Agent Based Security for the Active Network Infrastructure

Stamatis Karnouskos, Ingo Busse and Stefan Covaci

German National Research Center for Information Technology
Research Institute for Open Communication Systems (GMD-FOKUS)
Kaiserin-Augusta-Allee 31, D-10589 Berlin, Germany
http://www.fokus.gmd.de/ima/

Abstract. Security in Active Networks is still in its infancy! This paper presents a new Agent-Based Security architecture for the Active Network Infrastructure (ABSANI). It is explained why agents in combination with Java are considered the appropriate solution for security architecture and how this can be applied in the Active Networks. An agent-based Active Node architecture is introduced and ABSANI is placed within that approach. Subsequently, all the basic components of the ABSANI are analyzed arguing for the benefits they offer. Finally an application scenario of Place-oriented Virtual Private Networks is demonstrated.

1 Introduction

This approach integrates multi-domain parallel evolving technologies (Agents, Java, CORBA). We try to mix the benefits of Agent Technology and where needed of CORBA in order to apply it successfully to the Active Networks domain. We present shortly these areas, how each one can be used as a benefit to the other and where and why our approach stands today in relation with the already ongoing research.

1.1 Active Network Technology

The last years a variety of approaches have been pursued in order to provide a flexible programmable network infrastructure that could "change its behavior on drop of a dime". Active Network (AN) technology aims to move dynamic computation within the network and therefore making it more intelligent not just to its end-points but also in the intermediate nodes. An Active Network is a group of network nodes (switches, routers, -called Active Nodes hereafter-) that support the deployment and execution of user applications (embedded in the user communications), without interrupting the network operation. In this way, an Active Network is in the position to offer dynamically customized/programmed network services (e.g. connection) to the customers/users or even enables users to inject their own applications to support their communication needs. Programmable networks open many new possibilities for innovative applications that are unimaginable with traditional data networks. This dynamic network programmability can be conceived by two different approaches:

I. In-band programming of the network nodes (also widely known as the capsule approach). The program is integrated into every packet of data sent to the network (the program is injected on the same path as the data). When these capsules arrive at the Active Node, the node evaluates the programs and adapts its functionality. The programs within the capsules are typically very small due to the size limitation of the packets and the transport overhead imposed by the capsule programs. *Active Network programmability based on capsules is therefore limited.* That is definitely negative especially in the context of connection-oriented communication environments where active node re-configuration/programming (activated by the reconfiguration of network connections) is needed much less frequently than the processing of packet payload. It is not necessary and not very efficient to equip each data packet with a computation capability as this adds too much overhead to the processing of packets. Thus capsules have very low utilization in such context.

II. Out-band programming of the network nodes. Here the programs are injected into the node in a different session from the actual data packets that they affect. User would send the program to the network node (switch/router) where it should be stored and later when data arrives, it is executed processing that data. The data can have some information (e.g. special tags) that would let the node decide how to handle it or what program to execute. Within this approach which makes clear the separation of data/communication packets nodes can be programmed *via injection of new program code* into the active nodes, where injection can typically be done by specific packets (e.g. mobile agents) that are evaluated at the network nodes. Our architecture supports exactly this approach.

Finally in this category falls also the notion of *remote manipulation* (binding) of the node's resources through a set of well defined interfaces [1]. This is not considered a pure AN approach as we have high-level configurability/remote manipulation and not programmability of the node. The difference between remote manipulation and active code injection is similar to the difference between a RPC-based and a Mobile Agent (MA)-based software design paradigm, where MAs can help to increase the flexibility and robustness. In addition, it allows for load balancing of the active network services.

1.2 Agent Technology

Software agents is a rapidly developing area of research. The research community has still not found a clear answer to the most popular question "What is an agent?" and the debate still goes on. A general answer could be: *Agents are software components that act alone or in communities on behalf of an entity and are delegated to perform tasks under some constraints or action plans.* However agents come in myriad of different types depending on their nature and the environment.

Examples are: Collaborative agents, Autonomous/Proactive agents, Interface agents, Mobile agents, Reactive agents, Hybrid agents, Intelligent/Smart agents, Mental/Emotional agents etc.

