Realization of a secure active and programmable network infrastructure via mobile agent technology

Stamatis Karnouskos

Fraunhofer Institute FOKUS, Kaiserin-Augusta-Allee 31, D-10589 Berlin, Germany

Received 28 January 2002; accepted 28 January 2002

Abstract

The deployment of sophisticated telecommunication services poses demanding design and implementation challenges to the underlying infrastructure. The end-goal of more flexibility, fault tolerance, quality of services, intelligence, component-based service integration, service personalization, programmability, openness and of course security in a heterogeneous infrastructure can be reasonably achieved via active and programmable networks. In this paper, we first investigate an integrated architecture for active and programmable network infrastructures that is based on mobile agent technology. Subsequently we present a security architecture for our node and comment on its functionality and technology choices made. At the end a dynamic VPN deployment with nomadic user support scenario is analyzed in order to argue about the pros and cons offered by this approach.

Keywords: Active networks; Security; Active code; Mobile agent technology; Nomadic user support

1. Introduction

The global telecommunication market in an open economy world requires rapid deployment of personalized interoperable services and has therefore several sophisticated requirements from the underlying infrastructure. Computational Intelligence (CI) has the potential to overcome the emerging challenges whose increasing complexity is at least difficult to satisfy with the usage of conventional computing technologies. The CI backbone constitute technologies such as neural networks, granular computing (also termed as fuzzy set technology), evolutionary computing as well as combinations between them, e.g. neurofuzzy computing. CI's applicability to active networks (AN) [17] as well as agent-enabled AN infrastructures provide an interesting framework where several problems such as difficult and slow integration of new technologies, standards and services that haunt current passive networks cease to exist.

Active and programmable networks [1] represent an evolution of current dumb passive network carriers to a more general programmable network model. They foster the idea of moving service code, which traditionally was placed outside the transport network, directly to network's nodes. Those nodes are now open and allow applications (e.g. web proxies, firewalls, etc.) to configure them optimally for their tasks, via open interfaces (programmable networks). Furthermore, those nodes are able to accept injected code that computes on data received by the node, before they pass them to the next node (AN). Network-aware software is expected to change the way we design and deploy applications and services, as it will allow us to test and adopt more sophisticated and intelligent approaches coming mainly from the CI domain. The challenge such an infrastructure poses is to find the right balance among mobility, flexibility, performance, robustness, usabilility and last but not least safety and security.

Software agent technology is a rapidly multi-developing area of research since the early 90s. An agent is a software component that acts alone or in communities, on behalf of an entity and is delegated to perform tasks based on its internal goals. There are several types of agents [5] but for the AN case, we are especially interested in mobility and partially in intelligence. Mobile agent technology [2] offers a new computing paradigm which shatters the notion of client/server model and its limitations [3], in which a program in the form of a software agent (intelligent or dumb) is capable of migrating autonomously.

2. Combining active networks and agent technology

There are two approaches via which we can realize active networking, namely the *in-band* (also known as the capsule

E-mail address: karnouskos@fokus.fhg.de (S. Karnouskos).

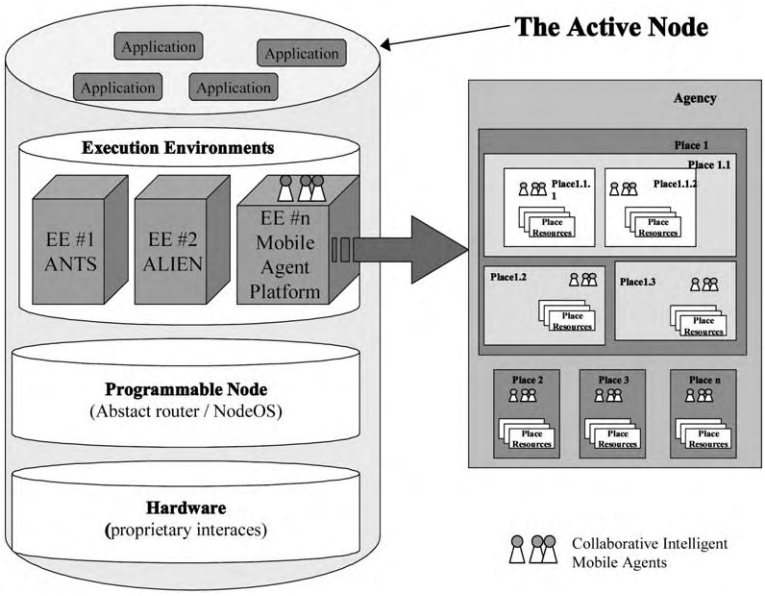

Fig. 1. The agent-based AN node.

approach [4]) in which the active code is integrated into every packet of data sent to the AN node, and the *out-band* in which the active code is injected in the AN node in a different session from the actual data packets that it affects. The agent approach which is discussed in this paper falls within out-band programming category.

