, the apparatus comprising:

- a receiver for receiving Model Object Libraries representing the at least one Management Information Model and Model Metadata associated with the Model Object Libraries;
- a creator for instantiating the at least one Management Information Model based on the received Model Object Libraries and Model Metadata;
- a memory for storing the instantiated Management Information Model in a repository of the process for use by the at least one application instance.

18. The apparatus according to claim 17, wherein said at least one Management Information Model is used by other instances of said application as well as by instances of other applications.

19. The apparatus according to claim 18, wherein the receiver is further operative to receive from a first application instance a request for a write access to the Management Information Model stored in the repository, wherein the apparatus comprises a locker for locking said Management Information Model, whereas in a locked state other application instances in the or another host are not allowed to read said locked Management Information Model and are not allowed to write in said locked Management Information Model and unlocking said Management Information Model to make said Management Information Model available for reading and writing operations for applications in the or another host, wherein the creator is further operative to modify said Management Information Model, in a write operation, based on the request received from the first application instance and the memory is operative to store the modified Management Information Model.

20. The apparatus according to claim 19, wherein in checking if the Management Information Model is available for modification the apparatus is operative to check if a

locked state is imposed on said Management Information Model by any process running on any host in the network and wherein said Management Information Model is available for modification only if it is not in a locked state.

21. The apparatus according to any one of claims 17 - 20 comprising:

- a monitor for monitoring operations of: instantiating a Management Information Model, reading a Management Information Model, writing to a Management Information Model, locking and unlocking a Management Information Model; and
- a reporter for sending usage logs entries containing monitoring data to a Model Usage Collector which centrally records Management Information Model usage data for the network.

22. The apparatus according to claim 17 operative to impose a write lock state on a Management Information Model if said Management Information Model contains data from a system external to the network.

23. The apparatus according to any one of claims 17 to 22, wherein the distributed hosts are distributed in various geographical locations.

24. The apparatus according to any one of claims 17 - 23, wherein the at least one Management Information Model represented by the received Model Object Libraries and Model Metadata associated with the Model Object Libraries has been identified as being used by the at least one process.

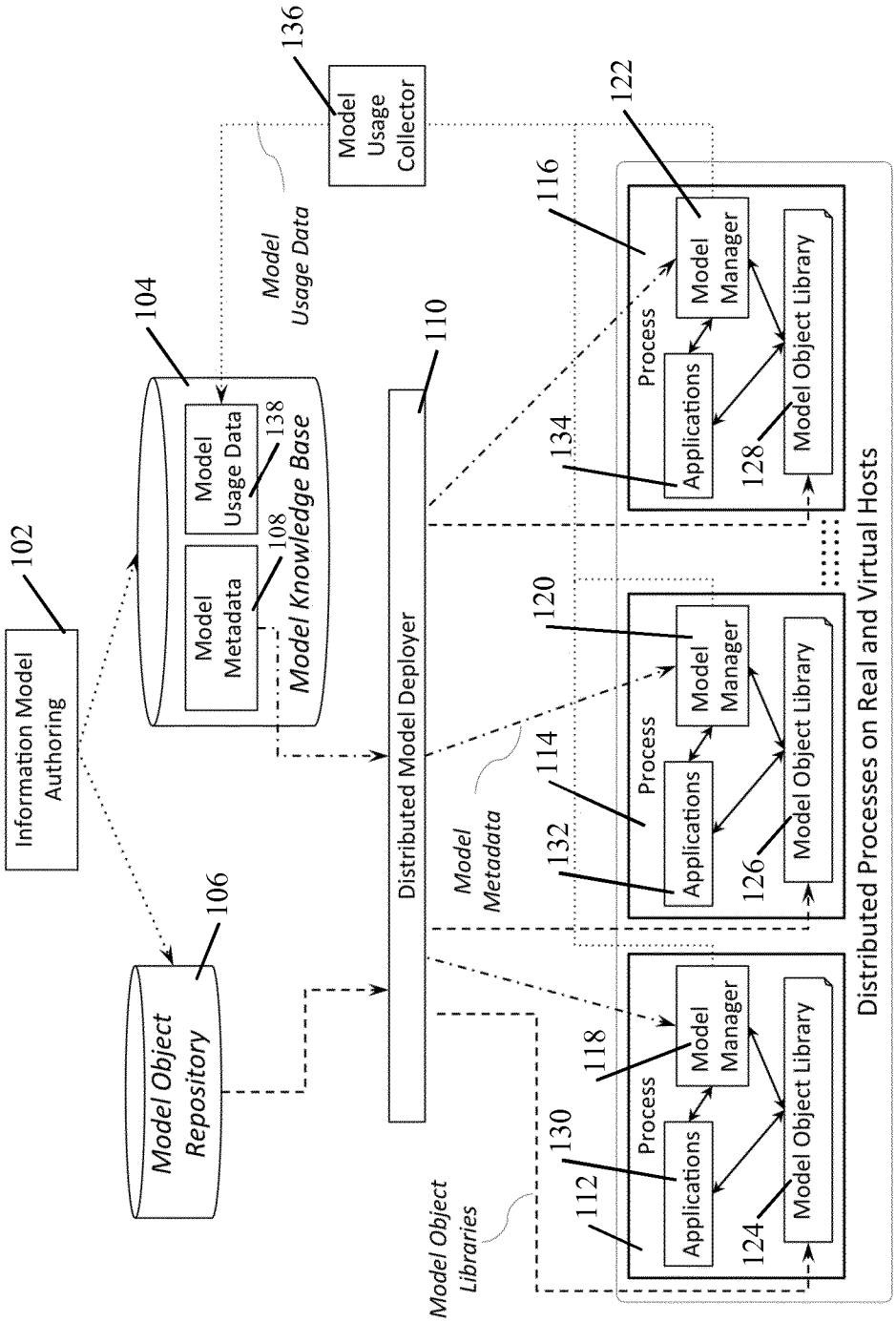
Information Model Authoring
102
Model Object Repository
106
Model Knowledge Base
104
Model Metadata
Model Usage Data
108
138
Model Usage Collector
136
Model Usage Data
Distributed Model Deployer
110
Model Metadata
Model Object Libraries
Distributed Processes on Real and Virtual Hosts
Process
Applications
Model Manager
Model Object Library
112
114
116
118
120
122
124
126
128
130
132
134

Fig. 1

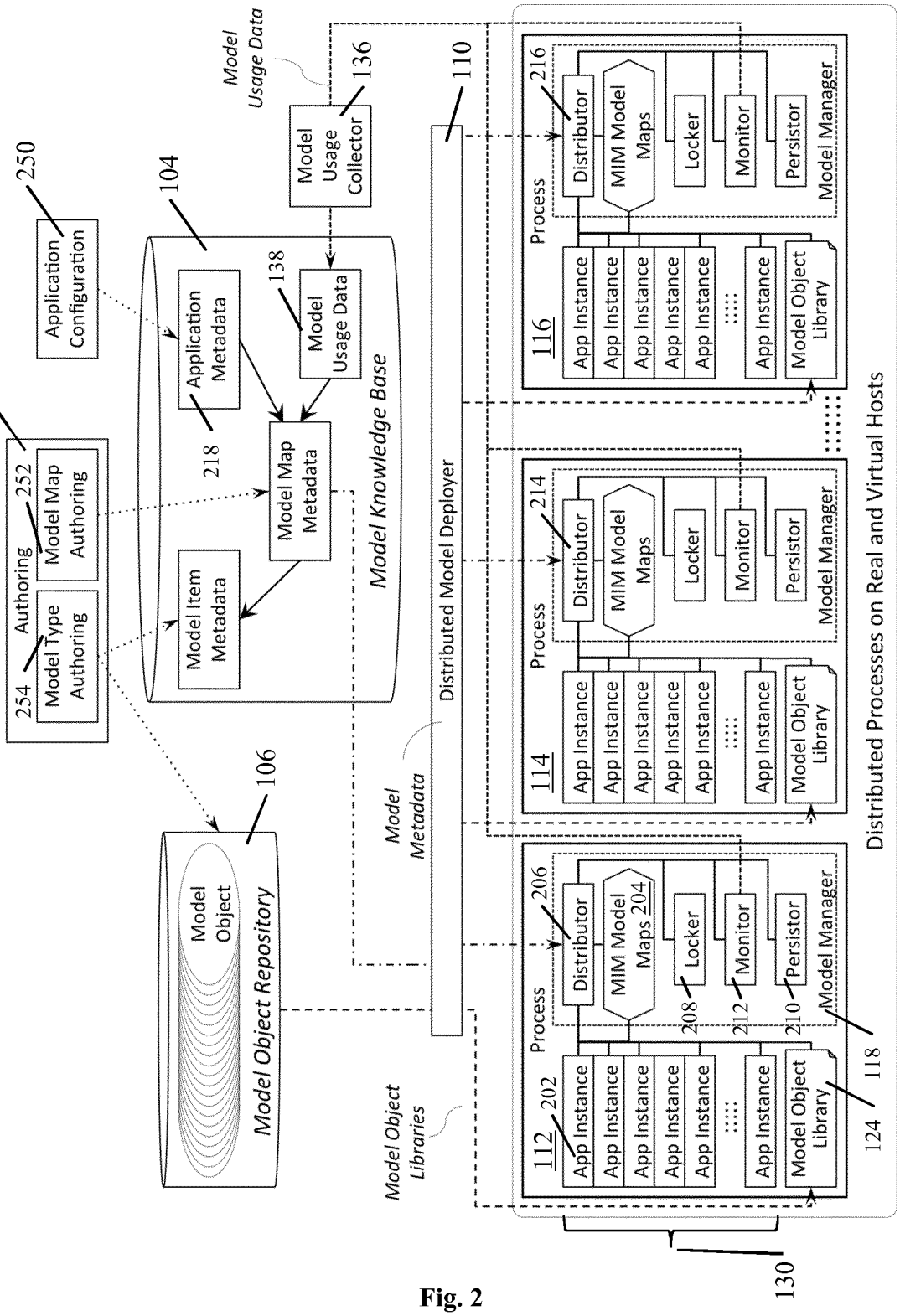
Model Usage Data
136
Model Usage Collector
110
Distributed Model Deployer
250
Application Configuration
104
Application Metadata
138
Model Usage Data
218
Model Map Metadata
Model Item Metadata
Model Knowledge Base
102
Authoring
252
Model Map Authoring
254
Model Type Authoring
106
Model Object
Model Object Repository
Model Metadata
Model Object Libraries
Process
Distributor
MIM Model Maps
Locker
Monitor
Persistor
Model Manager
App Instance
Model Object Library
216
214
206
204
208
212
210
112
202
114
116
118
124
130
Distributed Processes on Real and Virtual Hosts