The above categorization is not unique and depends on some of the attributes agents show in greatest degree. Of course there can be mixed agents i.e. an Intelligent Agent can also be Mobile. In our Active Network infrastructure a variety of agents can be used. E.g.

- Intelligent agents that reside on the node and "intelligently" configure the node's resources for optimal performance.
- Mobile agents that can be "dumb" but execute trivial tasks in all nodes of the Active Network Infrastructure
- Collaborative agents that work in teams and take care of the security within an Active Network domain. E.g., automatic certified security updates on the AN nodes, elimination of denial of service attempts by blocking the source of attack to the nearest AN node etc.

Mobile agent systems provide the AN infrastructure with many advantages. MAs shatter the notion of Client/Server model and eliminate its limitations. They provide robust networks as the hold time for connections is reduced only to the time required to move an agent, the agent carries credentials and therefore the connection is not tied to constant user authentication, load balancing can be achieved as there is no request flow across the connection in order to "guide" the agent and respond to results, there even has been already standardization efforts defining interoperable interaction between agent systems [2].

2 Motivation

*Security in Active Networks is still in its infancy!*Active node programming is typically a security-critical activity. Of course in such a programmable network the security implications are far more complex than in current environments. Although there has been some research concerning the security of AN little or no effort has been made to make a *dynamic, extensible, configurable and* interoperable. ANs demand that this security architecture is as highly programmable and evolvable as possible. Extensive and expensive authentication measures are necessary to protect the active node resources from malicious intrusions. Such security measures can not be applied on the basis of individual packets due to their time and space requirements.

Our solution is an Agent Based Security Architecture for Active Networks. With this approach we don't seek a one-side technological approach to the AN security problem but the integration of parallel evolving technologies. ABSANI aims in integrating cutting-edge technologies in order to produce a high-security architecture and deal with the advanced security threats that Active Network technology introduces. There is no need in re-inventing the wheel in the security approach we take. By building upon existing security schemes we make sure that our architecture is open and interoperable.

We understand that these are parallel developing domains into which much research effort has been invested the last years and which will keep on evolving fast.

By integrating state-of-the-art components we make sure that our architecture stays up-to-date and advances/adapts to current needs as its components evolve. That not only is in favor of its internal/external security but also of its lifetime. Within the ABSANI architecture we try to encompass the flexibility and special characteristics of agent technology.

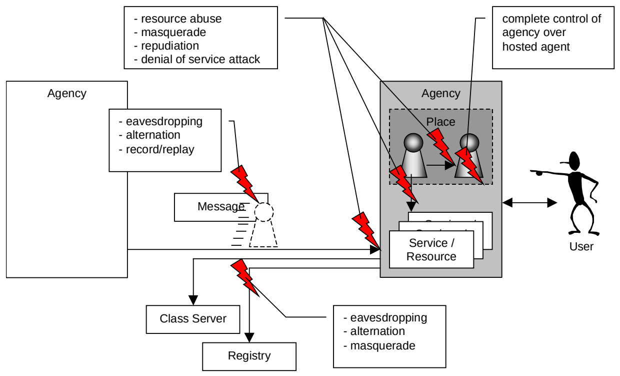

Fig. 1. Security Threats to Agent-Based Applications

We use the agent-based approach to program an Active Node. In such an environment author of the MA code, the user, the owner of the hardware, the owner of the execution platform can be different entities governed by different security policies in a heterogeneous environment. As we also see in Fig.1 security in such an environment is an extremely sensitive issue. The hosts have to be protected from malicious agents and the agents themselves have to be protected from malicious hosts or other malicious agents who could attack them. Moreover the communication road between the AN nodes has to be protected with state of the art security techniques. The Agent Community as well as the AN Community work on these topics. Our open security architecture assures that future solutions in the agent security domain can be applicable to our approach, therefore strengthening the Node's protection system.

