A practical approach towards a distributed and flexible realization of policies using intelligent agents

René Wies, BMW AG, Corporate Network, 80788 Munich, Germany, Phone: +49-89-382-40098, Fax: +49-89-382-49145, Email: rene.wies@bmw.de

Maria-Athina Mountzia, Munich Network Management Team, Technical University of Munich, Oettingenstr. 67/D04, 80538 Munich, Germany, Phone: +49-89-21782164, Fax: +49-89-21782262, Email: mountzia@informatik.tu-muenchen.de

Philip Steenekamp, Telkom SA Limited, Pretoria, South Africa, Phone: +27-12-311-4355, Fax: +27-12-311-1734, Email: steenepj@telkom.co.za

Abstract

For the past decade it is a known fact that management of heterogeneous networks, distributed systems, and applications can only be done "computer aided". Ever since then, management agents and platforms have been deployed or installed throughout the IT infrastructure; on networking devices, systems, peripherals, and even on mobile end systems and in applications. Now that we network managers have complete (?!) control over our infrastructure, we see ourselves in a very similar position as system analysts were 15 years ago: managing a network is like programming in some assembly language. Aspects of automation, flexibility, and dynamics are rare!

In this paper we will very briefly introduce the concept of management policies as being derived from business goals to raise the level of abstraction for network, system, and application managers. Based on an architecture to realize management policies, we will outline the basic characteristics of the Intelligent Agent technology. The paper focuses on management policies but uses the concepts of mobile and intelligent agents to enable and facilitate the enforcement of high level management tasks and thereby providing for effective network and systems management. We will also present the first steps towards a prototype implementation using the IONA ORB (Orbix CORBA implementation) to realize the ideas and concepts described herein.

Keywords: *Network and Systems Management, Management, Policy, Intelligent Agent*

1 Introduction

1.1 Management Policies

The task of managing information technology resources becomes increasingly complex as managers must take heterogeneous systems, different networking technologies, and distributed applications into consideration. In order to deal with this complexity the notion of Management Policies has evolved during the last years [SLOM94, LUSL96]. Management policies are often seen as a link between corporate management and technology management. Thus, the word **policy** is surrounded by a vast number of other related terms, such as strategy, goal, vision, direction, mission, process, tactic, procedure, plan, scheme, course, and guideline, to name just a few.

The corporate mission leads to a number of long-term business strategies [PORT85] from which policies are derived. Policies define technical management measures that are specific for a particular department, such as the network and systems provisioning department. Hence, management policies are derived from the goals of management and related business strategies to define the desired behavior (i.e., technical characteristics) of distributed heterogeneous systems, networks, and applications. Management policies thus define **what** has to be accomplished by use of appropriate management tools.

In order to be implemented, policies must eventually lead (i.e., be transformed) to management scripts or rules that are interpreted by agents which can act on IT resources. Therefore, policies have to be converted into an algorithmic and functional representation of management goals and strategies, acting on a generic representation of resources supported by management agents. When specifying a corresponding and appropriate architecture to support the application of policies, it is obviously necessary to make use of existing standardized management architectures and management platforms, as far as these provide the necessary functions within their infrastructure. The aim of such a system is to aid an IT manager when specifying management policies and to guide IT operators in applying these policies. For this reason, such a system must be integrated in the management systems which administrators use on a regular basis. Such systems are frequently network and systems management platforms based on standardized frameworks [KASE94]. Section 2.1 will introduce the proposed policy architecture.

Until recently, it was believed that network and systems management applications will manage resources directly rather than invoking intelligent management services provided by the agents in the resources. This was because management information available in vendor-specific MIBs or MIFs is insufficient to do *real management*, they generally only provide heterogeneous monitoring information [BEYU93, HNGU94]. With the use of more advanced agents (such as RMON agents) and the concept of intelligent agents (see below) real automated and policy driven management can become reality.

1.2 Intelligent Agent Technology

Intelligent Agent Technology, as defined in [WOJE95], is another key technology in the area of distributed network and systems management. Similar to management policies, a large amount of research has so far been done on intelligent agents [GOYE95, MEBS95, FFMK92, FFMM93]. However, most of the work has focused on these two concepts in isolation, and very little research has been done to bring these two concepts together. [SLOM95, LUSL96, KOCH95] are the only other efforts known to the authors. In [SLOM96] intelligent agents are used to interpret obligation policies. The approach is based on a similar concept as the one presented in this paper. However, the advantages of our approach will be pointed out later on in the realization and implementation issues. It should be noted that we will make use of intelligent agent ideas and concepts formulated by the artificial intelligence community and apply those ideas in the area of network and systems management. We will not re-invent mature concepts, but rather re-use them for our purposes.

Intelligent agents can be described as pieces of autonomous software that act on behalf of a specific user or management application in various roles. Stephen Hawking once described viruses as being "... life forms because they satisfy all the criteria for being a life form - they react to stimuli, they reproduce, they consume.“ The same is valid for intelligent agents. Agents can also be classified according to their behavior into:

- **static agents** which are statically configured and cannot move around from one physical location to another,
- **dynamic agents** which can be extended in their functionality and can move from one location to another autonomously.

Examples of such agents are information gathering agents, email filtering agents, or service management agents. The aspects below will be used in the following as characteristic properties describing intelligent agents:

- **Autonomy:** agents operate without the direct intervention of humans or others and have some type of control over their actions and internal state.
- **Social ability:** agents interact with other agents (and possibly humans) via some kind of agent-communication language.
- **Reactivity:** agents perceive their environment and respond in a timely fashion to changes that occur in it.
- **Pro-activeness:** agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative thereby reacting to indicators rather than reacting to severe faults or problems as perceived by the user.
- **Flexibility and Dynamics:** agents are flexible in their reaction, i.e., their behavior depends on the particular environment and is not merely a deterministic set of actions. Flexible rule bases that can be updated in combination with inference engines usually form the basis of flexible and dynamic agents.

Based on these characteristics and the proposed policy architecture, we will outline how intelligent agents can be used to enable the architecture in Section 3.2. Section 4 will briefly characterize the prototype-implementation of the framework using the IONA ORB (Orbix CORBA implementation) that is still being refined, followed by a conclusion and outlook in Section 5.