As active code (AC) we consider the code that executes within the execution environment of the active node. This code can be stateless, statefull, stationary, or mobile. An intelligent piece of code that moves around as AC can be considered as the most advanced form of AC. All other forms derive from this one by combining some but not all characteristics mentioned within the intelligent and mobile agent research domain. It is important to clarify that an agent can be an AC or not. Agents can slip easily in the role of AC, but also serve as middleware technology. Agents can be used in two ways in ANs (i) as Active Code Carriers (ACC) and (ii) as AC. From hereafter, we consider the AC as an intelligent mobile agent that traverses the network. We will refer to it as AC or agent but for this paper, both references are synonyms.

2.1. The agent-based AN node architecture

The active node architecture with the agent execution environment is depicted in Fig. 1. An active node (router, switch, etc.) can be realized via the composition of three different layers representing hardware and software parts, i.e. the static part, the programmable part and the active part.

Static part. This is the hardware that is delivered by the manufacturer. It contains the optimized components and algorithms implemented in their hardware form for performance reasons. Software approaches in this level will only slow node's function down, e.g. the forwarding function.

Programmable part. This part integrates the manufacturer proprietary interfaces of the fixed part and exports an open standardized interface. The APIs are standardized by the IEEE P1520 project [11]. At this level, the node can be programmed but only via a parameter specific approach. The programming can be done, e.g. via an RPC method and it has the advantage that the node always falls into deterministic states. This open interface represents the abstraction of the hardware available resources, ranging from computational resources (CPU, memory, etc.) to packet forwarding resources (bandwidth, buffer, etc.). The NodeOS provides the basic functionality from which the EEs built the abstractions presented to the active applications. The architecture of the NodeOS and its functionality is outlined in detail by the AN Node OS Working Group [6]. Let us mention that the NodeOS could also be a distributed processing environment (DPE) that makes the necessary abstractions.

Active part. The full ability of programming the node is unfolded here as this part hosts several execution environments that allow via code injection and execution sophisticated programmability of the node. Applications that need task specific manipulation of node's states, can implement, in the form of AC, the specific algorithms they need from the scratch, or by combining the services that are available in the node in a Lego-like way. The executed AC, uses also the interfaces that are provided by the programmable part (generic interfaces) in order to access functionality implemented in the hardware part. As also noted [7], the functionality of the AN node is divided among the node operating system (NodeOS [6]), the Execution Environments and the active applications. The architecture allows multiple EEs of various providers to co-exist and be present on a single active node. Each EE (e.g. ANTS [8], ALIEN [9], Agent

EE) exports a programming interface or virtual machine that can be programmed or controlled by third party code. One of the EEs is the mobile agent EE where agents execute when they visit the node. The applications are able to access all the services offered by the EEs. Usually an application is bounded to one EE but we can foresee applications that will take advantage of the various characteristics of more than one EEs and possibly combine their services.

As shown in Fig. 1, one of the EEs is the agent execution environment. This is the agency as described within the MASIF [10] standard. The agent system consists of places. A place is a context within an agent system in which an agent is executed. This context can provide services/functions such as access to local resources, etc. 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. Places can contain other places. All places follow the parent-child paradigm of Unix processes in the way that each child is assigned/makes use of its parent's resources. In addition, its policy is an extension/customization of its parent's policy.

The existence of different EEs for agents (which are the places within the agent architecture) that have the same owner/characteristics serves the need to avoid unwanted interactions. Isolation done by places is similar to the sandbox idea that exists in Java. Since in each place are gathered agents with common characteristics (e.g. of the same owner), the possibility of attacking each other is lower as usual. Of course further security countermeasures [14] have to be taken in order to provide a secure working system.

Cooperating agents reside in the agent-based EEs and via the facilities offered to them program the node. These can be either mobile agents (e.g. visiting agents) or even stationary intelligent ones that reside permanently in the EE implementing various services. The agent can either be generated at a place locally (e.g. out of a pool of ready-programmed objects) or it can just carry on with an execution it suspended in another node.