3 The Active Network Architecture

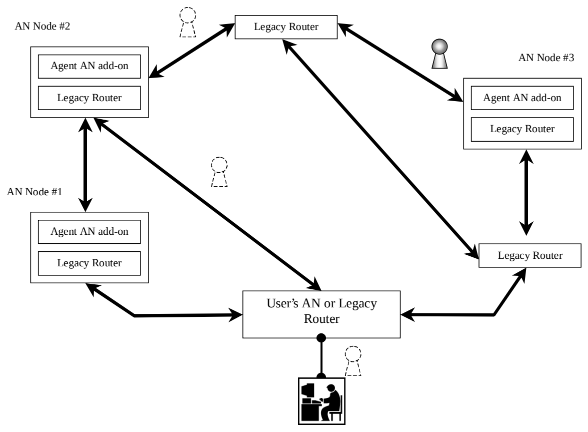

Fig. 2. Active Network Infrastructure

The Active Network Infrastructure is seen as a network of co-existing AN nodes and legacy nodes. User initiates agents that traverse the network and configure the Active Nodes. In Fig.2 the user has initiated an Agent to change the behavior of AN Node #2 and AN Node #3. The agent visits the target node and executes. Then, having fulfilled its tasks, moves to the next AN Node via the Legacy Router. There he executes again. Our notion of an Active Node architecture is with embedded the agent technology (illustrated in Fig.3). As we can see agents can empower current Routers and transform them to Active Nodes. The resources of the node can be accesses/controlled by visiting agents and according to the node’s policy schemes.

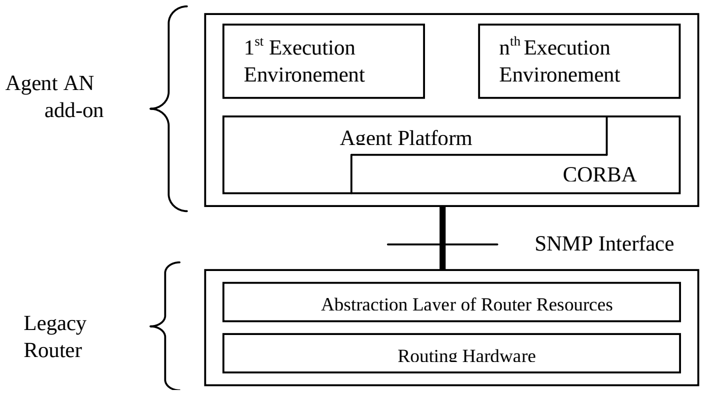
Agent AN add-on
1st Execution Environement
nth Execution Environement
Agent Platform
CORBA
SNMP Interface
Legacy Router
Abstraction Layer of Router Resources
Routing Hardware

Fig. 3. Active Node Architecture

4 The Security Architecture

Security can’t be an afterthought! It has to be integrated with the node's core function and not implemented at the end as an extra, optional or explicitly called service. The new security architecture for AN proposed hereafter is based on mobile agent technology. Wherever we detect significant benefits we make use (Fig.4) of the Common Object Request Broker (CORBA) [3] which is today an established standard that enhances the original RPC based architectures by allowing relatively free and transparent distribution of service functionality. Currently no standard that handles the interoperability between different agent platforms and the usability of CORBA services by agent based components exists. By further developing this architecture we hope to provide feedback to future standardization efforts.

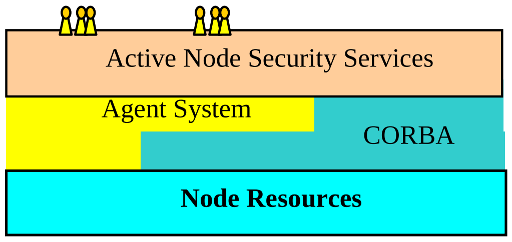

Fig. 4. Technology view of ABSANI

The architecture consists of Places that interact with the core of the architecture (Fig.5). The communication is done mainly between the enforcement engines and Resource Managers. Analytically the components that this architecture consists of are:

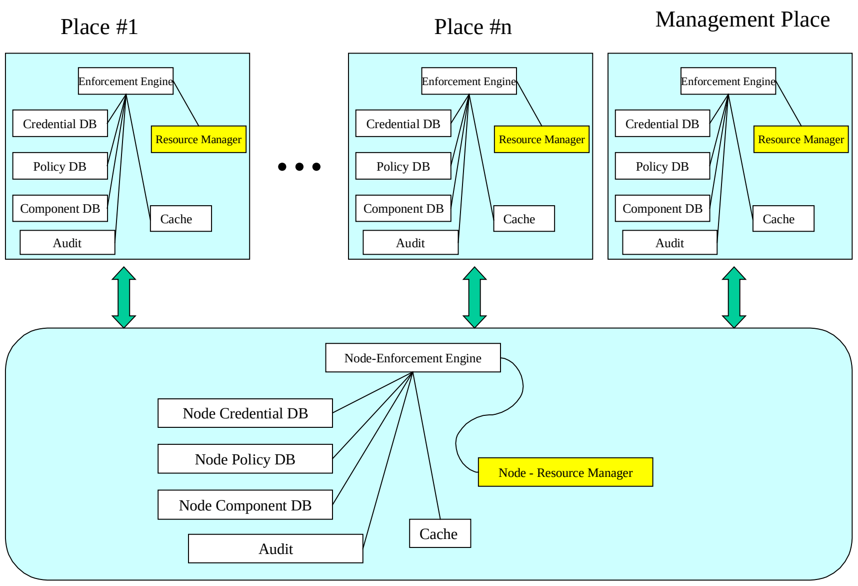

Fig. 5. Overall Architecture view

4.1 Place

A Place is a context within an agent system[1] in which an agent is executed. This context can provide services/functions such as access to local resources. A place is associated with a location which consists of a place name and the address of the agent system within which the place resides.

A Place can be used in different ways. Places are:

- *Dynamically* assigned to agents as they enter the node. The criteria can vary e.g. all agents coming from a specific user or agents belonging to a specific policy scheme etc. A policy manager and a resource manager are assigned to the Place and are given the general security guidelines, which can never be bypassed. If an agent has sufficient credentials then he can fully interact with the components

[1] An agent system is a platform that can create, interpret, execute, transfer, and terminate agents. An agent system is identified by its name and address uniquely. One or more Places reside within an Agent System.

e.g. change the Place's policy, ask for more resources, insert elements into the component database etc.

- *Statically* assigned per entity (e.g. user, enterprise etc). Again static resources are given to the Place and the local Resource Manager manages them. With this way it is possible for an Enterprise to setup a network of Places in various nodes, creating a Place Oriented Virtual Private Network (PO-VPN). This offers several advantages e.g. secure communication between company-trusted places etc.

4.2 Policy DB

The Policy Database is responsible for maintaining all policy schemes. By separating the PolicyDB from the Enforcement Engine we insert a dynamic way of policy modification within the node. We use an already existing language to define the policies to be stored in the database.

The security policy defines the access each piece of code has to resources. Signed code can run with different privileges based on the key that it used. Thus users can tune their trade-off between security and functionality (of course within the allowed limits).

We make use of the principal of least-privilege. This principal states that only the minimally powerful authority should be used to authorize a request for access. Thus any mistakes from "powered" users will lead to the least possible damage. Following this thought a principal with the authority to do many different things should be able to indicate which one of those authorities should be used in a specific request. E.g. An administrator wants to backup the Node's databases. He holds two keys the Supervisor_Key (allowed to do anything within the DB) and Read_Key (allowed only to read the DB). He should use the second key to backup his DB. Thus even if something goes wrong no modification/damage can occur at the DB.

Any attempts to describe the security policy in terms of each individual principal's authority to access each individual object is not scalable and not understandable for those instituting the policy. Thus it has been proposed to group principals and objects into sets with common attributes, where the attributes are used in making security decisions rather than the individual identities. So we have Role-based Policy, Group policy, clearance labels, domains etc

We are also experimenting with the KeyNote Trust Management System [4] in order to realize flexible policies. In any case policy files are human-readable/understandable.

4.3 Credential DB

Credentials of principals/code & components are stored in this database. A principal is an entity that can make a request for access that is subject to authorization. Security relies not only to the authentication of the entity but also to the activities he wants to perform. The credentials combine a description of the identity of the principal and also attributes associated with the principal and the actions he wants to perform to take the decision whether he is granted to do what he asks for or not.