2 Management Policies and Intelligent Agents

2.1 Policy Architecture

The Policy Architecture depicted in Figure 1 (see also [WIES95b,LEWI95], enables the implementation of systems management policies from a top-down perspective. This implies that all policies applicable to a specific domain have to be defined and transformed into low-level implementable policies as mentioned in Section 1.1. Management policies are defined, transformed, checked for conflicts, enforced and monitored by means of a Policy Support System.

The architecture can be implemented as a separate management application, for example on top of an open management platform such as HP OpenView or CA Unicenter. The most important modules in this architecture are the "Definition, Transformation, and Conflict Resolution Module“, the "Enforcement Module“ and the "Monitoring Module“. The end product of the policy transformation and conflict resolution are management scripts which define how existing management tools, agents, functions, or services are to be used to enforce and monitor the policy.

However, it should be noted that policies are very dynamic, both in terms of the goals they must enforce and the methods (management functions) they may use to achieve their goals. Hence, the enforcement of policies is not something one can define or derive once and code in some programming language. Policy enforcement is dynamic in that it responds to changes in the environment and is interdependent with other policies acting on the same or related resources. For example, keeping software versions of business applications (e.g., MS-Word, Excel, etc.) up to date by automatic software distribution can be realized by distribution lists in different software servers. However, permanent changes within the environment requires frequent updates of these distribution lists. The deployment of intelligent agents in the environment can ease this tedious process of interpreting and enforcing policies specifying when and where to distribute distribution lists to. The previous implementation of the policy architecture is not distributed [WIES95a], policies are not interpreted and thus require recompilation in order to be changed. This paper addresses exactly these points by applying the concept of intelligent agents for the distributed and flexible realization of the policy architecture avoiding policy conflicts and utilizing the "flexibility" of intelligent agents.

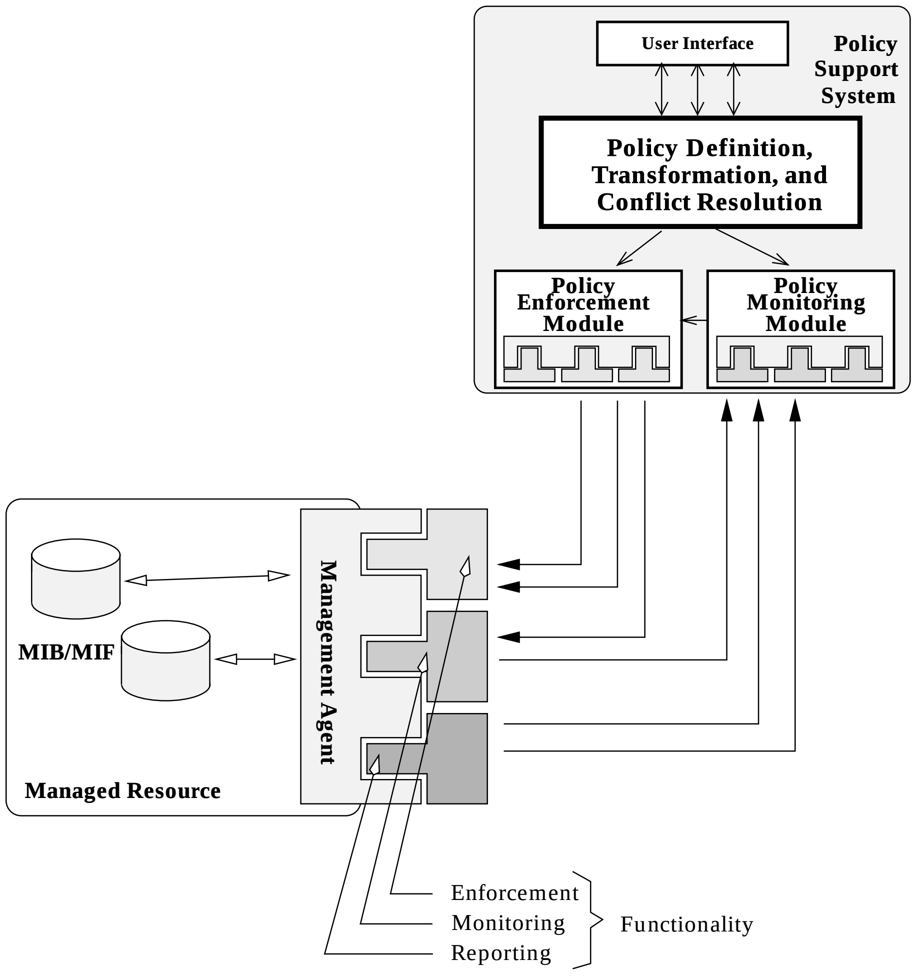

Figure 1: A simplified View of the Policy Architecture

In the above example, the intelligent agents identify the need for a new version of software based on the date the user last used the specific software product. If the user has not used MS-Word for more than four months, a new version is not installed on the user’s workstation in advance, but only on demand. This scenario can be enforced by intelligent agents. Depending on the frequency of usage, the agent could also automatically decide to choose a fixed license for the new version of the product rather than a floating license, (i.e., if the software product is not used very extensively, there is no need for an expensive floating license and a fixed license will be sufficient.) This policy at a lower and interpretable level of specification could have the following format (see also [NEWI96]:

```
if (applicationSW.name == "Word“ .and. applicationSW.version == "7.0“
   ∧ applicationSW.request_license("Word“,“7.0“) == true)
then
         if (applicationSW.frequency_of_use("Word“,“7.0“) <= 5 (* usage per month *)
            ∨ applicationSW.frequency_of_use("Word","7.0") == not_used_in_last_month)
```

```
        then
                (applicationSW.allocate_license("Word“,“7.0“)->number :=
                 applicationSW.allocate_license("Word“,“7.0“)->alloc_table(IP_Adr,u1);
                 applicationSW.allocate_license("Word“,“7.0“)->user := u1;
                 applicationSW.allocate_license("Word“,“7.0“)->type := "fixed“;)
        else
                (applicationSW.allocate_license("Word“,“7.0“)->number :=
                 applicationSW.allocate_license("Word“,“7.0“)->allocate_float(next_free);
                 applicationSW.allocate_license("Word“,“7.0“)->user := u1;
                 applicationSW.allocate_license("Word“,“7.0“)->type := "float“;)
        end;
end;
```

If necessary, the existing management agent may be extended with policy specific functionality such as engines to interpret enforcement or monitoring scripts. By making use of the idea of "management by delegation" [GOLD96] the load on the manager is transferred to the managed systems.