Fig. 2

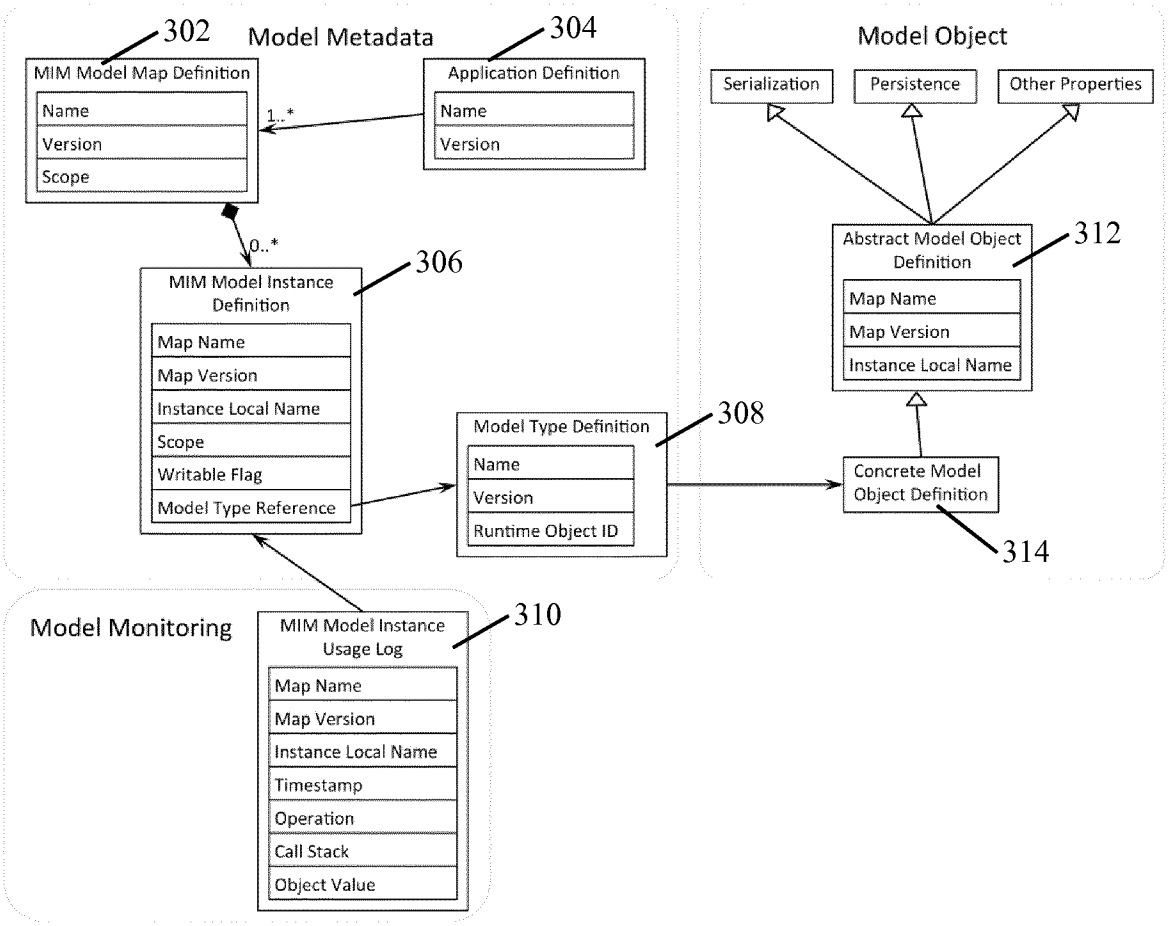
302
Model Metadata
304
MIM Model Map Definition
Name
Version
Scope
1..*
Application Definition
Name
Version
0..*
306
MIM Model Instance Definition
Map Name
Map Version
Instance Local Name
Scope
Writable Flag
Model Type Reference
Model Type Definition
308
Name
Version
Runtime Object ID
Model Object
Serialization
Persistence
Other Properties
Abstract Model Object Definition
312
Map Name
Map Version
Instance Local Name
Concrete Model Object Definition
314
Model Monitoring
MIM Model Instance Usage Log
310
Map Name
Map Version
Instance Local Name
Timestamp
Operation
Call Stack
Object Value

Fig. 3

Applications

Name	Description
CP	Cell Power
BSHVAC	Base Station HVAC
CQOS	Customer QoS

Model Maps for Distributed MIMs

Name	Scope	Applications
CELL	Global	CP,CQOS
BASE_STATION	Global	CP,BSHVAC
CUSTOMER	Global	CQOS
QOS_PROFILE	Global	CQOS, CP
METEO	External	CP, BSHVAC
TRANSPORT	External	CQOS
CELL_POWER	Application	CP
BS_HVAC	Application	BSHVAC
CUSTOMER_QOS	Application	CQOS

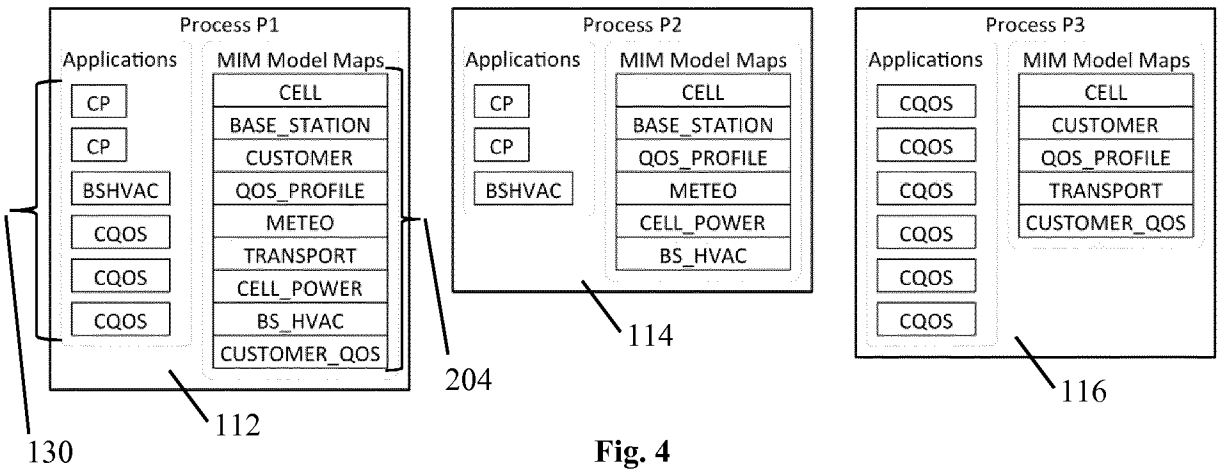

Fig. 4

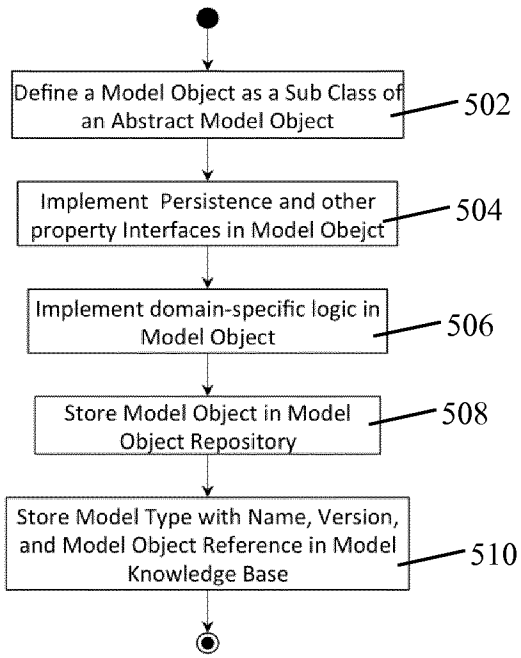

Fig. 5

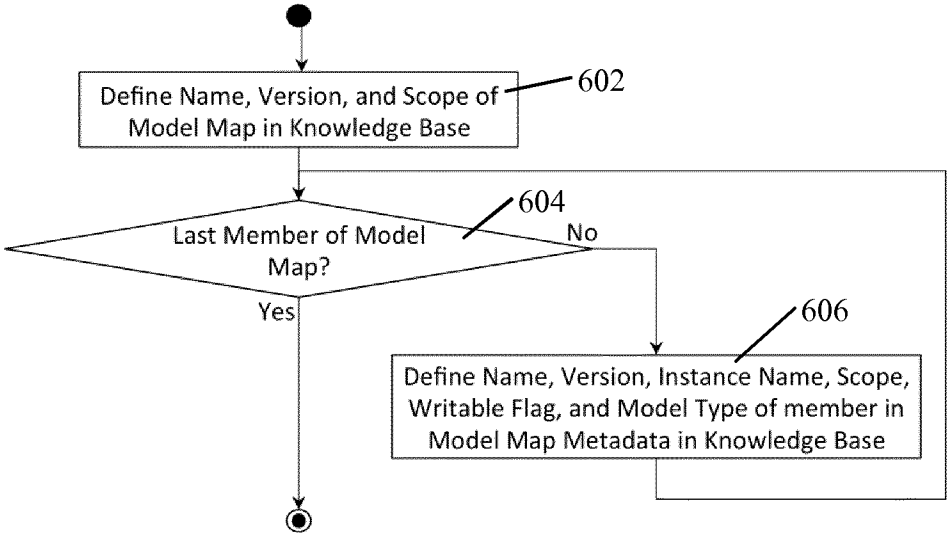
Define Name, Version, and Scope of Model Map in Knowledge Base
602
Last Member of Model Map?
604
No
Yes
606
Define Name, Version, Instance Name, Scope, Writable Flag, and Model Type of member in Model Map Metadata in Knowledge Base

Fig. 6

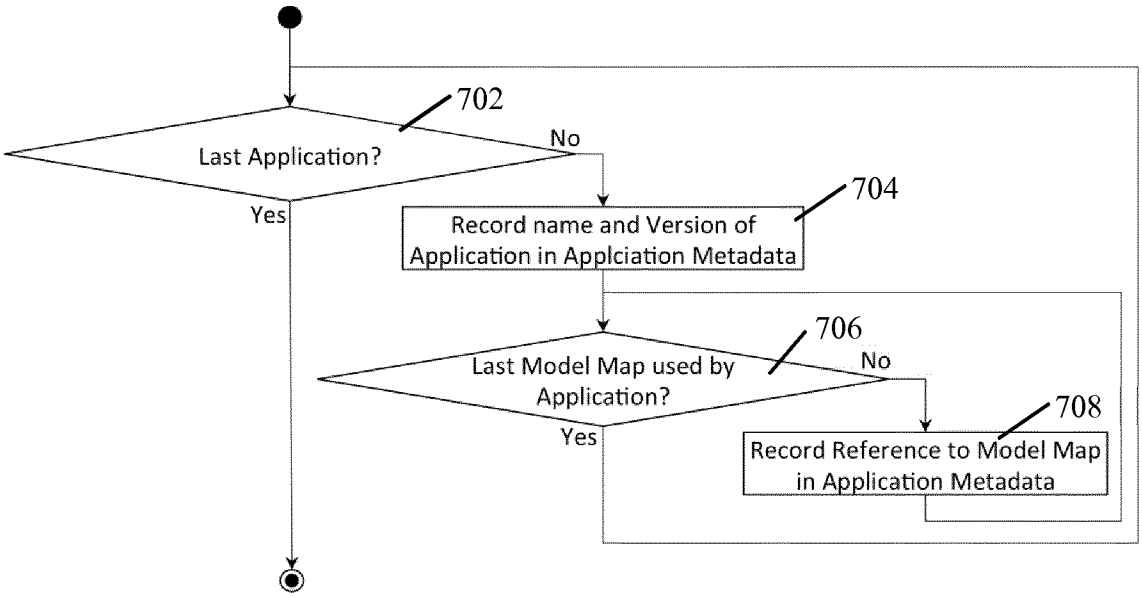
Last Application?
702
No
Yes
Record name and Version of Application in Applciation Metadata
704
Last Model Map used by Application?
706
No
Yes
Record Reference to Model Map in Application Metadata
708