Scenario: The principal may want to execute code that is not trusted (but the principal is trusted). On hard node security level this should be denied. Therefore the Enforcement Engine checks a) if the principal is trusted and allowed to perform the desired action b) if the code he wants to execute is trusted.
X509v3 and SPKI Certificates [5] are used as credentials in a heterogeneous environment with a key used as the primary identification of a principal. The credentials include a hash of the content, list of signers and their signatures, certificates, other info associated with the specific action or agent. Credentials can be associated with various components such as agents, code, policies etc.

Credentials are used to:

- Verify that the component was created/distributed/authenticated by the claiming principals.
- Verify that the component hasn't been altered after it has been signed.
- Fulfil partially the non-repudiation need so that the originator of that code can't deny it.

4.4 Component DB

The Component Database can be considered a general Database of active code, protocols, etc. It can also be used for caching agent's code but its use is far more extended than simple caching. As we will demonstrate it is a non-removable part of this architecture that strengthens the overall security. Security is by nature overhead in the communication and execution in order to protect the system. We accept that. Yet there are novel ways/techniques to minimize this overhead (under certain conditions) and fortify the Security on the node.

4.4.1 The multiple re-visit by the same agent scenario:

An agent performs multiple visits to the node. Each time we verify the agent's credentials, put him within a specific policy framework, check it while it executes, authorize every call it makes in other objects or resources it wants to use. It is obvious that if this agent is a frequent visitor it is dull to re-apply the same actions again and again. A caching scheme must be used. Now this caching can be done in different levels. We can cache the agent's code, the agent's credentials, components that the agent needs, monitor the agent's use of resources and associate with a specific agent code etc. Then the next time the agent comes to the node we don't have to verify its user nor its code. Also as it has executed before we know approximately what its behavior and needs are. Furthermore we have in the Component DB stored its verified, checked and authorized code. Thus we take from our Component DB the code of the agent (which we trust) and only the data of the newly arrived agent. In that way we avoid the repetition of authorizations which are time consuming. Of course this is a policy matter and can be changed but the node should have the means to provide this flexibility, and in order to do that we need the Component DB.

4.4.2 The common component usage scenario :

As before, we have agents that visit our node. In this case the distinguishing-characteristic is not that the code of the agent comes is the same (as in 4.4.1), but that they make use of similar components. E.g. the agents in order to execute an action need some special protocol or some special cryptographic module etc. We (the Node Manager or even the Place Manager) could provide such components in the Component DB signed and tested in the specific environment. The agent then can make a call to those components and perform its actions. As all components will be signed the agent can decide whether it is safe to use those components or not.

Such a DB serves in multiple ways. The agent can be lighter as there is no need to carry everything he needs, the node security is enforced as it executed components that have been thoroughly tested by the Node Provider and all the actions are faster as overhead due to security actions are minimized.

4.5 Resource Manager

A Resource Manager is available in order to handle the resources.

- Place Resource Manager. The Place resource Manager can handle the resources that are dedicated to a specific place. It can be contacted also directly via the agents that reside in the associated place also in the case that there is a need for more resources.
- Node Resource Manager: Handles the LocalNode Resources. It is contacted via the NodeSecurityManager or via the PlaceResourceManager (Fig.6). It is also the gateway to the resources of another node or nodes. An interface is provided on how this security Architecture interacts with the Resource Manager.

Note that the resources available to a certain Place are transparent to the Agent. That means that local resources could be extended via CORBA in order to access resources in other AN nodes. This helps with the Place Oriented Virtual Private Network (PO-VPN) as we will explain later.

4.6 Cache

The Cache is another essential part of the architecture in order to improve performance. Security checks are time-/computing- consuming processes. In our effort not to duplicate all the time the security checks we have a cache. Caches exist in all Places and are accessible via the Security Enforcer only. Security checks that have been done via the Enforcement Engine are stored with a time limit in the cache. If the time limit expires then the security checks are performed again, otherwise the security check is considered valid and is used by the system.

The Policy DB can be dynamically updated via the Enforcement Engine any time. Thus the problem is faced that the cache contains outdated information. We solve this problem by deleting -each time the policy for an Entity changes- the cached security checks that are associated with this key/person partially or completely. So next time a security check is requested it will not exist in cache and it will be performed from the beginning. This is a novel method to speed-up the performance of our architecture.