A different example, where feature interaction [POAT96, BRAT94] of agents can also be used is the following: an intelligent agent (e.g., Microsoft’s SMS agent, Tally’s NetSensus agent) detects that a new machine has been introduced in the distributed system environment. The feature of automatically detecting a system leads to a delegation of configuration tasks to other agents in the network, such as for updating software on the other systems, reconfiguring the firewall, informing the name and directory server, etc.

2.2 Intelligent Agent Framework

Centralized realization of management applications reveals a number of problems that indicate the need for distributed and flexible realization of existing management practices. In order to reduce network traffic and to increase the reliability and effectiveness of the management system, a hierarchical, distributed management architecture is necessary. Such an approach requires distribution of functionality across the managed environment.

The same applies to management policies. The realization of systems responsible for the enforcement and monitoring of policies requires a distributed approach as well. This distribution is imposed by the same reasons that impose distribution in most (management) applications. The organization of different domains with different types of policies applying to them is another argument towards a distributed realization.

Besides this, due to the dynamic nature of management policies, it is necessary for the system to be dynamically adjustable to new requirements. This includes new policies resulting from requirements that could not have been anticipated during the management system design phase. The realization of these policies should be done in the most effective way regarding resources usage, speed, traffic, and cost. As a result of the dynamic nature of policies, we have the need to update monitoring and enforcement rules to adapt to new requirements.

Delegation of functionality from a manager to selected management entities in the distributed system enables hierarchical, distributed management. Dynamic delegation allows a management system to be adjusted to changes and/or new requirements at system runtime, i.e., be flexible. Dynamic delegation of functionality can be realized with the Management by Delegation paradigm [MEBS95, GOYE95] or language-based mobile agents. The dynamic approach involves delegation of functionality at management application operational phase. This means that the functional parts to be assigned to the remote agents are sent there while the systems are running and are dynamically incorporated to the agents’ normal operation.

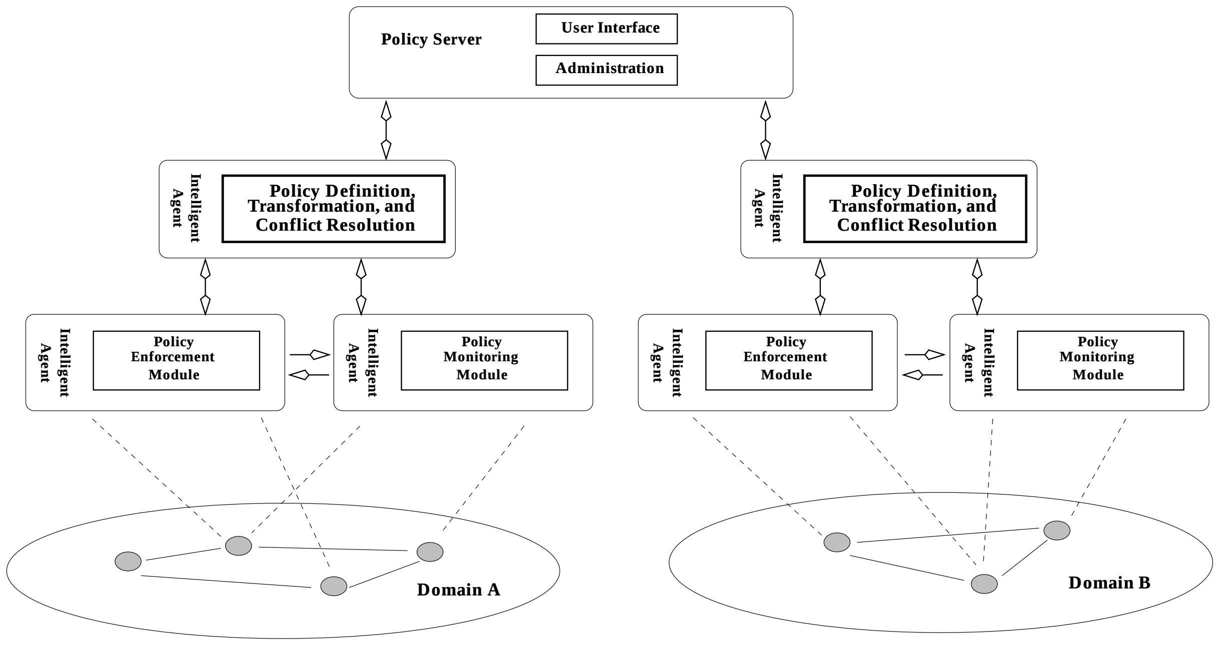

Figure 2: A distributed Policy Architecture

The concept of Intelligent Agents and specifically their flexibility in terms of their ability to dynamically expand their functionality allows the hierarchical, flexible realization of management policies. The policy architecture presented in the previous section allows the implementation of such a policy support system. However, applying the concept of IAs within this architecture requires a number of additions to the management infrastructure.

First of all we need to decide which parts of the policy support system can be delegated to the managed systems. From the building blocks of the policy support system, it is obvious that the Policy Enforcement and the Policy Monitoring modules are good candidates for delegation. Therefore we could assign those tasks to IAs for each domain. The number of IAs used depends on the number and types of policies respectively. The same could be done for the Policy Definition, Transformation and Conflict Resolution Module. An IA could undertake this task for each domain taking care of domain specific properties such as available agents, monitoring or metering tools, etc.

The decision of **what** to delegate brings up the question of **where** to delegate functionality to. A precondition for the execution of the delegated functionality is that the managed systems have the infrastructure to receive and execute it. The new notion introduced as a result of the delegation is the **flexible agent**. This is an agent whose functionality can be dynamically enhanced with new management functionality and can communicate not only with the manager but also with other management agents (e.g. SNMP agents or other flexible agents). A flexible agent consists of the delegated functionality and the execution engine for it. Agents possessing the above properties incorporate a specific form of intelligence. An overview of these agents and an analysis of the resulting requirements on management architectures is given in [MODR96]. In this way we end up with the architecture depicted in Figure 2.