3. Security support for active networks

Active networking supplies the users with the ability to download and execute code within a node. In such an infrastructure the security implications are far more complex than in current static environments. In ANs the author of the AC, the user who deploys it, the owner of the node hardware, the owner of the execution platform (or even the execution environment) can be different entities governed by different security policies and possibly competitive interests. There fore, in such a heterogeneous environment security becomes an extremely sensitive issue.

3.1. Threat model of active networks and extended security requirements

AC is transferred in some way to the node or is itself mobile, e.g. in the form of a mobile agent. Therefore, the attacks that AC and EE are susceptible to are more than those in current passive networks. In general, we can have:

- Misuse of execution environment by the AC.
- Misuse of AC by other AC.
- Misuse of AC by the execution environment.
- Misuse of AC and/or execution environment by the underlying network infrastructure.

Finally a combination of the earlier categories is possible. This kind of attacks (the complex and collaborative ones), are very difficult to detect not to mention prevent or effectively tackle. Classical examples include the co-operation of various hosts and ACs against another EE or AC.

The AN infrastructures come with a double status; that of a legacy networks (e.g. data transportation) and that of a highly programmable network model adjustable on the fly to application's specific requirements. Thus, the spectrum of threats for such a new network model is extended. It includes not only the threat models of the legacy node and network systems, but also those of general purpose computing engines (e.g. safeness). Privacy, confidentiality, integrity, accountability, non-repudiation, availability, authentication, authorization, and secure communication are requirements inherited from the current passive networks. However, new requirements come into the scene some of which are:

- trusted identification of active node neighbors
- verification of the EE
- secure node-targeted code distribution
- policy based AC management
- AC revocation
- runtime access control of AC
- safe code execution
- prevention of unauthorized EE interactions
- persistence of AC/services/state of the node.
- secure auditing
- predefined controlled node programmability

How our agent based approach can deal with the majority of the earlier threats is explored in our previous work [12]. Here we will focus on an agent-based security architecture that will allow us secure code (and service) deployment for exploiting the advantages of the AN infrastructures.

3.2. An agen-based security architecture

Security cannot be an afterthought. It has to be integrated with the node's functions and not implemented at the end as an extra, optional, explicitly called service. Approaches that try to incorporate security after the design phase have been proven to fail.

The agent system that represents the agent-based EE in Fig. 1 consists of places. A place is the execution environment of the visiting agents. A policy scheme and a resource

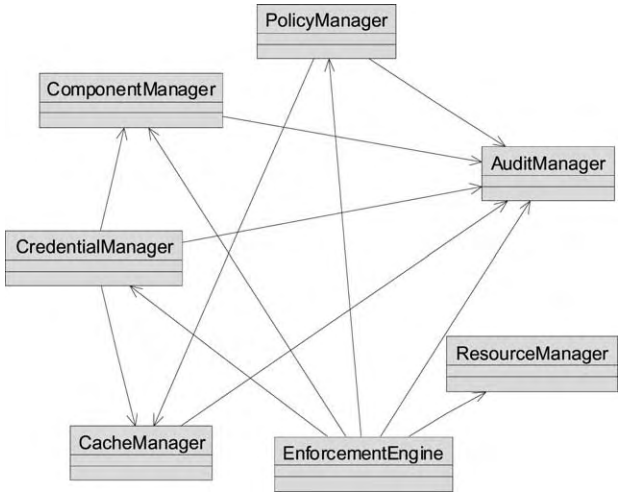

Fig. 2. The components of the security architecture.

access scheme are assigned to each place and the respective policy and resource manager are given the general security guidelines, which can never be bypassed. If an agent has sufficient credentials, then it can fully interact with the components, e.g. change the place's policy, ask for more resources, insert elements in the component database, etc. Places beyond having unique IDs, also hold their own public/private keys. An agent can ask to be signed in order to have a proof that it passed via this place. An overview of the components of the security architecture is presented in Fig. 2 and their functionality will be analysed below.