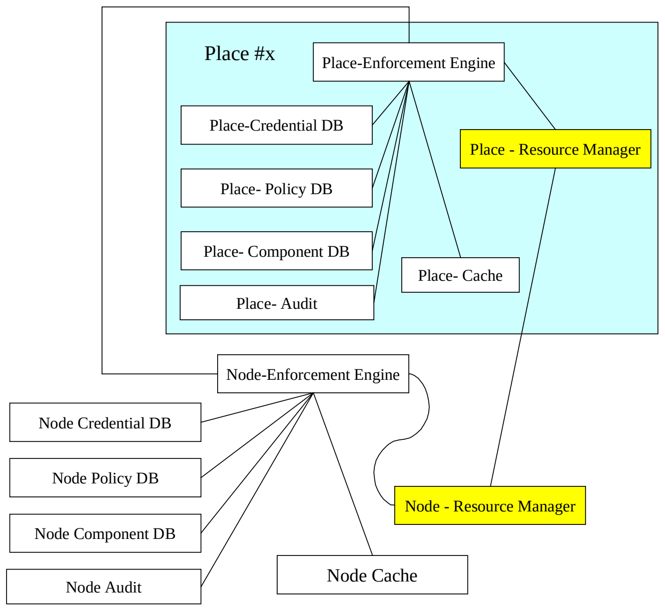

Fig. 6. Figure 1 - Component Communication View

4.7 The Node Management Place

A special dedicated Place, the Management Place is responsible for changing the Node's general behavior (Policy, DBs etc). Agents that execute in that environment are "privileged" agents and are under highest security controls. They are able to modify the node Databases and its security scheme, thus extra care has to be taken. Generally this environment should be restricted only to Node Administrators. Normal users can change the behavior of Places assigned to them but they are not able to contact/execute within the isolated and highly protected Management Place. Provisioning and Configuration is done only via the Management Place.

4.8 Auditing

Experience has shown that 100% security is difficult to realize - if not impossible - due to the multiple factors that interfere. Collecting data generated by network activity provide a useful tool in analyzing the existent security and also trace back (if possible) the originators of a security breakout. Audit data include any attempt to achieve different security level or change entries in the system's databases etc. Intrusion attempts can also be detected via audit e.g. when we see repetitive failures in the attempt to use a component/service we can adapt our policy/behavior so that we prevent any possible intrusions. The more detailed the audit process is the better can various activities be debugged and protected from repeated errors or false configurations.

4.9 Enforcement Engine

The Enforcement Engine is used to enforce the policy on the Node and on the Places. An Enforcement Engine must satisfy three important rules.

- It is always invoked. The Enforcement Engine should *not* be called explicitly. Each action should be evaluated and allowed only if it complies with the Policy.
- It is tamperproof. The information that the Enforcement Engine relies on shouldn't be altered in any way by third unauthorized entities. This calls for Signed objects that no-one can alter.
- It is verifiable. Enforcement Engine relies on trusted unchanged basic code in order to boot-up. Then its abilities can be expanded.

The Node Administrator is able to use a GUI and edit the Node Policy & Credential Database prior of system run. Place Administrators are able to alter their Policy & Credential DBs via Agent Interface.

5 The Language Decision

One approach is to design a new language tailored to the needs of active networking and our system. The difficulty would be i) designing from the scratch a new language with a bunch of desired features (e.g. safety, performance), ii) if we don't manage to address all required features needed by the user it would be impossible for user to implement the mobile code he wants, iii) it would require a huge amount of work to keep the language up-to-date with all needs, iv) it would be used by a limited number of people (AN people only) and therefore bugs, errors, etc would be seldom if not at all reported.

The other approach is to use an existing language. Java is a very popular language designed especially for mobile code and with security in mind. Multiple research (and not only) domains use this language. Therefore bugs, errors are found and reported fast. The language is a commercial product and advances as day by day new features/libraries are added. Also Java is a safe language. The basic security concepts in Java are based on the following components: language design, byte code verifier, class loader, security manager.