For every domain there is one IA for Policy Definition, Transformation and Conflict Resolution, and a number of IAs with the functionality of the Policy Enforcement and Policy Monitoring Modules respectively. We will call the IAs for the realization of the Enforcement and Monitoring Modules PEAs and PMAs correspondingly. This enables the parallel execution of these in the different domains.

The decision to split the Policy Enforcement and Policy Monitoring Modules to the PEAs and PMAs (instead of having just one performing both tasks) is due to the different type of functionality they incorporate and the fact that this way we have a better structure of the system. The same PEAs can be used by many other PMAs and the corresponding IAs are less bulky. This is a modular approach that allows the easy accommodation of changes. Since a PEA might be used by many different PMAs it makes sense to keep them separated and "shared" among the different PMAs. They are reusable and parameterisable. If a PEA that is used from many different PMAS changes, then we all need to change that and the rest of the system remains the same. Otherwise, if we had kept them together then all corresponding PMAs would have to be updated.

PMAs need a mechanism to know which PEAs to call for the enforcement of the policies and communicate with them.

New requirements imposed on the system - reflected to new policies - can then be dynamically sent to the IAs. This enables the creation of new and/or update of existing Policy Enforcement and Policy Monitoring rules as well as Definition, Transformation and Conflict Resolution rules. These rules are handled by an execution engine in each IA. The IA architecture to meet the above requirements is depicted in Figure 3.

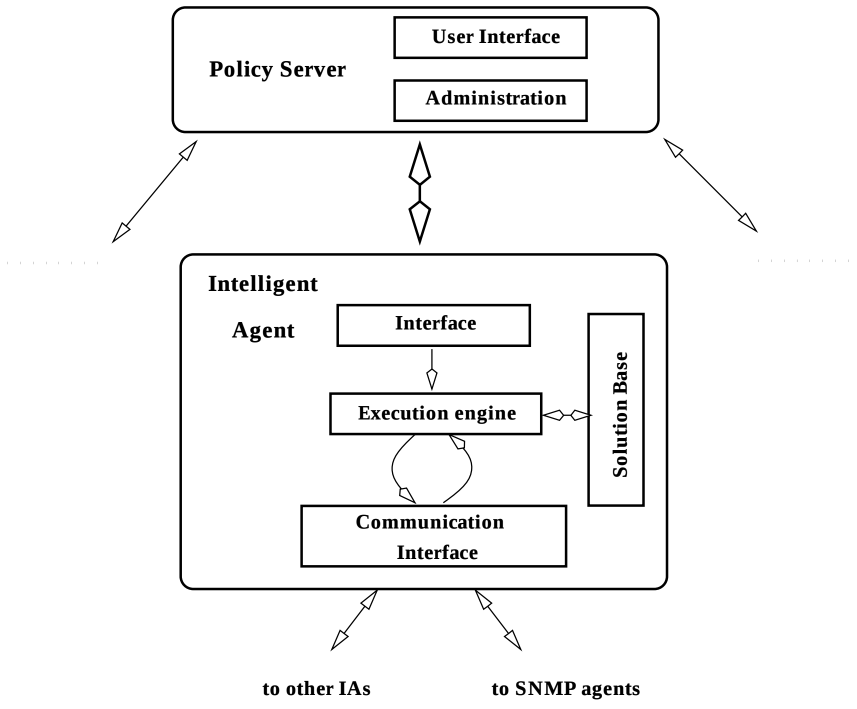

Figure 3: Intelligent Agent Architecture

The distributed realization of the management policies imposes new requirements on the policy server too. Due to the fact that the policy Monitoring and Enforcement rules are dynamically distributed among the PEAs and the PMAs the policy server needs to take the following into account:

- Which policies are transformed, enforced, and monitored by which IAs. This requires an administration unit on the manager for the delegated functions and for the handling of results (as shown in Figure 2 and Figure 3). The information kept in the administration unit consists of the functions that have been delegated, where they have been sent and what their status is (i.e., running, suspended). If there are more systems involved in the delegation, which is usually the case for different domains, then the usage of one or more traders, keeping information on which manager has delegated what type of functionality to which flexible agent, seems to be more appropriate.

- The manager should also possess the ability to remotely control the execution of the delegated functions. That is remotely suspend, abort, stop, reinitiate a policy.
- Conflicts among policies realized by different IAs. For this purpose we would need a global Conflict Resolution Module to handle inter-domain conflicting policies.
- Dependencies on management policies. If something changes on the monitoring rules for one IA, how does it affect the others or the enforcement rules?

After the functionality has been distributed in the system and when more than one systems are involved, there is the question of how they communicate with each other and how the overall operation of the system is coordinated. How do PMAs know where the correct PEAs they need to invoke for the enforcement of a specific policy are located? For this purpose the trader concept is also applicable. The PEAs and PMAs export their services to the trader and they can be used by other IAs.

2.3 Realization of the framework

So far in this paper we took a top-down approach for the development of a model enabling a flexible and distributed realization of management policies. This consists of a number of flexible agents that are responsible for the enforcement and monitoring of policies in the different domains. The corresponding requirements to the policy architecture have been specified along with the architecture of the agents and the communication models.

The framework in [STRO96] provides a configuration server and policy server for domain and policy definition, implementation and maintenance purposes. This mapping of the policy architecture to the framework applying intelligent agents is outlined in Figure 4. The Configuration Server operates from a domain perspective. The environment is configured in terms of domains with a set of "intelligent“ agents implementing the policies applicable to that domain. The Policy Server is used to control policies in the domains it is responsible for. The Policy Server will notify an IA when the policies, attached to the domain it is responsible for, change.

The functionality provided by the modules (Policy Enforcement Module, Policy Monitoring Module and Policy Definition, Transformation and Conflict Resolution Module) is distributed to the IA-level of operation. The Policy Server will ensure that the policies are distributed to the correct IAs to be refined and checked for conflicts. This will happen in a dynamic fashion, when the IA is ready for an update of its policies.