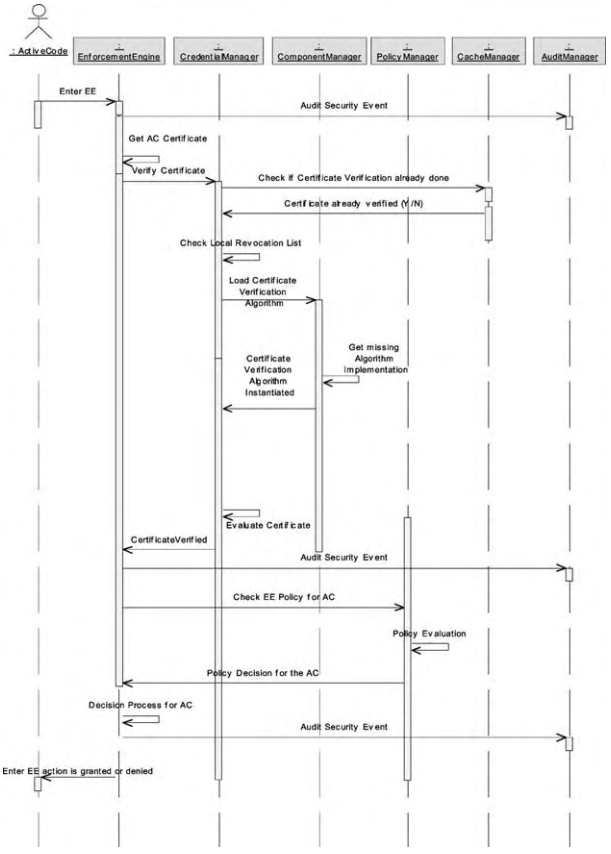

Fig. 3. AC authentication and authorization process while trying to enter in an execution environment.

3.2.1. Policy manager

The policy manager is responsible for managing the policy schemes stored in the policy database. By separating the policy DB from the enforcement engine, we insert dynamicity into the system. The security policy defines the access each piece of code has to resources. Signed code can run with different privileges based on the credentials of the person or place who signed it. Therefore, the trade-off between security and functionality can be tuned.

When an agent comes to an agency, he is subjected first to the general agency's policy, which is set by the user that initiated the agency and is considered to be the super-user. Subsequently, after passing successfully that first layer of control, the agent actions are authorized based on the place's specific policy. It is clear that with this sequential check of policies we avoid the problem of granting contradictory access rights for the same action by different policies. The policy of the 'father' place is always first checked and therefore it has precedence over 'child's' place policy. This hierarchical policy evaluation decision makes it easy for an enterprise to set-up an agency and then provide to the customer places, which are managed by the customer, and do not violate the general policy framework of the enterprise.

Notification of malicious agents (that have attacked other hosts) can be distributed in the network (like CERN security notifications). When our agency subscribes to a security notification service and receives such info it adapts agency's general policy (that is always checked first) so that it will not allow agents bearing those malicious characteristics to migrate to any of the hosted places, e.g. it will not allow migration of agents signed by a user considered as malicious. One can also simply forbid agents from a specific user/domain for personal reasons, e.g. because they consume too many resources, or belong to a competitor company, etc. This is a kind of local black list that in co-operation with the local certificate revocation list provides a higher level of flexibility and customization of the system.

Any attempts to describe the security policy in terms of each individual principal's authority to access each individual object is not scalable and often not fully understandable by 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. Furthermore, by grouping policies we allow for faster execution times while trying to enforce the policy. In our approach all security checks are identity-based (as shown in Fig. 3) in order for an agent to enter a place. After an agent successfully enters a place future security checks become role-based. Thus, we do not have each time to verify agent's

credentials. We check only to see in which place the agent resides and what is the appropriate policy for that place. This approach is followed in the effort to speed up security checks and improve architecture's performance.

3.2.2. Credential manager

Credentials are used to (i) verify that the component was created/distributed by the claiming principals, (ii) verify that the component has not been altered after it has been signed, (iii) fulfill partially the non-repudiation need so that the originator of that code cannot deny it.

Credentials are stored in the credential database. All actions concerning the credentials (including management of the credential database) are handled by the Credential Manager (CM). The CM checks the validity of the certificates, updates them, maintains the local revocation list, etc. The local revocation list acts as a second black list only that this time the user can locally invalidate the agent's certificates and therefore force the system to treat the agent as an anonymous one or reject it. While the first list forbids migration to the agency (via SSL authentication) here we have only sandboxing of the agent (treated as possibly malicious).