In the following each part will presented and investigated on how to use the concept to support a security model for the agent platform. First of all, Java is a safe language. That means, there are several mechanism inherent to Java, providing protection against incorrect programs, notably: strictly typed language, careful control of casts, lack of pointer arithmetic,automatic memory management including garbage collection to avoid memory leaks and dangling pointers, check of array references to ensure that they are within the bounds of the array

Even though a compiler performs thorough type checking, there is still the possibility of an attack via the use of a “hostile” compiler. Since the agency does not load the source code of an agent but already compiled code in the form of class files there is no way of determining whether the bytecodes were produced by a trustworthy compiler or by an adversary attempting to exploit the agency. Therefore a class verifier is called for.

The class verifier [6][7] of Java is used to check every class that is loaded into the Java virtual machine over the network. Before any loaded code is executed, the class is scanned and verified to ensure that it conforms to the specification of the Java virtual machine. The class verifier operates in four passes.

The first pass checks that the class file is conformant to the class file format. The second pass performs all verification that can be performed without looking at the bytecode. This includes for example a check whether final classes or methods are subclassed or overridden, respectively. The third pass is a data-flow analysis on each method assuring that there will be no stack over- or underflow, registers always have a value when being accessed, methods are called with appropriate arguments, types are correctly used, and that the opcodes have appropriate typed arguments on the stack and in the registers. This is also referred to as the bytecode verifier. The fourth pass is done during run-time. It ensures for example that a method exists when being called, i.e. it guarantees that the symbolic references are working.

The class loader [8] first checks the local codebase of an agency. If a class is available locally it is not loaded over the network but from the local codebase. This prevents the system classes with access control checks from being replaced. In addition the class loader sets the protection domain.

The security manager [9] is contacted whenever sensitive system resources, such as the file system or the network for example, are accessed. A check method is called in order to determine whether the calling entity has the required access permissions. To distinguish between the access of foreign classes and the access of system classes the call stack is analyzed. The class loader of each call on the stack is determined and the permissions are the intersection of the permissions of each protection domain which is contained in the class loader.

6 Design Goals Fulfilled

This Security Architecture has been designed with the following guidelines in mind:

- ***Simplicity***.
  The model is as simple as possible to understand and administer. The simpler the whole thing is the better the Security Architecture functions and evolves.

- ***Scalability.***
  Our Security architecture can be applied from small and low agent/node populated systems up to large intra- and inter-enterprise ones. To make that sure we: i) have a flexible and advanced policy and access controls (role-based security etc), ii) support of various domains that enforce different policies, iii) Manage distribution of data and cryptographic keys e.g. across the network without human intervention.

- ***Flexibility***
  This is probably the most significant driving force in the design of this security architecture. Flexibility is enhanced to the maximum for end-users and administrators but not at the cost of safety/security. Choice of access control Policy, choices of audit policy or security functionality profiles are some examples.

- ***Interoperability***
  The architecture uses CORBA for interoperability reasons. CORBA guarantees consistent security schemes among heterogeneous systems where different ORBs are deployed by various vendors. We raise this security to a higher level so that the Agent world is able to use these advantages. Also we use the Grasshopper [10] a MASIF [11] compliant Agent Platform.

- ***Performance***
  The trade-off between Performance and Security is always a controversial issue within the research community. Security is by its nature overhead. Though different users have different needs. We can't simply provide a homogeneous security facility. Security should be user or even task specific. The super-user who enforces a specific security scheme (in a Place or a node) should also be able to select/decide on the tradeoff between certain security and performance (of course within some limits). For better performance we have included Caches within each execution place as well as Component_DBs where code can reside. We hope that in the future the performance of Java will be also improved.

- ***Object-Orientation***
  Interfaces are purely object oriented. We think that in this way system's integrity is promoted and complexity of security mechanisms is hidden under simple interfaces. Those interfaces could be at any time changed/enhanced without an impact on the way this architecture nor its users use them. This approach offers also survivability as well as ability to advance and adapt to future needs.

- ***Access-Control***
  Access Control aims at preventing an agents from accessing unauthorized resources. In our Security Architecture calls to resources are intercepted. Then the Security Manager is called in order to decide whether this call complies with the Policy. If so the action is allowed otherwise denied and an error code is returned. Primarily the following goals must be satisfied : *Safety* - A Safe system limits the possibility that an agent will write to another agent's namespace and therefore

bringing it into an unstable, false or unintended state. *Privacy* - Agents should not be able to access the address space of another Agent and read it's data.