Fig. 7

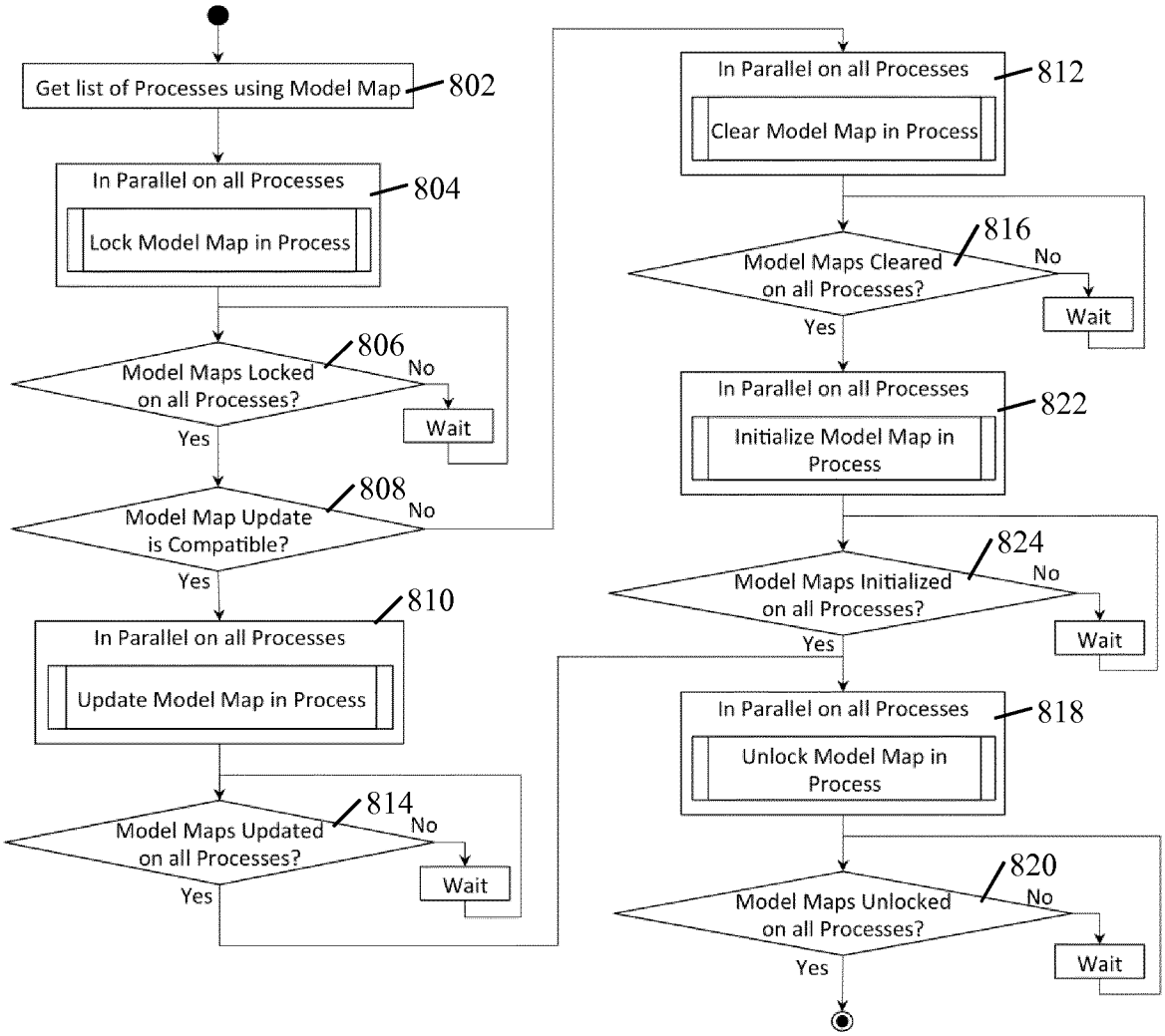
Get list of Processes using Model Map
802
In Parallel on all Processes
Lock Model Map in Process
804
Model Maps Locked on all Processes?
806
No
Wait
Yes
Model Map Update is Compatible?
808
No
Yes
In Parallel on all Processes
Update Model Map in Process
810
Model Maps Updated on all Processes?
814
No
Wait
Yes
In Parallel on all Processes
Clear Model Map in Process
812
Model Maps Cleared on all Processes?
816
No
Wait
Yes
In Parallel on all Processes
Initialize Model Map in Process
822
Model Maps Initialized on all Processes?
824
No
Wait
Yes
In Parallel on all Processes
Unlock Model Map in Process
818
Model Maps Unlocked on all Processes?
820
No
Wait
Yes

Fig. 8

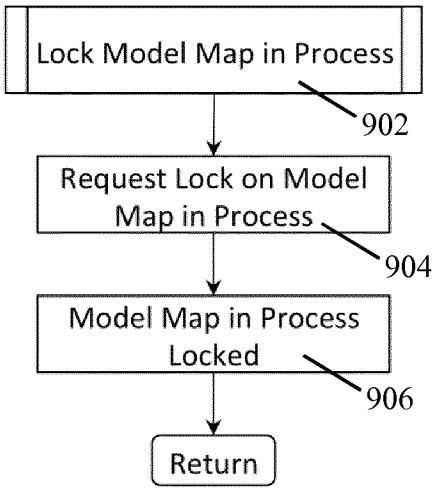

Fig. 9A

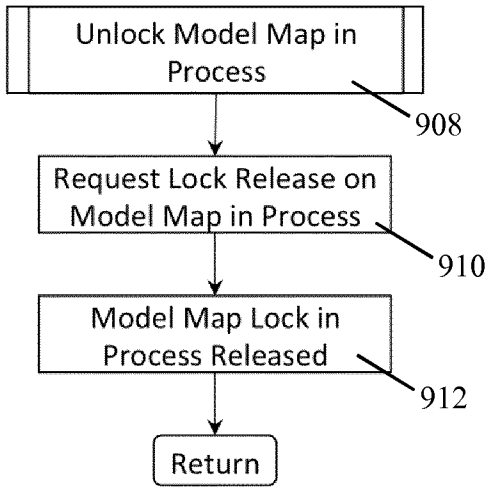

Fig. 9B

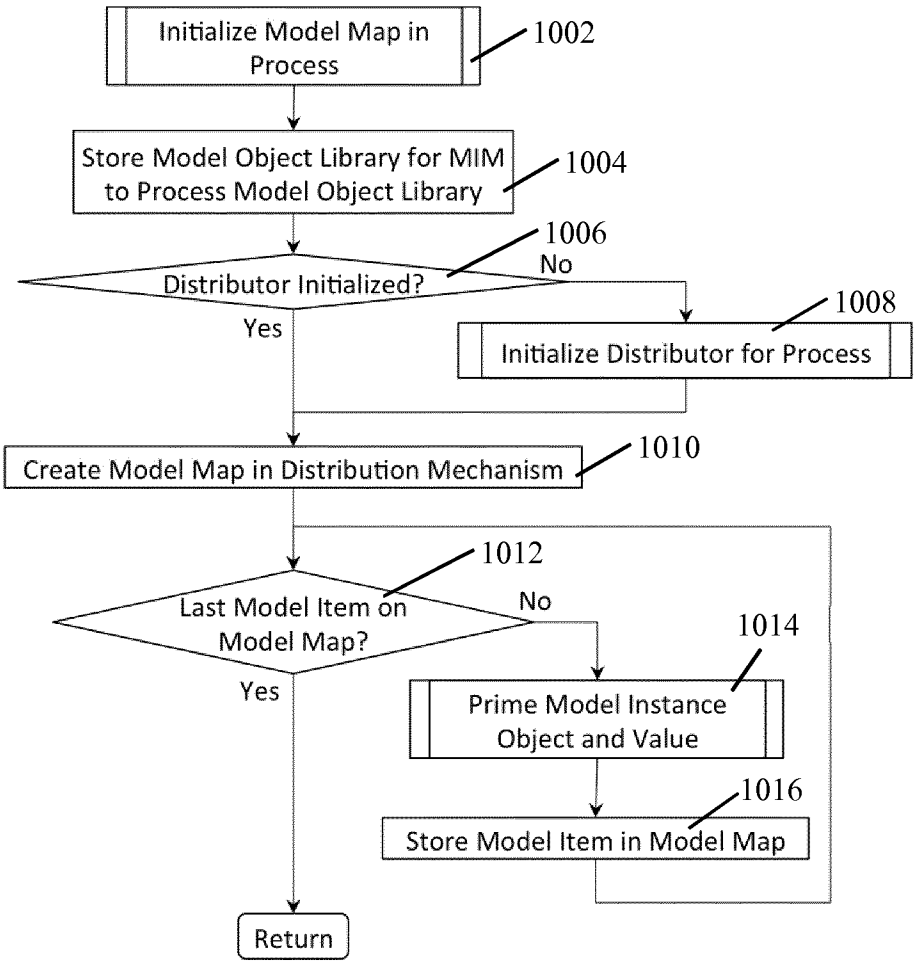

Fig. 10

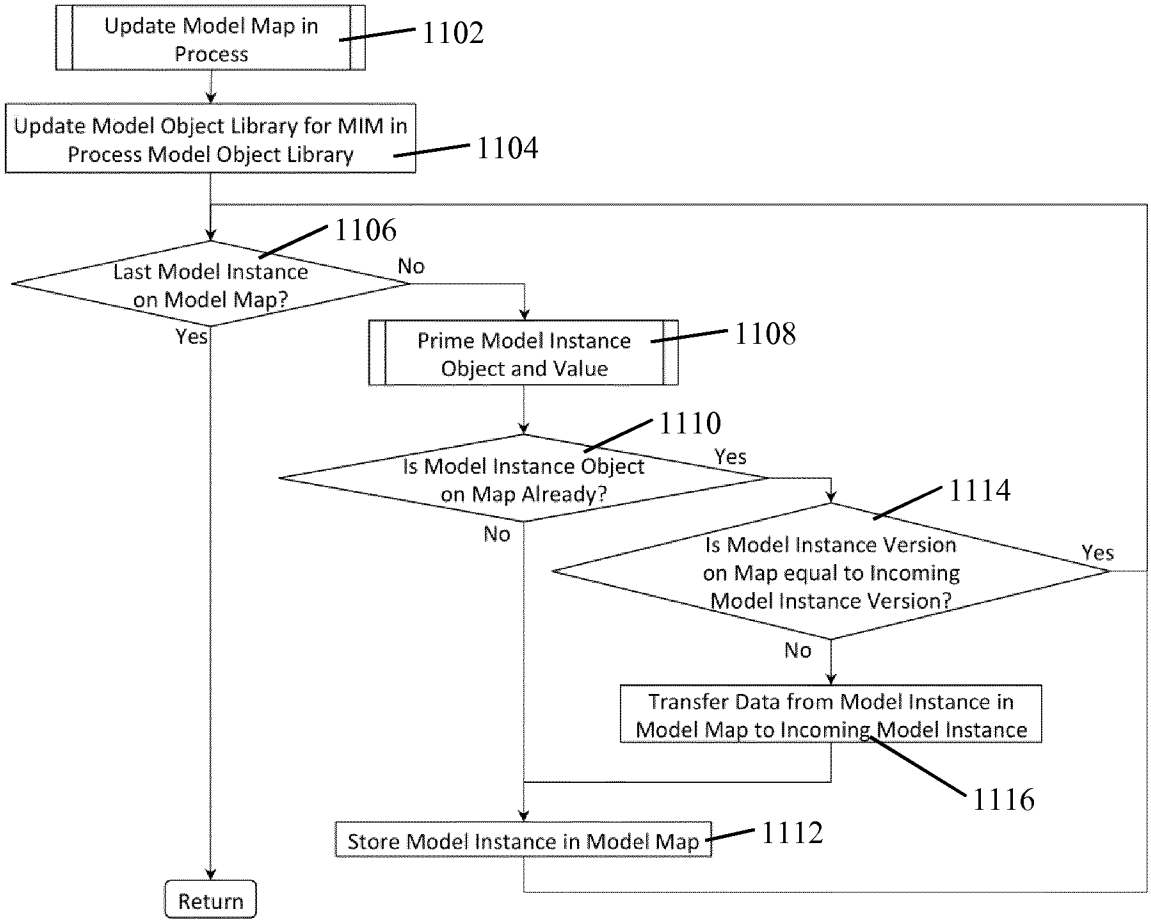
Update Model Map in Process
1102
Update Model Object Library for MIM in Process Model Object Library
1104
1106
Last Model Instance on Model Map?
No
Yes
Prime Model Instance Object and Value
1108
1110
Is Model Instance Object on Map Already?
Yes
No
1114
Is Model Instance Version on Map equal to Incoming Model Instance Version?
Yes
No
Transfer Data from Model Instance in Model Map to Incoming Model Instance
1116
Store Model Instance in Model Map
1112
Return