X509 Certificates can be used as credentials in a heterogeneous environment with a key used as the primary identification of a principal. It is assumed that users and hosts have certificates. EEs can also (optionally) have certificates in order to fulfill sign requests. As the nested-place approach we take is based on service-oriented logic in which the EE n can belong to a different provider than the sub-place $n + x$, we can ask from the nth place to sign a part of an agent. If that place does not have a certificate, it can use (if permitted by policy) the certificate of place $n - 1$ or if that place also does not have a certificate then that of $n - 2$, etc. Finally, if the host does not have a certificate or somewhere between the policy of place k (with $1 < k < n$) forbids the use of a certificate from parent places then the action fails.

The usage of certificates assumes the existence of a Public Key Infrastructure (PKI) with certification authorities (CA), which issue certificates that bind two principals in a speaks-for relationship. When checking the validity of certificates the CM looks up first his local database and his local revocation list. If the local lookup action returns with an error then via the usage of a protocol, e.g. the online certificate status protocol (OCSP [18]) its validity is checked in cooperation with a CA server.

3.2.3. Component manager

The component manager mainly manages all requests concerning components preinstalled by the administrator as well as user-installed components in the component database. The component manager allows first the administrator to install code and selectively via policy make it available to the users. This code can be signed so that other third parties can verify the originator of the code and decide whether to use it or not. This helps partially with the 'malicious host' problem, as now by verifying the credentials of the AC one can decide if he trusts the code and at which level. Of course, again here the verification process simply points out that the code is the original one and has not been modified but that does not give any guarantees that the platform will execute it correctly. Furthermore, the agents are able to verify a host before they migrate to it. Therefore, if every host n can verify host $n + 1$ then we can be relatively sure that our agent moves in a selected path of trusted hosts. If the host is not trusted then the agent may decide not to migrate and execute there. Of course, the agent can select where to execute but it does not have any guarantees after it arrives to that host, as its execution is controlled by that host's EE. User agents that are given permission can put their own code to the component database and make it available to third party agents permanently or for a limited time. This increases the flexibility as well as the security and performance of the platform. The flexibility and good performance are due to each user having its own implementations of custom code on the node, and thus his agents can be more lightweight and less complex. Security is also enhanced as the administrator will provide all new encryption/compression/etc. algorithm implementations with code he has tested and trusts. As a result, agents do not bring every time their own code, which in turn makes it less risky for the platform to be faced with unintentional side effects (e.g. buffer overflow). Not to mention that the administrator's implementations will be always updated and possibly platform specific optimized, providing therefore better overall performance to the system. This supports also the component-based service composition. The component database can be considered a general database of AC, protocols, encryption algorithms, etc. It can also be used for caching agent's code but its functionality is far more extended than simple caching.

Component database is of great significance to this approach as it ensures the up to date status of various components and in parallel minimizes security risks for agents and for the platform. Security is by its nature overhead in the communication and execution in order to protect the system. We accept that. Yet, there are novel general ways/techniques to minimize this overhead (under certain conditions) and fortify the security on the node. In the future, more specialized techniques that take optimal advantage of the underlying network resources could be used if this approach is to leave the research domain and enter the commercial one.

3.2.4. Resource manager

A resource manager is available in order to handle the resources assigned to the agency or place. We assume that resources are assigned from the administrator (that is the person that creates the place and this can be the agency administrator or one of the previous $n - 1$ place administrators who created the nested place n) to a place n and are managed by the owner of the newly created place. The resources and their management (static or dynamic) are

transparent to place users and to nested places that place *n* might contain. The place resource manager can handle the resources that are dedicated to a specific place.

Note that the resources available to a certain place are transparent to its users. That means that local resources could be extended via CORBA in order to access resources in other nodes. With this idea in mind, one could consider network-wide working space and resource consumption (e.g. distributed disk space). This helps also with the Place Oriented Virtual Private Network (PO-VPN) [15]. In a PO-VPN scenario an enterprise can setup places spawned in a network infrastructure and therefore create a VPN of places where its agents can execute according to custom security policies and services. The transparency of resources across multiple agencies which host places that belong to a VPN or a third party entity, and the applicability of the MarketNet [14] idea in order to provide a dynamic resource-driven security scheme, offers new hardly scratched ground for further interesting research.

3.2.5. Cache manager

The cache is another essential part of the architecture and its usage is mainly focusing on improvement of the overall performance. Security checks are time and computing consuming processes. In our effort, not to duplicate all the time the necessary security checks, we have a cache. 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 they are considered still valid and used by the system. Although this approach may speed-up authorization decisions, it leaves a controllable window of error since the authentication status of the AC may have changed.