The existing monitoring functionality of the managed resources can be used by intelligent agents to monitor the policies enforced upon a certain domain's Managed Objects (MOs). In [STRO96] the objects that provide functionality to be used by intelligent agents to enforce or monitor policies are called Solution Service Object (SSO). There are two classes of SSOs, one for monitoring and one for enforcement. In the current implementation the two classes were realised in one object. When the functionality gets more complex, they can be separated. The Ias will obtain access to the SSOs via a Directory Server and a Trader. The SSOs will register themselves at the Trader. A reference to the applicable SSO for a specific policy will be provided by the Directory Server.

For the implementation of this architecture different approaches can be taken. The framework proposed in [STRO96] meets the requirements we have identified through our top-down analysis. However, other approaches are also possible. The authors believe that Web and Java technologies can be applied very successfully in the implementation of the Policy Server's User Interface and Administration functionality. This also applies to the User Interfaces of the Configuration Server. Furthermore, the agents could be implemented as

JavaBeans. The Java RMI (Remote Method Invocation) will be used for inter-communication between the agents (JavaBeans). E.g., the Policy Definition, Transformation, and Conflict Resolution IA will use the RMI to communicate with the Policy Enforcement Module IA and the Policy Monitoring Module IA. The Java IDL (IIOP) will be used for communication between JavaBeans and CORBA objects (e.g., SSOs). I this scenario, it is important to recognize that Java be seen as a client (user interface) technology, whereas CORBA is a server technology[1].

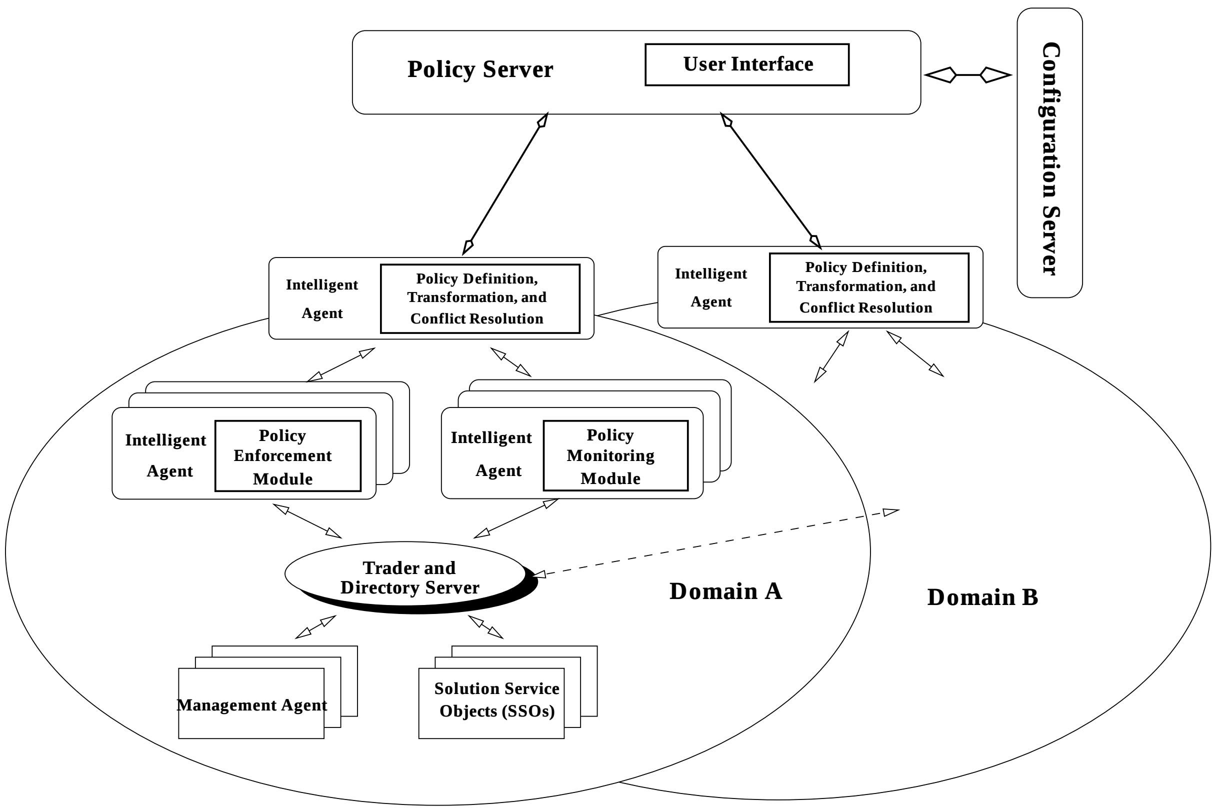
Policy Server
User Interface
Configuration Server
Intelligent Agent
Policy Definition, Transformation, and Conflict Resolution
Intelligent Agent
Policy Definition, Transformation, and Conflict Resolution
Intelligent Agent
Policy Enforcement Module
Intelligent Agent
Policy Monitoring Module
Trader and Directory Server
Domain A
Domain B
Management Agent
Solution Service Objects (SSOs)

Figure 4: Mapping the Policy Architecture to the Framework using Intelligent Agents

3 Assessment of the proposed architecture

After having described how the policy architecture to enforce network and systems management goals and objectives can be realized using intelligent agents, we will now discuss the characteristics of the proposed architecture such as the decide-and-delegate or the provisioning of well-defined policy enforcement code. We will also outline several aspects of the Intelligent Agents technology, that promote the proposed concepts.

3.1 Characteristics of the system

The above concept in combination with the framework by [STRO96] provides the following major characteristics:

[1] Nothing prohibits one from using Java on a large server if a Java interpreter is available for that specific server OS. However, we believe that Data Center managers will not allow applets to be downloaded to machines without being rigorously tested and thereby loosing most of Java's power. The security issues also have to be overcome first. In the client (user interface) world every individual user accepts responsibility to his/her machine. The client side is therefore much more dynamic and Java will be very powerful and successful in this area.

- **Provisioning of policy enforcement code in a standardized way:**
  policy enforcement code is provided by means of the SSO concept, i.e., IAs accessing enforcement and monitoring functionality through traders. New enforcement code will be provided by (simply) creating a new SSO (by encapsulating the required code in the SSO), and registering it at the trader. Alternatively, the changed enforcement code may be downloaded to the SSO and embedded in the extensible agent. The SSO will only be registered after stringent quality assurance procedures, to ensure that the SSO does what it is supposed to do.