Fig. 11

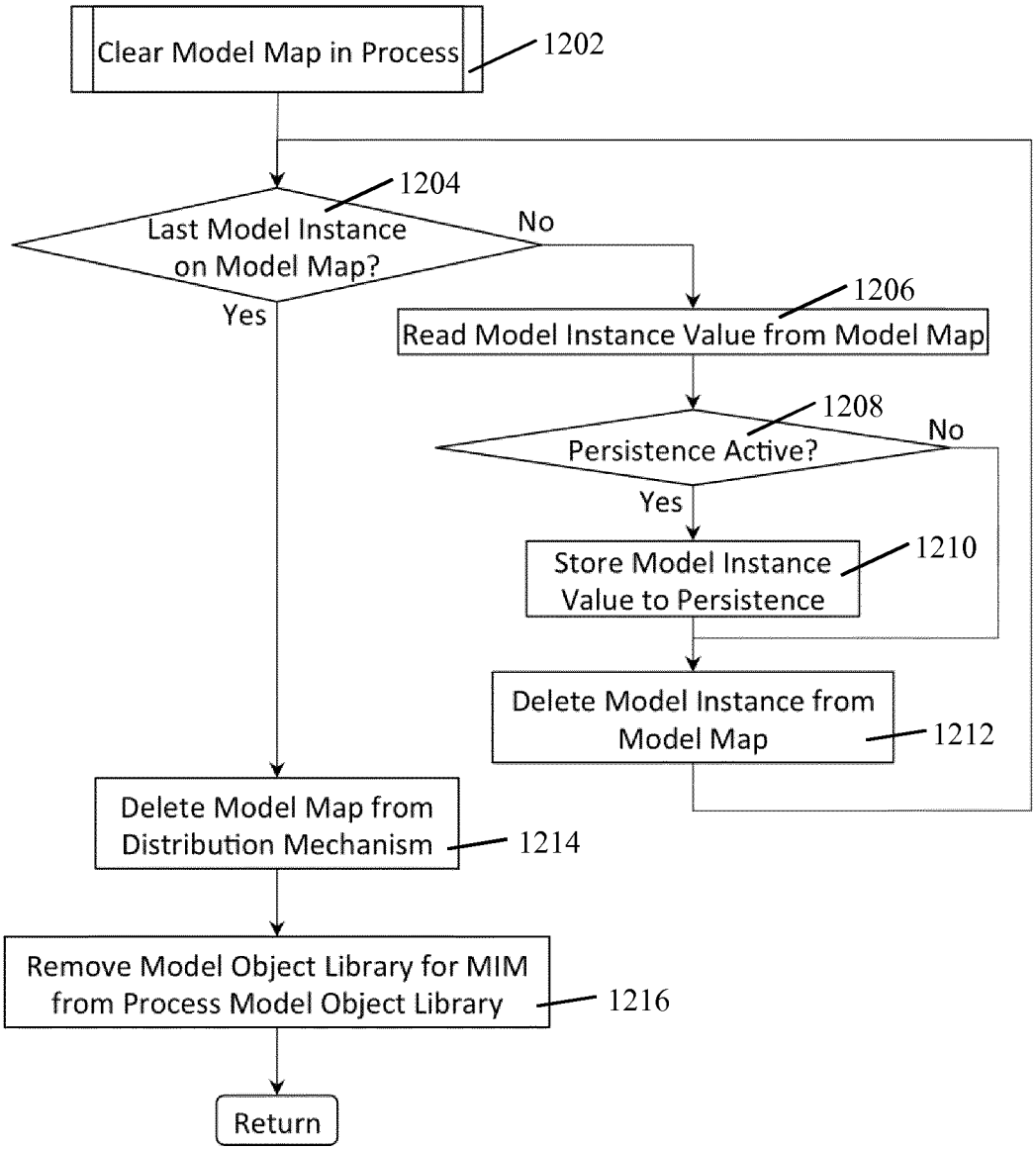
Clear Model Map in Process
1202
Last Model Instance on Model Map?
1204
No
Yes
Read Model Instance Value from Model Map
1206
Persistence Active?
1208
No
Yes
Store Model Instance Value to Persistence
1210
Delete Model Instance from Model Map
1212
Delete Model Map from Distribution Mechanism
1214
Remove Model Object Library for MIM from Process Model Object Library
1216
Return

Fig. 12

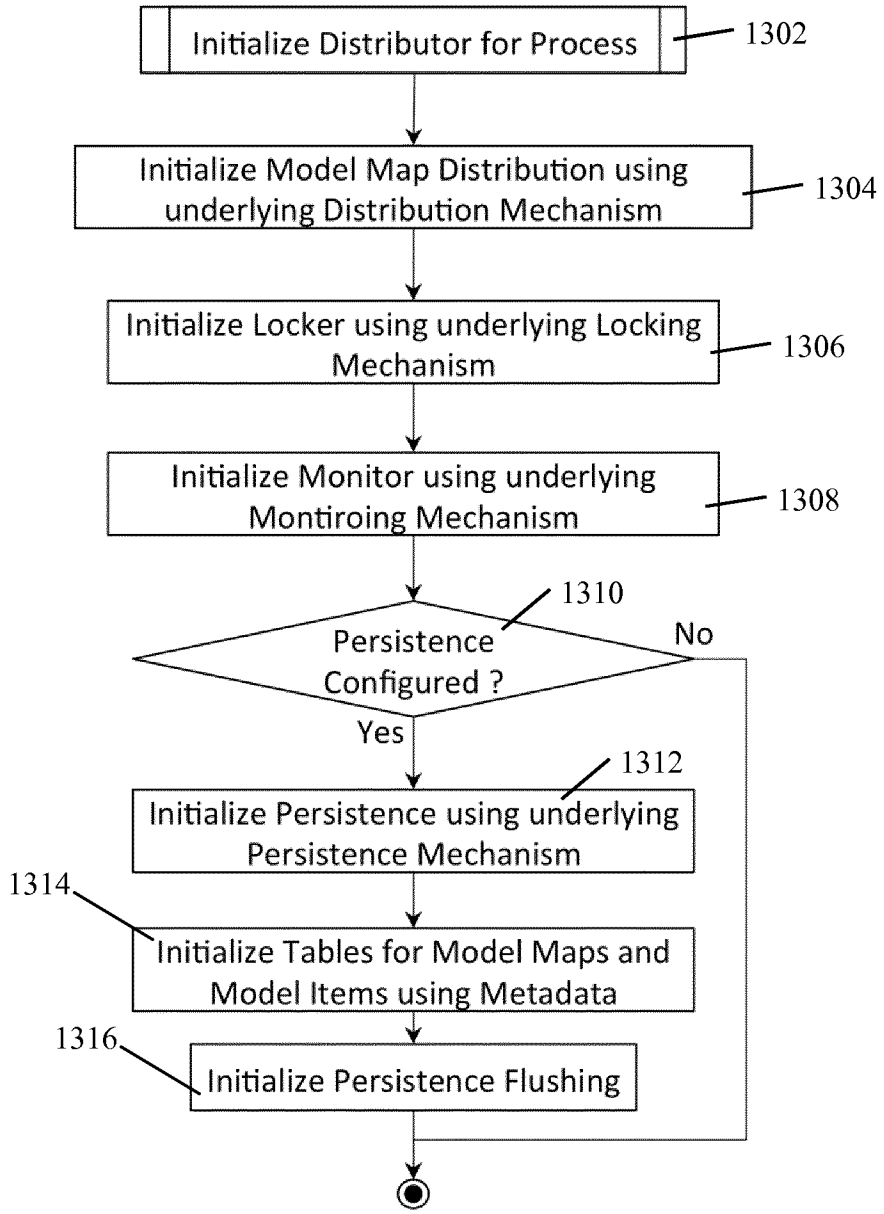
Initialize Distributor for Process
1302
Initialize Model Map Distribution using underlying Distribution Mechanism
1304
Initialize Locker using underlying Locking Mechanism
1306
Initialize Monitor using underlying Montiroing Mechanism
1308
1310
Persistence Configured ?
No
Yes
1312
Initialize Persistence using underlying Persistence Mechanism
1314
Initialize Tables for Model Maps and Model Items using Metadata
1316
Initialize Persistence Flushing

Fig. 13

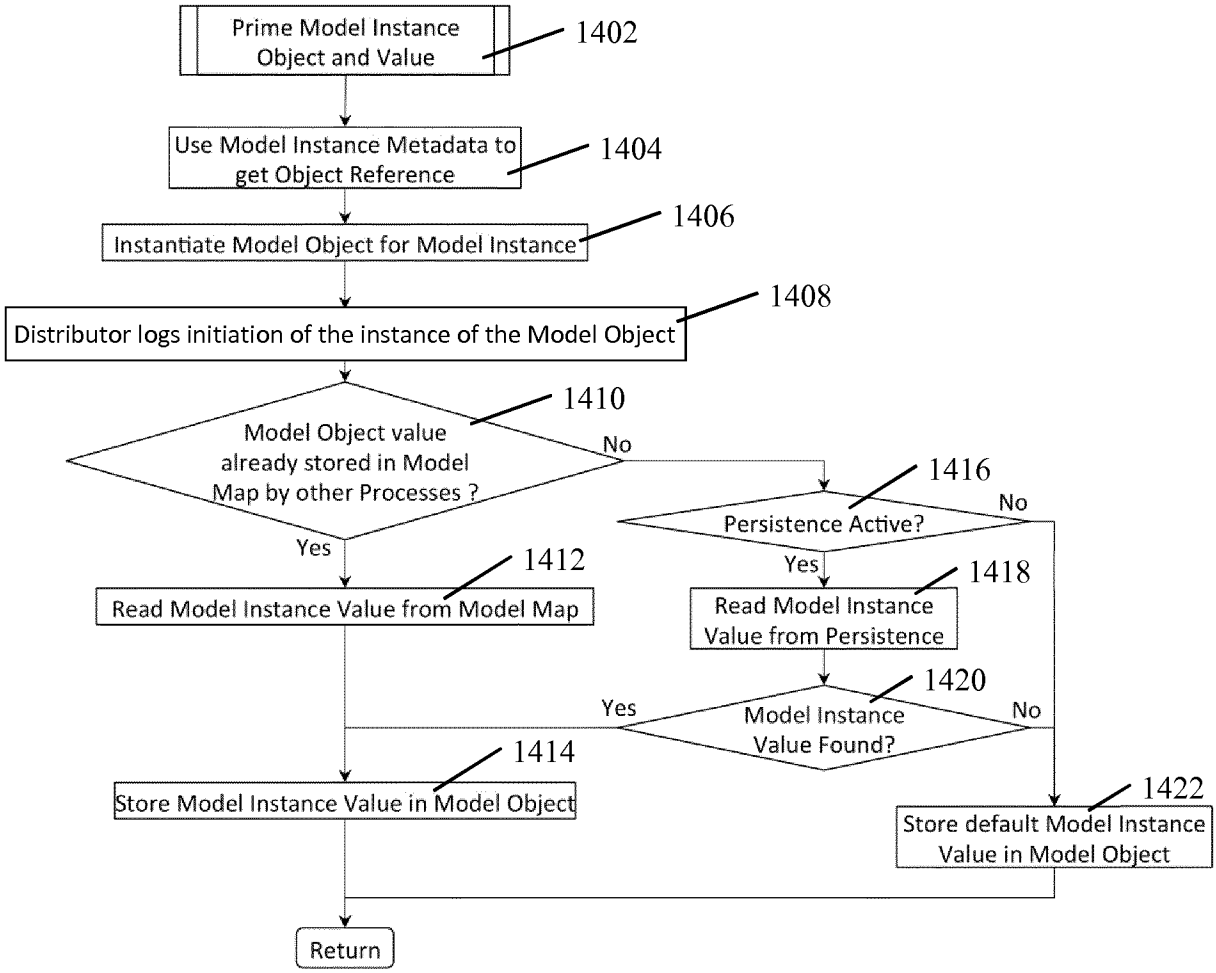
Prime Model Instance Object and Value
1402
Use Model Instance Metadata to get Object Reference
1404
Instantiate Model Object for Model Instance
1406
Distributor logs initiation of the instance of the Model Object
1408
Model Object value already stored in Model Map by other Processes ?
1410
No
Yes
Persistence Active?
1416
No
Yes
Read Model Instance Value from Model Map
1412
Read Model Instance Value from Persistence
1418
Model Instance Value Found?
1420
Yes
No
Store Model Instance Value in Model Object
1414
Store default Model Instance Value in Model Object
1422
Return