- ***Safety***
  The use of a safe language such as Java provides some guarantees concerning the Safety.

- ***Conditional-Access***
  Most traditional operating systems deny or allow access. Via our Security architecture we are able to allow conditional resource access. E.g. one Agent can request more memory in order to execute additional tasks. The PlaceResourceManager contacts the NodeResourceManager and requests e.g. more memory. The Enforcement Engine checks to see if this request complies with the current policy and if so more memory is dynamically assigned to a Place for a certain time.

7 Place-Oriented Virtual Private Networks – An Application Scenario

An application scenario is introduced here in order to show the flexibility/advantages ABSANI offers. We introduce the concept of Place-Oriented Virtual Private Networks (PO-VPNs). VPNs offer to Enterprises the opportunity to construct their own Network and administer it the way it eases their needs. The ABSANI architecture has been designed with this goal also in mind: The offer of Places which can be leased to 3rd party entities and managed by them. This of course assumes partitioning or multiplexing of available resources.

An Enterprise can obtain for its needs a Place in strategically located Active Nodes, thus constructing a PO-VPN. As it can manage all policies/resources on the assigned Place, it has complete control (to the allowed limits set by the Node Operator) over the PO-VPN. Actually this looks like a distributed Agency spread over various nodes.

A possible scenario is that an Enterprise wants to create a PO-VPN. As various providers would offer services in various prices it could be a benefit to chose among the best offers not overall (as an offered packet) but partially (specific service selection). What we mean exactly with that? Suppose that a provider offers high speed processing power (fast CPUs) but limited storage capacity to the node. A second one offers better price on huge amount of storage but low processing power. The user should be able to combine both. E.g. execute the code in the fast node and have results stored on the slow node with the extended storage.

That is of course difficult to realize/implement at the moment but we would like to leave this possibility open as this could be a future evolutionary step. In any case those kind of scenarios are supported by the Security Architecture we have presented here.

8 Summary and Conclusions

An agent-based Security architecture has been presented. ABSANI uses the mobile agent technology and the benefits that derive from it in order to apply them in the Active Network domain. It has been demonstrated with an agent-based active node architecture how agents can empower the current passive routers and transform them to Active Nodes. We have placed the security architecture within this AN nodes. We have showed that benefits such as simplicity, scalability, flexibility, interoperability, performance and safety have been addressed successfully. Agent community has invested a lot of effort trying to make mobile code secure and flexible and Active Networks' objectives can be achieved via our approach. This approach provides a dynamic, extensible, configurable, and interoperable way to secure Active Networks. Also with the use of Java we can guarantee a high level of safeness. Furthermore by combining approaches (in a Lego-like way) we enhance not only the interoperability of our architecture but as well its lifetime. ABSANI offers a security scheme dealing successfully with the current needs of secure active networking, and will continue its fast evolvement as long as agent technology keeps on advancing.

References

1. C. M. Adam, J.-F. Huard, A. A. Lazar, K.-S. Lim, M. Nandikesan, E. Shim, "Proposal for Standardization of ATM Binding Interface Base 2.1", submitted to P1520, January 1999.
2. Mobile Agent System Interoperability Facility, URL : http://www.fokus.gmd.de/research/cc/ima/masif/
3. OMG Web Site : http://www.omg.org/
4. The KeyNote Trust-Management System. URL : http://www.cis.upenn.edu/~angelos/keynote.html
5. Simple Public Key Infrastructure. URL : http://www.ietf.org/html.charters/spki-charter.html
6. F. Yellin, „Low Level Security in Java", 1997, deleted from www.javasoft.com.
7. B. Venners, „Security and the Class Verifier", JavaWorld, October 1997, http://www.javaworld.com/javaworld/jw-10-1997/
8. B. Venners, „Security and the Class Loader Architecture", JavaWorld, Septermber 1997, http://www.javaworld.com/javaworld/jw-09-1997/
9. B. Venners, „Java Security: How to Install the Security Manager and Customize your Security Policy", JavaWorld, November 1997, http://www.javaworld.com"javaworld/jw-11-1997/
10. IKV++ GmbH - Grasshopper URL : http://www.ikv.de/products/grasshopper