- **Provisioning of policy monitoring code in a standardized way:**
  policy monitoring code is provided by means of the IA concept. The IA framework proposed in [STRO96] provides a rulebase for the implementation of low-level policies. The monitoring part of the (low-level) policies will be specified as rules. For the software versions example used in Section 1, the following could be a rule implemented in an IA's rulebase: "If MS-Word NOT_USED_IN_LAST_MONTH then Execute FixedLicenseSSO." This rule could easily be changed by downloading a new rule (or set of rules) to the IA's rulebase, should changes be made to the IT department's business policies. The update of the policy (rule) will be initiated by the policy server. The policy server will send a notification to the IA to update its policies. The IA will then, at its earliest convenience, download the changed policy to its rulebase, to reflect the change in the IT department's business policy.

- **Decide-and-delegate approach to the implementation of policies:**
  the combination of the IA and SSO concepts to implement the policy architecture proposed above, results in a decide-and-delegate approach to the implementation of policies. The IA decides (based on received events, etc.) what to do and then delegates the task of doing it to an SSO. This approach has the advantage that an IA does not become bulky due to the fact that it only makes use of policy enforcement or monitoring code (embedded in an SSO and therefore physically independent from the IA) when needed. The IA does not have to "carry“ code with it that it does not use often. This also results in lightweight IAs, which is an important requirement from a systems management perspective. Current implementations of IAs taking up 30% of the CPU are just not acceptable. The separation of the actions that enforce and monitor policies (realized as SSOs) from the actual IA is a major advantage that also differentiates our approach from similar approaches. We also make use of a "pull" model for the delegation of management policies, which makes the agents more autonomous, by allowing them to get the functionality they need whenever they decide it is necessary. This is a more flexible approach than the corresponding "push" model (which is nevertheless still supported by the architecture).

3.2 Aspects of Intelligent Agent Technology supporting the Implementation

The main intelligent agent characteristics supported by the framework proposed in Section 2.2 are:

- **Autonomy:**
  The IA's autonomy results from the application of rules, implementing low-level policies. For illustration purposes again consider the following rule: "If MS-Word NOT_USED_IN_ LAST_MONTH then Execute FixedLicense". The IA will make an autonomous decision based on the current state of the IA's environment, and execute the FixedLicense if the condition of the rule is true for the domain within which the IA operates. It should be noted that the IA could make use of an SSO (TestAppUsageSSO), with "MS-Word" as parameter to determine the extent to which the application has been used over the last month.

- **Re-activeness:**
  The IA’s re-activeness also results from its application of rules. In response to changes that occur in the IA’s domain, it makes a decision and reacts accordingly. Assuming that from May 10 until June 9 MS-Word was not used by the specific user and that the last day of use was May 9, then on June 10 the condition (MS-Word NOT_USED_IN_LAST_MONTH) becomes true and the rule fires. The IA thus reacts to changes in its domain of responsibility.

- **Social Ability:**
  Agents used for policy enforcement can communicate and exchange information with other agents to enforce for example a common strategy requiring several cooperating agents. In the previous policy example for license allocation strategies, IAs could communicate with each other to optimize their allocation strategy, e.g., agents within a specific domain (e.g., corporate business unit) could "talk“ to each other and exchange information on the availability of floating licenses from certain servers, the stability of these servers or the success rate when requesting licenses. They could even share information on alternative license servers within their domain or outside their domain in other departments including the prices for floating licenses. Hence, IAs do not need to be told by managers which server to request licenses from, but rather share information with other agents to fulfill an overall goal. In other words, the social ability to communicate provides a mechanism to tell agents the goal and not the method by which to achieve the goal.

- **Flexibility:**
  Finally one of the most important properties of the IAs as specified in our model is their ability to dynamically expand their functionality. This enables us to adjust the agent functionality to new requirements imposed on the system without having to reprogram it. Besides, due to the separation of the Policy Enforcement Rules from the Policy Monitoring Rules it is easy to update the former without having to update the latter and vice versa. In this way we get a dynamic and easily adaptable system. The goal is that agents do not possess predefined functionality but according to the situation this functionality can at system run time be dynamically enhanced, modified or deleted.

4 Implementation Aspects

A CORBA-based prototype-implementation of the framework has been done at the University of Pretoria, South Africa, using IONA Technologies' Orbix product. As work is still in progress, the information provided is not complete at this point in time. So far, a relatively straight-forward mapping of the framework [STRO96] to Orbix has been achieved. I.e., the intelligent agents, the "simple“ agents, and the trader are implemented as dynamic objects in the environment.

The following happens when the implementation, depicted in Figure 5, is activated:

1. The IA obtains the Trader's address by sending a message to the Configuration Server's getTraderaddress method.

2. The IA obtains the Directory Server's address by sending a message to the Configuration Server's getDSaddress method.

3. The SSOs obtain the Trader's address by sending a message to the Configuration Server's getTraderaddress method. On receiving the address from the Configuration Server, the SSOs export their services to the Trader by sending a message to the Trader's export method, with two parameters: a name for the service it provides (service) and the SSO's address (object reference.)

4. At any point in time, the Configuration Server can be used to add an entry to the Directory Server. The put method, with two parameters, is provided for this purpose. After the actions described in points 1, 2 and 3 are performed, the IA is ready to receive events from its environment. Upon receipt of an event, the IA will look up the service addressing the specific event (by means of the Directory Server), obtain the appropriate SSO's object reference from the Trader, after which the functionality provided by the SSO is executed. At this stage of the implementation, the SSO is programmed to only print a message to standard output.