Fig. 14

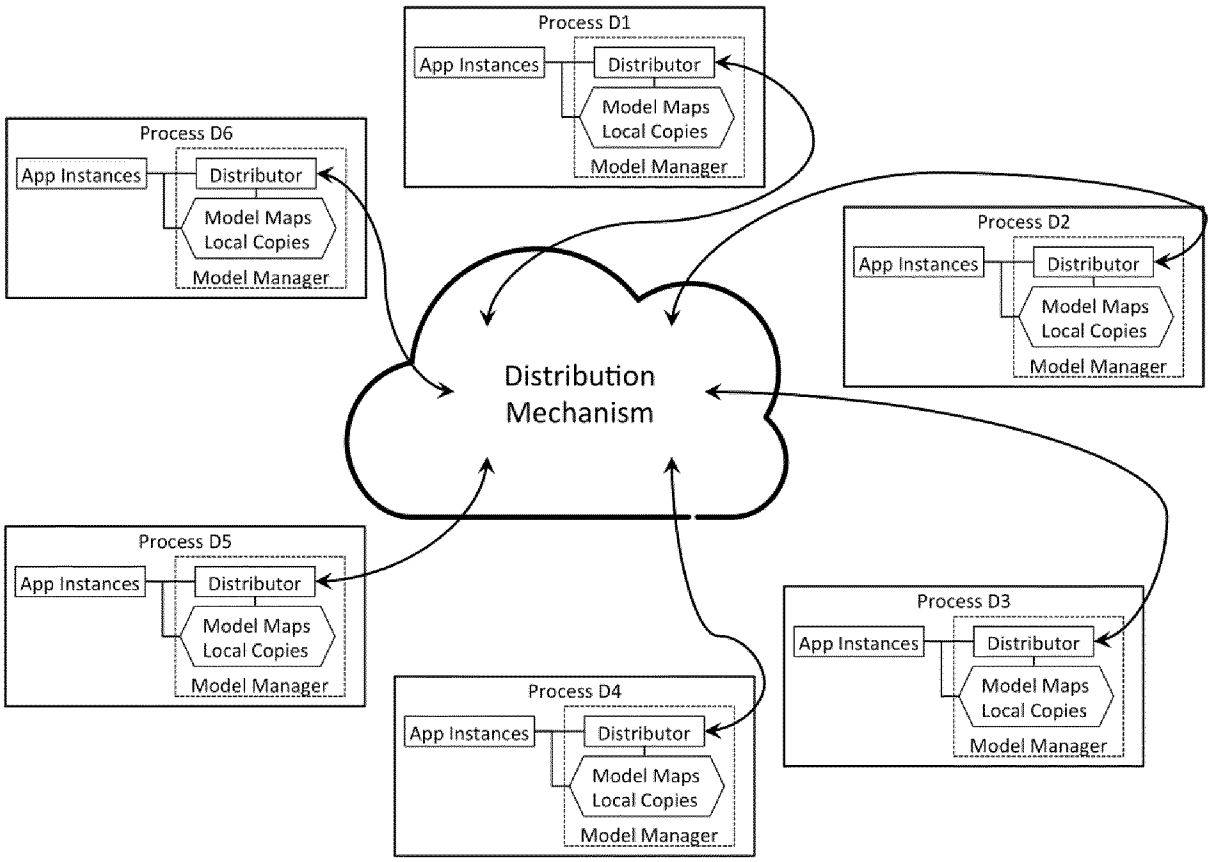
Process D1
App Instances
Distributor
Model Maps
Local Copies
Model Manager
Process D6
App Instances
Distributor
Model Maps
Local Copies
Model Manager
Process D2
App Instances
Distributor
Model Maps
Local Copies
Model Manager
Distribution
Mechanism
Process D5
App Instances
Distributor
Model Maps
Local Copies
Model Manager
Process D3
App Instances
Distributor
Model Maps
Local Copies
Model Manager
Process D4
App Instances
Distributor
Model Maps
Local Copies
Model Manager

Fig. 15

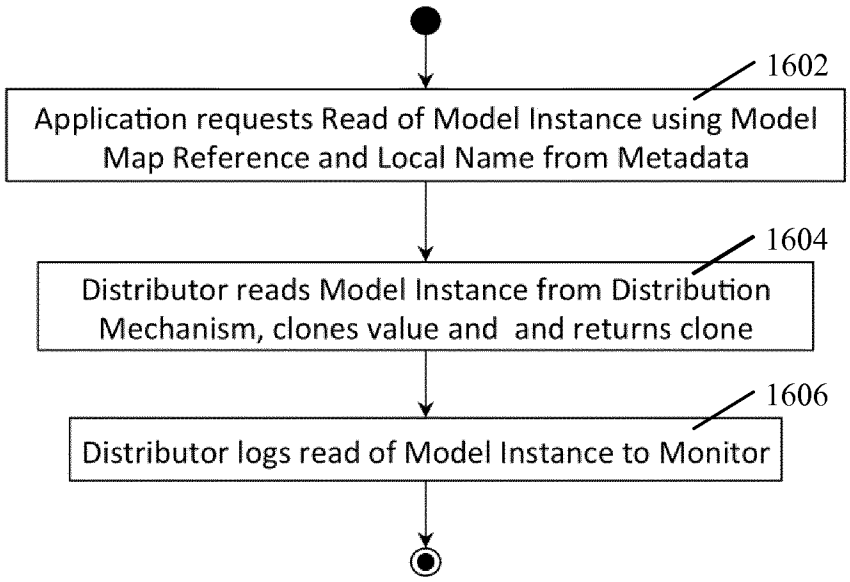
1602
Application requests Read of Model Instance using Model Map Reference and Local Name from Metadata
1604
Distributor reads Model Instance from Distribution Mechanism, clones value and and returns clone
1606
Distributor logs read of Model Instance to Monitor

Fig. 16

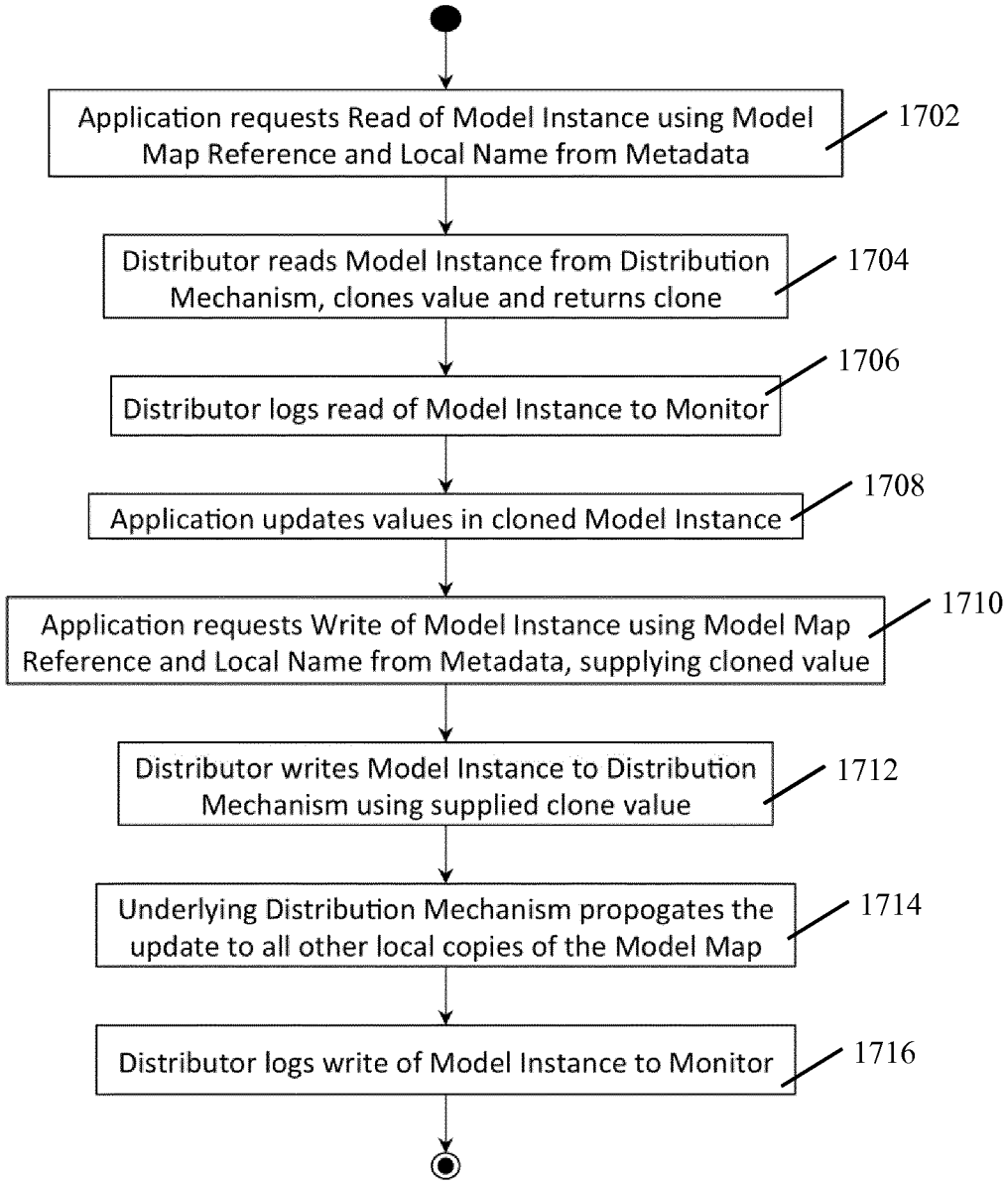
Application requests Read of Model Instance using Model Map Reference and Local Name from Metadata
1702
Distributor reads Model Instance from Distribution Mechanism, clones value and returns clone
1704
Distributor logs read of Model Instance to Monitor
1706
Application updates values in cloned Model Instance
1708
Application requests Write of Model Instance using Model Map Reference and Local Name from Metadata, supplying cloned value
1710
Distributor writes Model Instance to Distribution Mechanism using supplied clone value
1712
Underlying Distribution Mechanism propogates the update to all other local copies of the Model Map
1714
Distributor logs write of Model Instance to Monitor
1716

Fig. 17

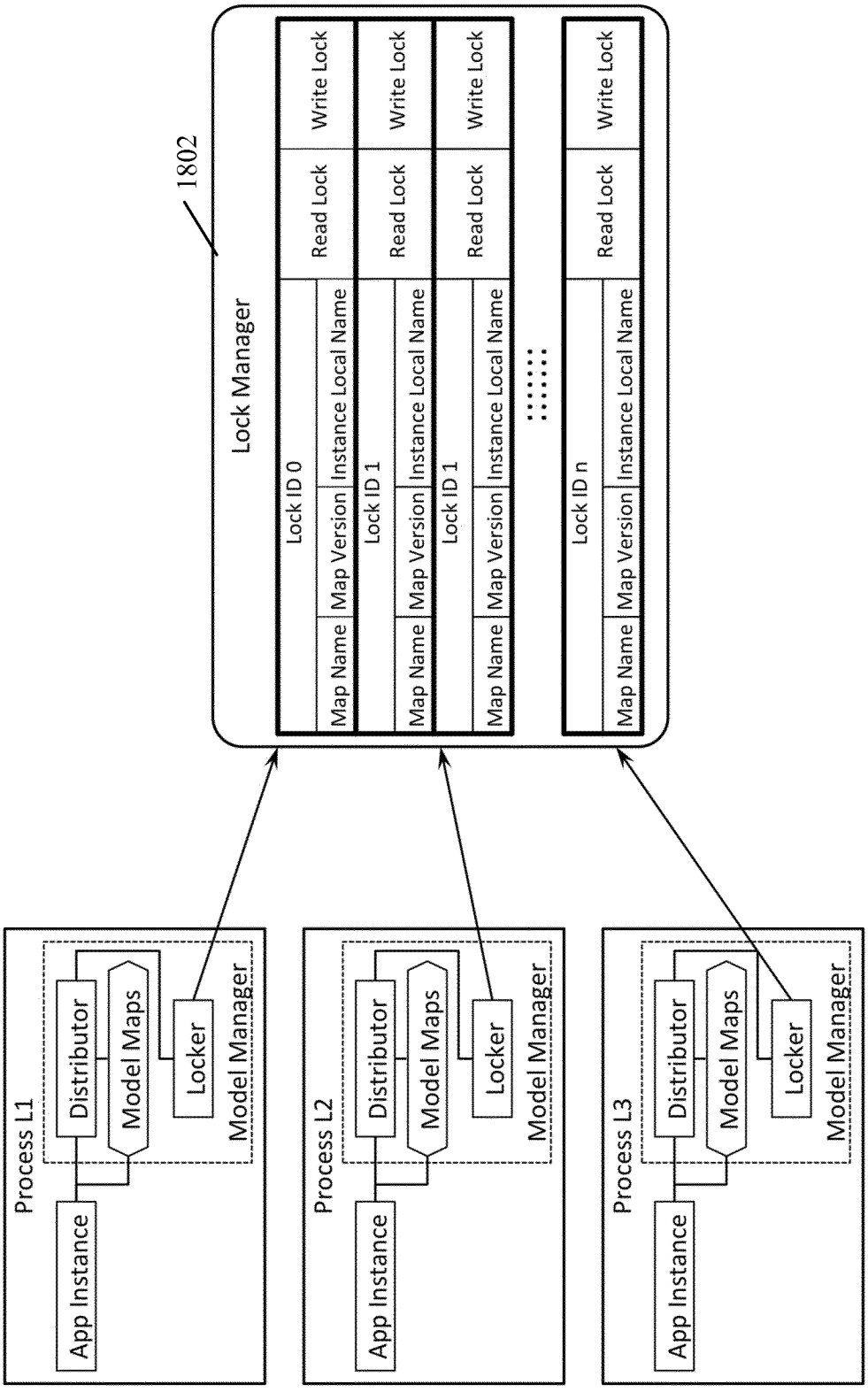
1802
Lock Manager
Lock ID 0
Map Name
Map Version
Instance Local Name
Read Lock
Write Lock
Lock ID 1
Map Name
Map Version
Instance Local Name
Read Lock
Write Lock
Lock ID 1
Map Name
Map Version
Instance Local Name
Read Lock
Write Lock
Lock ID n
Map Name
Map Version
Instance Local Name
Read Lock
Write Lock
Process L1
App Instance
Distributor
Model Maps
Locker
Model Manager
Process L2
App Instance
Distributor
Model Maps
Locker
Model Manager
Process L3
App Instance
Distributor
Model Maps
Locker
Model Manager

Fig. 18

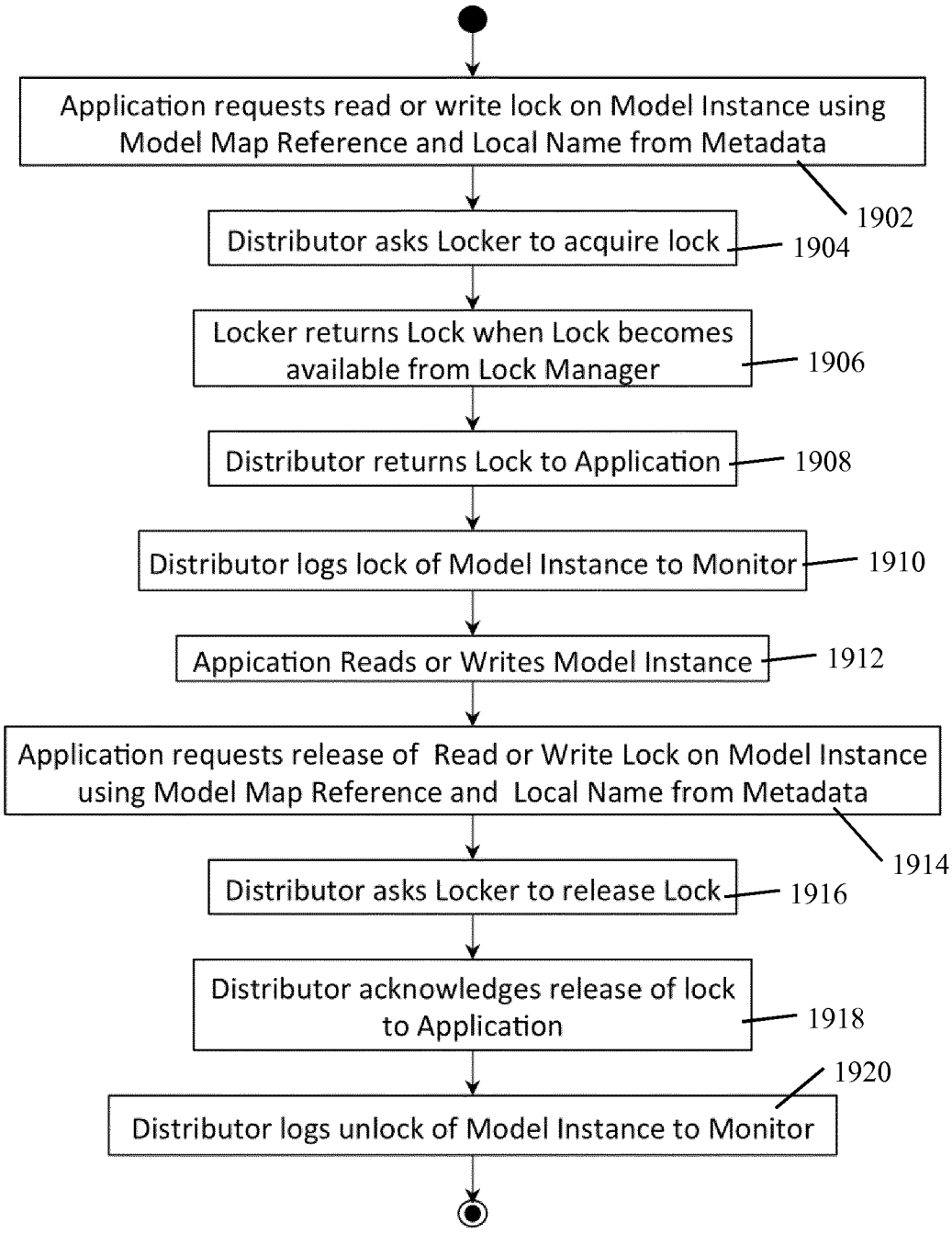
Application requests read or write lock on Model Instance using Model Map Reference and Local Name from Metadata
1902
Distributor asks Locker to acquire lock
1904
Locker returns Lock when Lock becomes available from Lock Manager
1906
Distributor returns Lock to Application
1908
Distributor logs lock of Model Instance to Monitor
1910
Appication Reads or Writes Model Instance
1912
Application requests release of Read or Write Lock on Model Instance using Model Map Reference and Local Name from Metadata
1914
Distributor asks Locker to release Lock
1916
Distributor acknowledges release of lock to Application
1918
1920
Distributor logs unlock of Model Instance to Monitor

Fig. 19

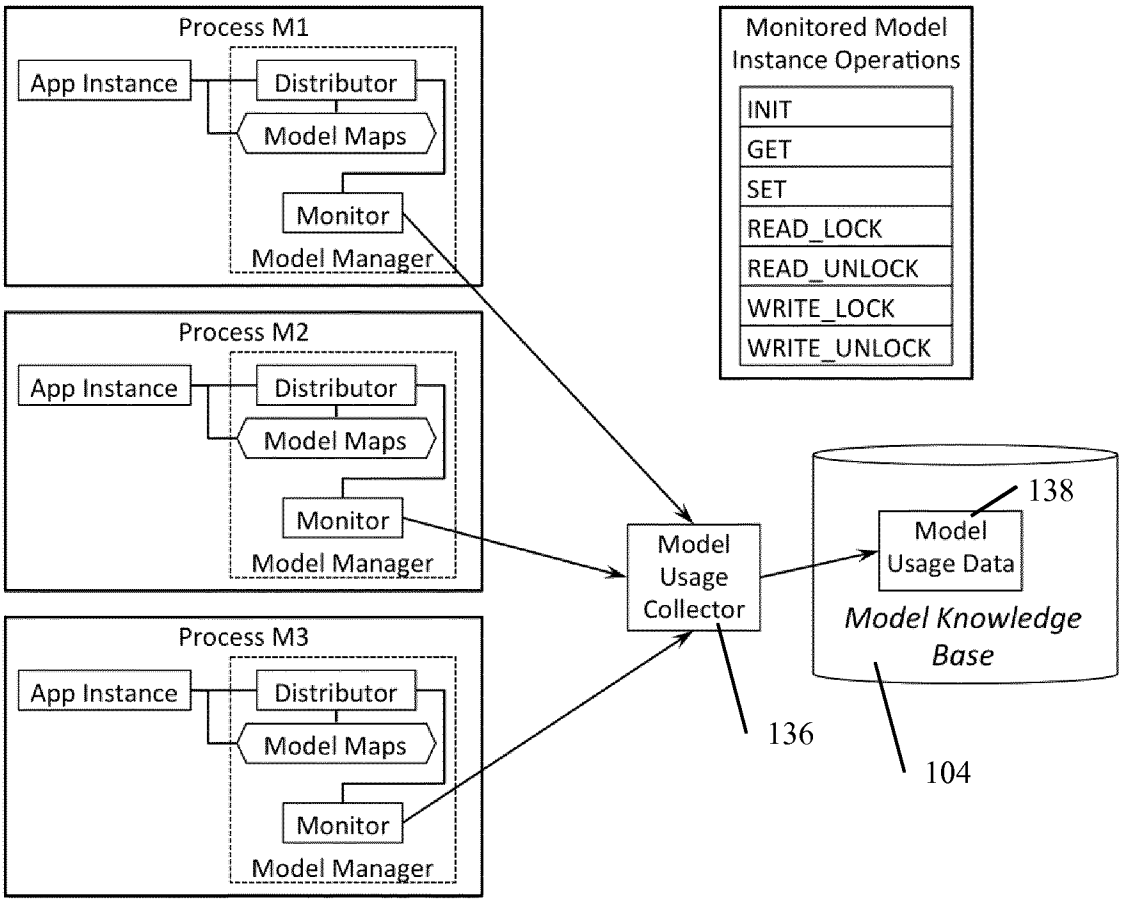
Process M1
App Instance
Distributor
Model Maps
Monitor
Model Manager
Process M2
App Instance
Distributor
Model Maps
Monitor
Model Manager
Process M3
App Instance
Distributor
Model Maps
Monitor
Model Manager
Monitored Model Instance Operations
INIT
GET
SET
READ_LOCK
READ_UNLOCK
WRITE_LOCK
WRITE_UNLOCK
Model Usage Collector
Model Usage Data
138
Model Knowledge Base
136
104