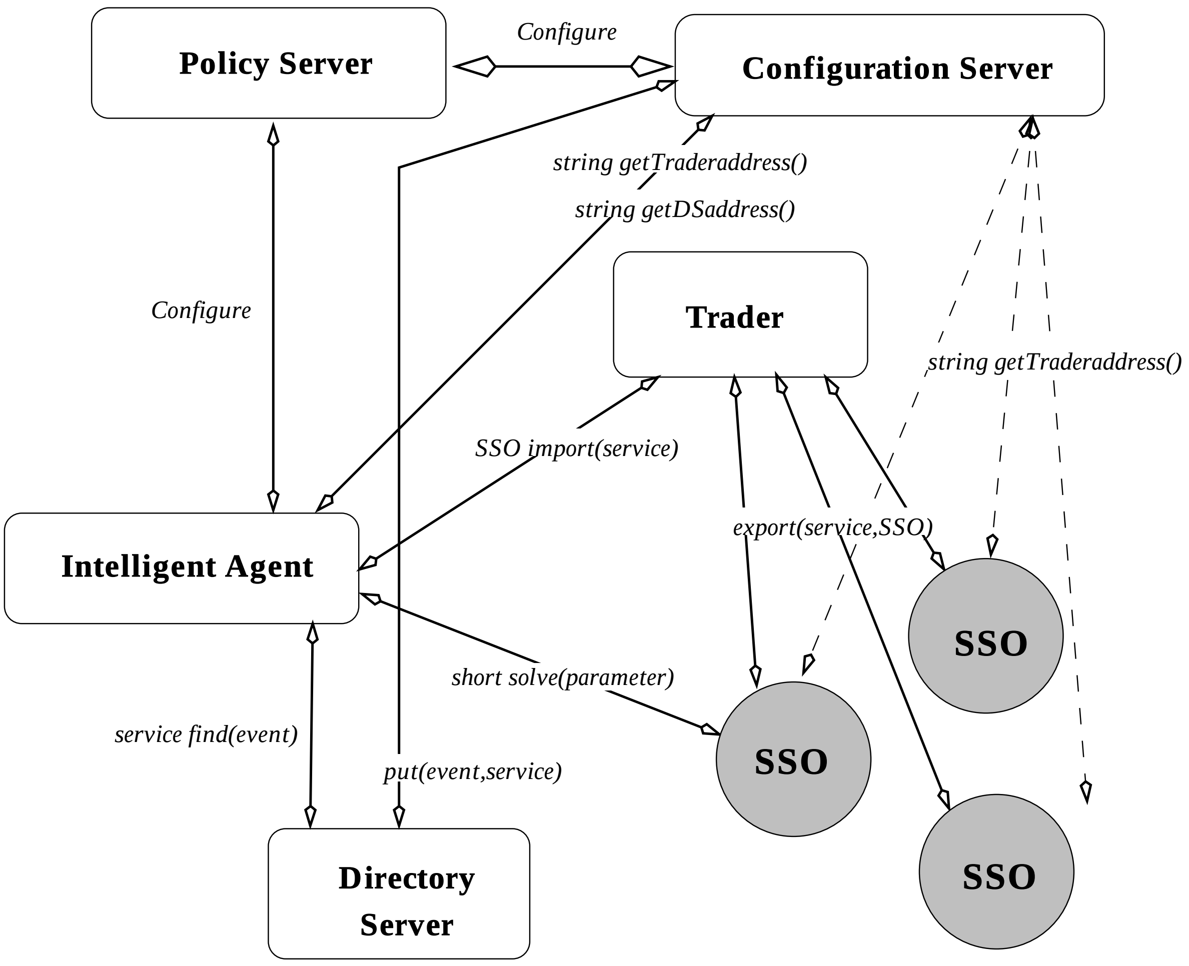

Figure 5: Implemented Infrastructure

As described above, all code written so far only implements the base for further implementations. The next step is to add the policy-dimension to the prototype implementation by means of the Policy Server. The limiting factor in the current CORBA based implementation is that the IAs can only be implemented as primitive objects. Thus, to add "intelligence" to the IA, a major compromise will be made regarding the implementation of rules (which represent low-level policies) by the objects representing IAs. According to the current design and to add the intelligence to the IAs, a matrix will be used by the IAs to store their rules (i.e., rules will be implemented as rows in a matrix.)

The execution engine (embedded as part of the IA's code) will match the events, received from within its domain, against the conditions of the rules in the IA's matrix. A rule will fire when a match is found for the event received. The result of the rule being fired is that a certain action will be taken, which is typically implemented by means of an SSO.

It should be noted that the Policy Server will maintain a global index of rule-matrices for all applicable domains, where as the IA will maintain a local matrix of rules for the domain under its control.

Different operations will be made on the matrix in order to add, delete and update the policies implemented by the specific IA. Matrices and rules will typically be processed in the following way:

- Rules will be added by adding rows to the matrix
- Rules will be deleted by deleting rows from the matrix
- Rules will be changed by overwriting an existing row with a new one

Referring back to the software version management example introduced in Section ???, the following shows the applicable rule that is stored in a matrix with numerous other rules:

Rule Name: Word License request

Condition 1: applicationSW.name = "Word"

Condition 2: applicationSW.version = "7.0"

Condition 3: applicationSW.request_license ("Word", "7.0") = true

Action: Test_App_Usage

The Test_App_Usage action will consist of the if-then-else statement defined in section 2.1. So, whenever a request for a Word license occurs, the Word license request rule will fire. One can almost think of Word license requests as events that occur. The testing and allocation will be delegated to an SSO implementing the functionality of the Test_App_Usage action.

It should be noted that an SSO (TestAppUsageSSO, introduced in Section 3.2) can be scheduled once a day to monitor the usage of applications within a specific domain of users. As soon as it detects that MS-Word has not been used by the specific user over the past month, it forwards the MSWord_NOT_USED_IN_LAST_MONTH event to the IA. When the IA receives the event, it looks up the matching action (FixedLicenseSSO.Activate()) and invokes it.

This implementation is not yet ideal, but it has the advantage that a standard platform is used for realizing the framework. No proprietary technologies are required to implement the framework.

Another advantage to the CORBA-based implementation is that in the application management context, it will be quite easy to instrument applications by means of "simple“ agents, which are implemented as CORBA-objects. The application will communicate status information to the agents, which will be interrogated by IAs. Legacy applications will be instrumented for management by encapsulating certain pieces of the legacy infrastructure (code) by means of SSOs. Integration with the Application Monitoring MIB, currently being defined by one of the IETF sub-groups, will be provided. An SSO will typically act as an element-level agent, used to obtain the information represented in the MIB. This is currently being investigated as one of the sub projects which focuses on Application Management.

5 Conclusions and Future Work

The term Enterprise Management already suggests that network and systems management can no longer exist on their own but need to take business aspects into account. Management policies provide an effective mechanism to enforce such management goals in distributed network and system environments. With the growing heterogeneity and complexity of these environments, it becomes increasingly important to relieve network and system managers from tedious tasks and raise the level of abstraction by adding "intelligence“ to the management resources and applications. Intelligent agents have proven to add this intelligence to managed devices.

Based on the policy architecture presented, this paper has shown how the intelligent agent technology can be deployed to enable the enforcement of policies in distributed environments. We have also shown, that the ideas presented can be realized using standardized mechanisms such as CORBA. Through the prototype-implementation, especially of the SSOs and an SSOManager, further experience with this approach will be gained in the near future.

Further work will focus on the reusability of our agent code in terms of inheritance to other agents and in terms of code delegation. Furthermore, we will extend the monitoring ability of our implementation to cater for advanced pro-active policies.

Acknowledgments

The authors wish to thank Prof. Morris Sloman of Imperial College London and the members of the Munich Network Management Team for fruitful discussions and valuable comments to preliminary versions of this paper. The MNM Team directed by Prof. Dr. Heinz-Gerd Hegering is a group of researchers of the University of Munich, the Technical University of Munich, and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences. The authors would also like to extent their thanks to the students, Nico de Jager and Marne Karsten, at the University of Pretoria, responsible for the current implementation of the CORBA-based prototype.

References

[BEYU 93] Lawrence Bernstein and Christine M. Yuhas, "Truce in Protocol Wars", Journal of Network and Systems Management, 1(2), June 1993.

[BRAT 94] K. Braithwaite and J. Atlee, "Towards Automated Detection of Feature Interaction", In Proceedings of the Second International Workshop on Feature Interaction in Telecommunications Software Systems, January 1994.

[FFMK 92] T. Finin, R. Fritzson and D. McKay, "A Language and Protocol to Support Intelligent Interoperability", Proceedings of the CE& CALS Conference, Washington, June 1992.

[FFMM 93] T. Finin, R. Fritzson, D. McKay and R. McEntire, "KQML: an Information and Knowledge Exchange Protocol", Proceedings of International Conference on Building and Sharing of Very Large Scale Knowledge Bases, December 1993.

[GOLD 96] German Goldszmidt, Distributed Management by Delegation, PhD thesis, Columbia University, 1996.

[GOYE 95] G. Goldszmidt and Y. Yemini, "Distributed Management by Delegation", In Proceedings of the 15th Interna-tional Conference on Distributed Computing Systems, June 1995.

[HNGU 94] H.-G. Hegering, B. Neumair and M. Gutschmidt, "Cooperative Computing and Integrated System Management-- A Critical Comparison of Architectural Approaches", In Manu Malek, editor, Journal of Network and Systems Management, volume 2, pages 283-316, Plenum Publishing Corporation, October 1994.

[INM-IV 95] Yves Raynaud and Adarshpal Sethi, editors, Proceedings of the 4th International Symposium on Integrated Network Management, Santa Barbara, IFIP, Chapman and Hall, May 1995.

[IWSM-II 96] IEEE, Proceedings of the IEEE Second International Workshop On Systems Management, Toronto, Ontario, Canada, June 1996.

[KASE 94] Pramod Kalyanasundaram and Adarshpal Sethi, "Interoperability Issues in Heterogeneous Network Management", In Manu Malek, editor, Journal of Network and Systems Management, volume 2, pages 169 - 193,Plenum Publishing Corporation, June 1994.

[KOCH 95] Thomas Koch, "Rule Base Management Architecture with Smart Agents", In International Workshop on Services for Managing Distributed Systems, IITB, IITB, Karlsruhe, Germany, September 1995.

[LEWI 95] L. Lewis, Managing Computer Networks, Artech House Publishers, 1995.[LUSL 96] Emil Lupu and Morris Sloman, "Towards a role based Framework for Distributed Systems Management", In Manu Malek, editor, Journal of Network and Systems Management, Plenum Publishing Corporation, 1996.

[MASL 96] D. Marriott and M. Sloman, "Implementation of a Management Agent for Interpreting Obligation Policy", In Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, L'Aquilla, Italy, October 1996.

[MEBS 95] K. Meyer, M. Erlinger, J. Betser, C. Sunshine, G. Goldszmidt and Y. Yemini, "Decentralizing Control and Intelligence in Network Management", In [INM-IV 95].

[MODR 96] M. A. Mountzia and G. Dreo-Rodosek, "Delegation of Functionality: Aspects and Requirements on Management Architectures", In Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, L'Aquilla, Italy, October 1996.

[NEWI 96] B. Neumair and R. Wies, "Applying Management Policies to manage Distributed Queuing Systems", In Distributed Systems Engineering Journal, Special Issue on Distributed Systems Management, 1996.

[POAT 96] K. Pomakis and J. Atlee, "Reachability Analysis of Feature Interaction: a Progress Report ", In Proceedings of the International Symposium on Software Testing and Analysis, January 1996.

[PORT 85] Michael E. Porter, Competitive Advantage, Free Press, 1985.

[SLOM 94] Morris Sloman, Network and Distributed Systems Management, Addison-Wesley, June 1994.

[SLOM 95] Morris Sloman, "Policy Driven Management for Distributed Systems", In Manu Malek, editor, Journal of Network and Systems Management, volume 2, pages 333-360, Plenum Publishing Corporation, 1995.

[STRO 96] P. Steenekamp and J. Roos, "Implementation of Distributed Systems Management Policies: A Framework for the Application of Intelligent Agent Technology", In [IWSM-II 96].

[WIES 95a] R. Wies, "Management Policies: Mapping Policy Specifications to parameterized Descriptions of Management Capabilities", Proceedings of the Second Workshop of the HP OpenView University Association, University of Munich, March 1995.

[WIES 95b] R. Wies, Policies in Integrated Network and Systems Management: Methodologies for the Definition, Transformation, and Application of Management Policies, PhD thesis, Universität München, June 1995.

[WOJE 95] M. Wooldridge and R. Jennings, "Intelligent Agents: Theory and Practice", submitted to: Knowledge Engineering Review, January 1995.