Fig. 20

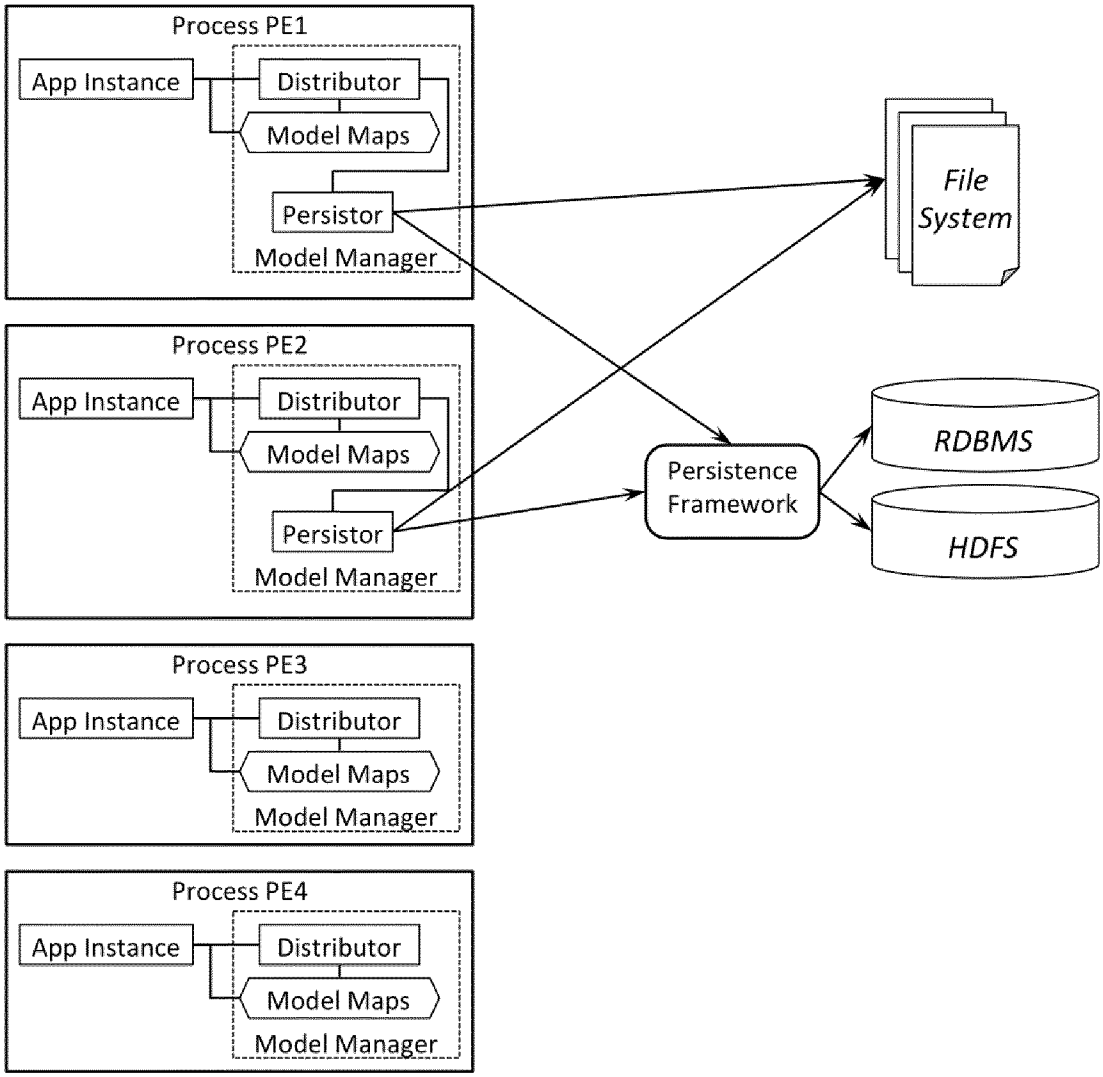
Process PE1
App Instance
Distributor
Model Maps
Persistor
Model Manager
Process PE2
App Instance
Distributor
Model Maps
Persistor
Model Manager
Process PE3
App Instance
Distributor
Model Maps
Model Manager
Process PE4
App Instance
Distributor
Model Maps
Model Manager
File System
Persistence Framework
RDBMS
HDFS

Fig. 21

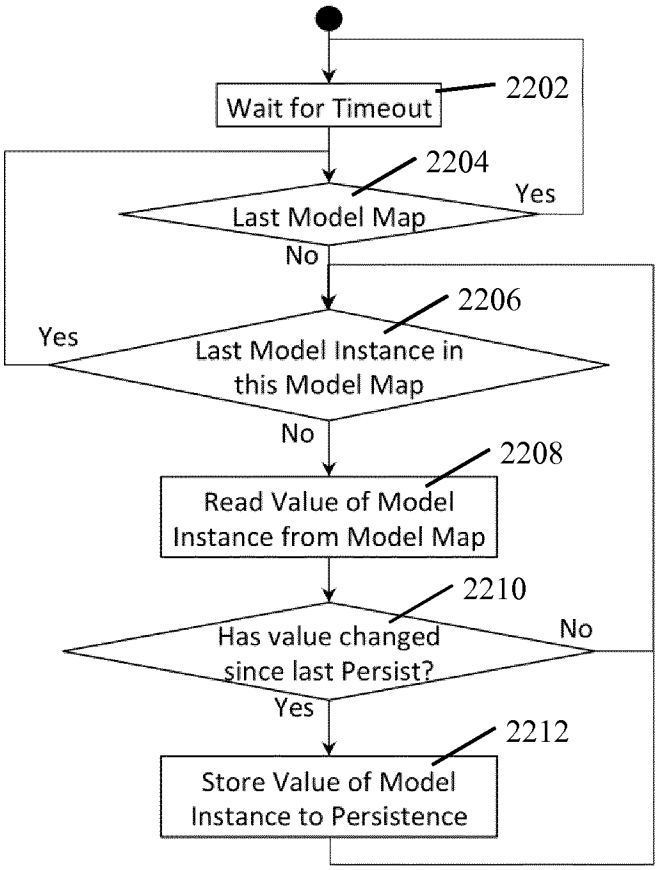
Wait for Timeout
2202
2204
Last Model Map
Yes
No
2206
Yes
Last Model Instance in this Model Map
No
2208
Read Value of Model Instance from Model Map
2210
Has value changed since last Persist?
No
Yes
2212
Store Value of Model Instance to Persistence

Fig. 22

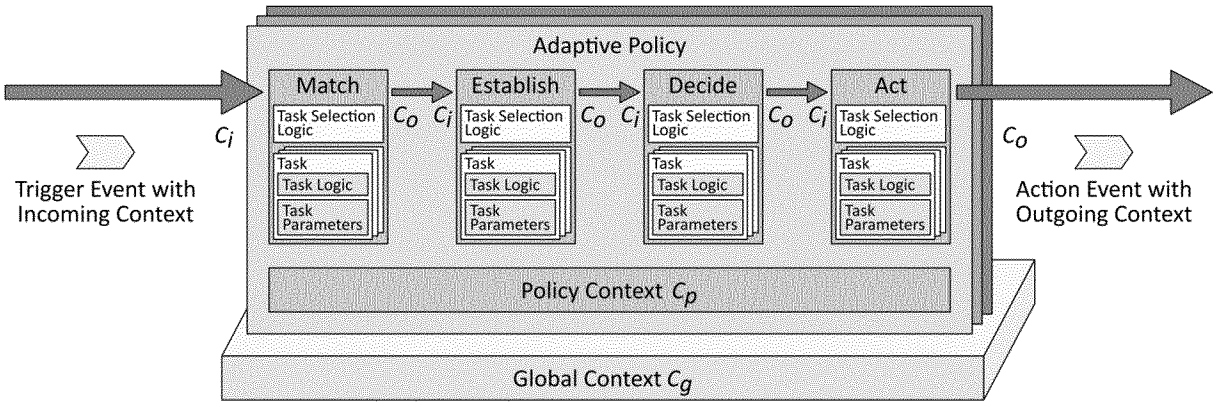
Adaptive Policy
Match
Establish
Decide
Act
Task Selection Logic
Task
Task Logic
Task Parameters
C_o
C_i
Policy Context C_p
Global Context C_g
Trigger Event with Incoming Context
Action Event with Outgoing Context

Fig. 23

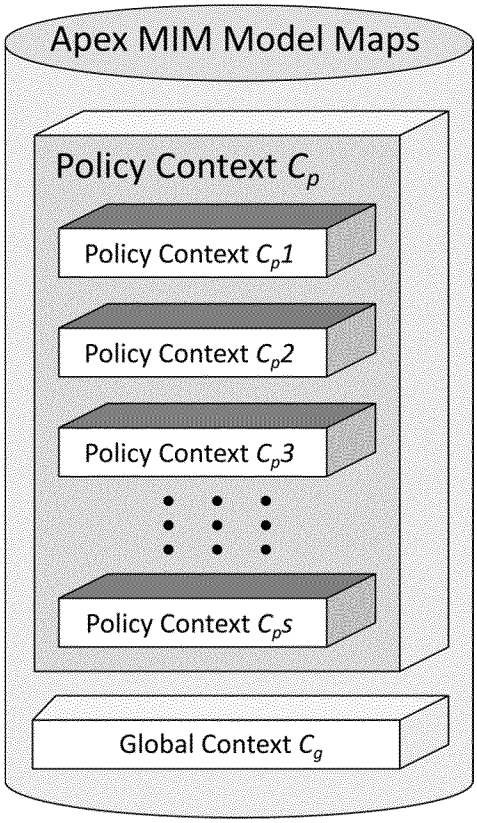
Apex MIM Model Maps
Policy Context C_p
Policy Context C_p1
Policy Context C_p2
Policy Context C_p3
Policy Context C_ps
Global Context C_g

Fig. 24

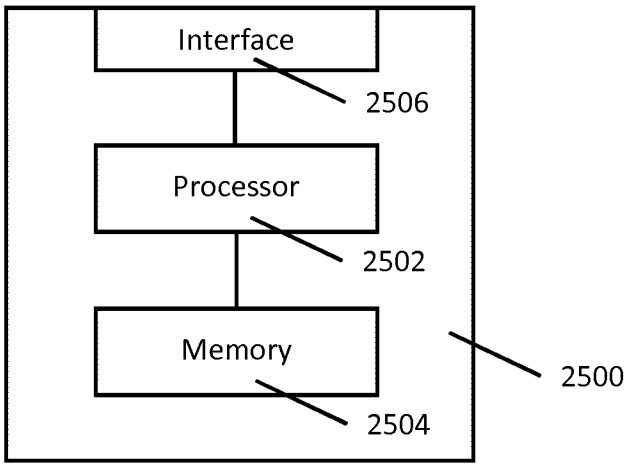
Interface
2506
Processor
2502
Memory
2504
2500

Fig. 25

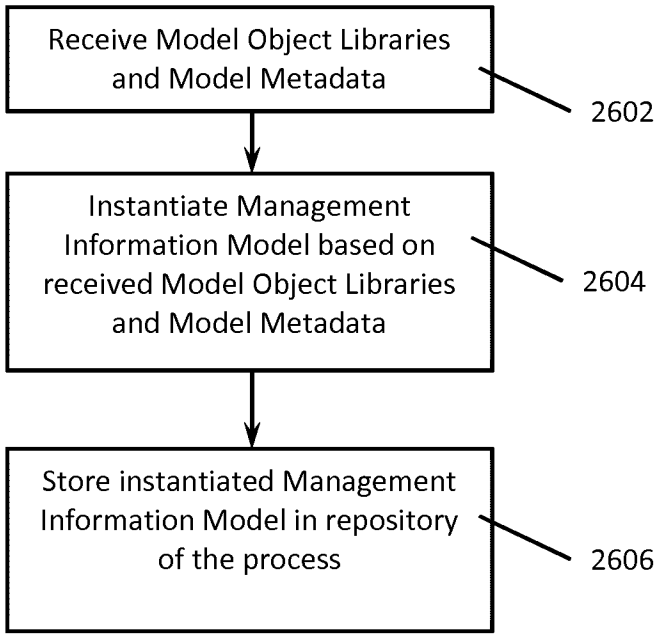
Receive Model Object Libraries and Model Metadata
2602
Instantiate Management Information Model based on received Model Object Libraries and Model Metadata
2604
Store instantiated Management Information Model in repository of the process
2606

Fig. 26

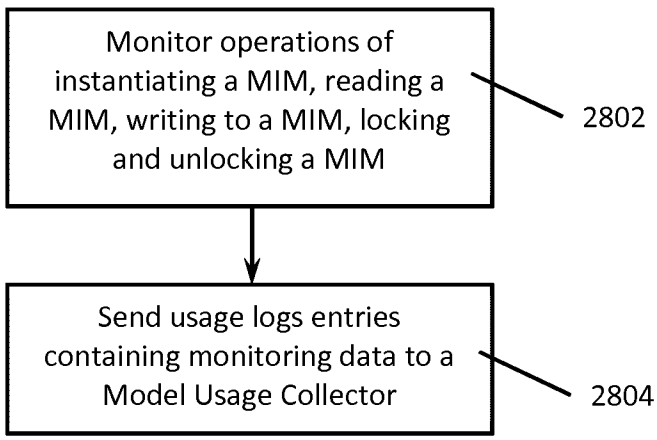
Monitor operations of instantiating a MIM, reading a MIM, writing to a MIM, locking and unlocking a MIM
2802
Send usage logs entries containing monitoring data to a Model Usage Collector
2804

Fig. 28

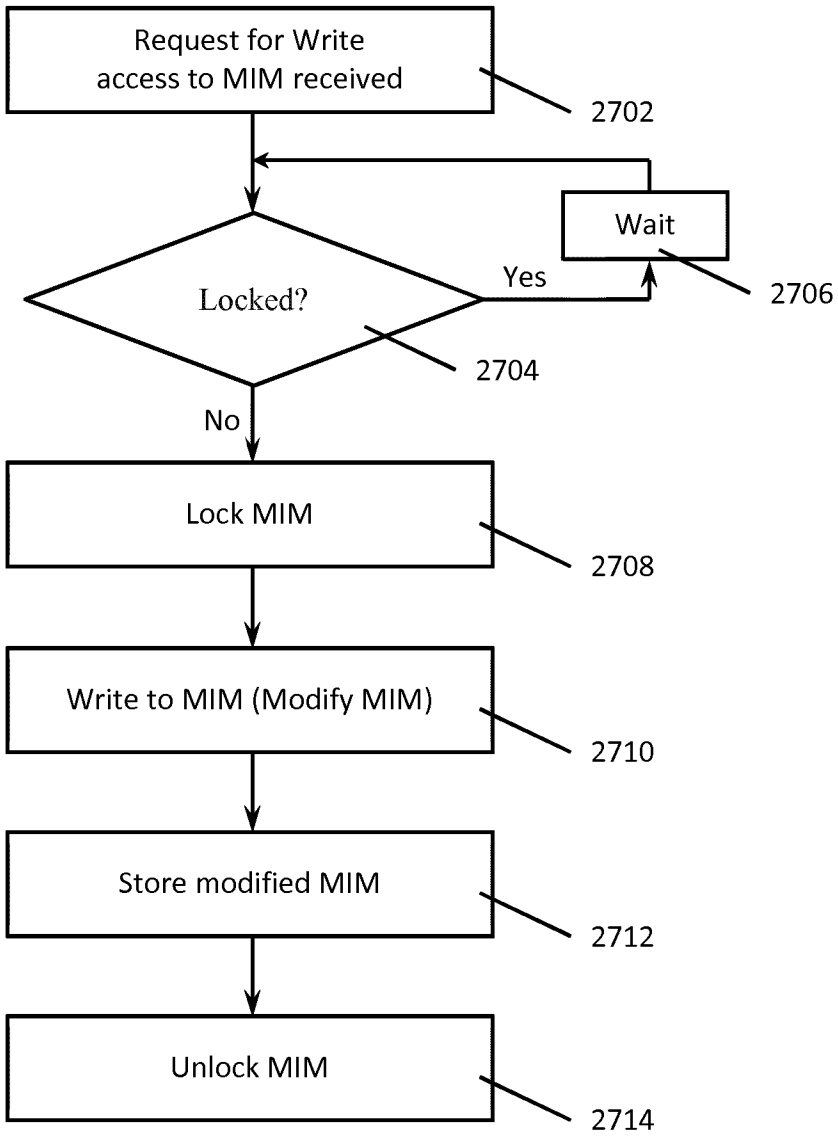
Request for Write access to MIM received
2702
Wait
Locked?
Yes
2706
2704
No
Lock MIM
2708
Write to MIM (Modify MIM)
2710
Store modified MIM
2712
Unlock MIM
2714

Fig. 27

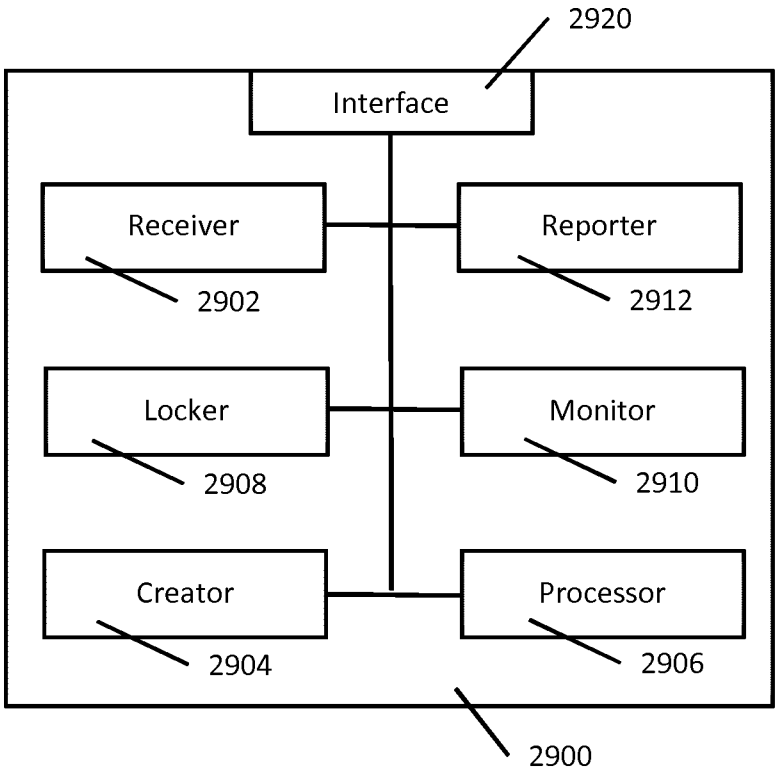
2920
Interface
Receiver
2902
Reporter
2912
Locker
2908
Monitor
2910
Creator
2904
Processor
2906
2900

Fig. 29

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2016/077180

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/52
ADD.

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)
G06F

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)
EPO-Internal, WPI Data, INSPEC, COMPENDEX

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	WO 2016/066438 A1 (ERICSSON TELEFON AB L M [SE]) 6 May 2016 (2016-05-06) abstract page 4, line 24 - page 5, line 26	1-24
Y	EP 1 843 259 A2 (COGNOS INC [CA]) 10 October 2007 (2007-10-10) abstract paragraph [0010] - paragraph [0027] paragraph [0124] - paragraph [0141] paragraph [0263] - paragraph [0276] figures 2,3 claims 1, 4	1-24
A	WO 2015/158377 A1 (ERICSSON TELEFON AB L M [SE]) 22 October 2015 (2015-10-22) the whole document	1-24

-/--

[X] Further documents are listed in the continuation of Box C.

[X] 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 but 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
14 March 2017	24/03/2017

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Beltrán-Escavy, José

1

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2014/088932 A1 (PIETZSCH TORSTEN [DE]) 27 March 2014 (2014-03-27) the whole document	1-24
A	US 2007/073877 A1 (BOYKIN JAMES R [US] ET AL) 29 March 2007 (2007-03-29) the whole document	1-24
A	US 2004/210607 A1 (MANCHANDA ARUN [CA] ET AL) 21 October 2004 (2004-10-21) the whole document	1-24
A	US 6 023 579 A (HELLGREN LARS VIKTOR [US] ET AL) 8 February 2000 (2000-02-08) the whole document	1-24
A	FALLON LIAM ET AL: "Apex: An engine for dynamic adaptive policy execution", NOMS 2016 - 2016 IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, IEEE, 25 April 2016 (2016-04-25), pages 699-702, XP032918172, DOI: 10.1109/NOMS.2016.7502880 [retrieved on 2016-06-30] the whole document	1-24
A	VAN DER MEER SVEN ET AL: "Dynamically adaptive policies for dynamically adaptive telecommunications networks", 2015 11TH INTERNATIONAL CONFERENCE ON NETWORK AND SERVICE MANAGEMENT (CNSM), IFIP, 9 November 2015 (2015-11-09), pages 182-186, XP032838999, DOI: 10.1109/CNSM.2015.7367357 [retrieved on 2015-12-28] the whole document	1-24

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2016/077180

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 2016066438 A1	06-05-2016	NONE	
EP 1843259 A2	10-10-2007	CA 2542379 A1	07-10-2007
		EP 1843259 A2	10-10-2007
		US 2007239769 A1	11-10-2007
WO 2015158377 A1	22-10-2015	CN 106464516 A	22-02-2017
		EP 3132567 A1	22-02-2017
		US 2017041181 A1	09-02-2017
		WO 2015158377 A1	22-10-2015
US 2014088932 A1	27-03-2014	EP 2711794 A1	26-03-2014
		JP 5675925 B2	25-02-2015
		JP 2014067417 A	17-04-2014
		US 2014088932 A1	27-03-2014
US 2007073877 A1	29-03-2007	CN 1921413 A	28-02-2007
		TW I392270 B	01-04-2013
		US 2007073877 A1	29-03-2007
US 2004210607 A1	21-10-2004	NONE	
US 6023579 A	08-02-2000	NONE	