In that case, the cache contains outdated information. We solve this problem by deleting (this is repeated each time the policy for an entity changes) the cached security checks that are associated with this key/person partially or completely. Therefore, the next time that 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 system.

3.2.6. Audit manager

Audit manager handles all audit events. Experience has shown that 100% security is at least difficult to realize (if not impossible) due to the multiple factors that interfere. Collecting data generated by network activity provides a useful tool in analyzing the existent security and trace back (if possible) the originators of a security breakout. Having a detailed audit can lead to reconstruction of a sequence of events and better understanding of past security failures. 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 an attempt to use a component/service we can adapt our policy 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. Unfortunately, not all activities can be monitored. Furthermore, these logs are usually plain text files, which introduces further security risks (acquirement of private info, alteration, etc). Thus, the log files should be protected with a computationally cheap method [13], which will make impossible for the attacker to read and in parallel impossible to undetectably modify or destroy. Survivability of audit info is often neglected but is a must for secure network infrastructures.

3.2.7. Enforcement engine

The Enforcement Engine enforces the policy on the agency in general and on the places. It is the front-end environment, via which users interact with the architecture. An Enforcement Engine must satisfy three important rules. It must be (i) always invoked, (ii) tamperproof (iii) verifiable. We try to fulfill the earlier requirements by implicitly checking access rights to all systems resources, signing the components and loading the basic parts of the architecture securely. The host/agency/place administrator can either edit the policy and credential data prior to system run or interfere dynamically during system runtime via agent interface. The enforcement engine we have heavily depends on Java's security architecture.

3.3. The language choice

Selecting a language for ANs is not a trivial issue. Tradeoffs between security and performance are critical parameters in the choice of a language especially if this language is to handle user-injected general-purpose code. If ANs were to operate in a completely trusted environment then any modern rich in features programming language would be appropriate, but this is not the case as we deal with a heterogeneous untrusted environment. The biggest problem in AN is security. Thus, all decisions in designing/operating it should be made with security in mind. Therefore, we require a language that can have some special characteristics such as:

Strong typing. This means that a program cannot arbitrarily access the host computer's memory. Memory access is limited to specific controlled areas having particular representations. Thus in such a language common programming errors are avoided.

Garbage collection. Of course each user/agent can manage the memory (allocate/de-allocate) he is assigned. Lets suppose that he frees some memory blocks (memory looses its type) and that these blocks are reallocated to another agent. Then this agent is able to read the data on that memory blocks and acquaint info (possibly security critical) about the operations of the previous user of that memory space. With automated garbage collection, we

make sure to avoid such problems associated with dangling pointers.

Access controllable module view. It allows us to view a module via multiple interfaces. That gives us the ability to deploy a rich feature module that provides different capabilities to different users. In this way we are able to modify flexibly who can see/do what and how.

Dynamic loading. It is desirable, as we want to make modifications and load new functions/capabilities while our system is up and running. It is out of question to shutdown an AN router every time we want to update a software component or provide a new capability. Furthermore, by dynamic linking it is easier to keep our system up-to-date since the latest version of code and libraries is always used.

Communication support. The language should have its own optimized libraries for basic communication between the applications and of course network support. Object communication, programming with sockets, establishment of URL connections, etc. are mandatory in a networking environment.

Widely used and evolvable. These are non-technical characteristics of the language we need. This is not for commercial/political reasons but for practical ones. A language used by a small group of people might be task-specific but it would be difficult to advance and keep up to date as bugs, errors and misbehavior would be seldom if at all reported. Thus, we need a language that is widely used so that it evolves fast and day-by-day new features are added depending on the needs that pop up.

Platform independence. It is not mandatory but would be of great help since our efforts could be ported/deployed easily to a heterogeneous environment such as that of AN.

Having in mind all the above one could design a new language tailored to the needs of active networking and our system. A small sample of difficulties he/she would face is:

- designing from the scratch a new language with a bunch of desired features as mentioned earlier (e.g. safety, performance),
- if we do not manage to address all required features needed by the user it would be impossible for user to implement the he wants,
- it would require a huge amount of work to keep the language up-to-date with increasing demands,
- it would be used by a limited number of people (AN people only) and therefore bugs, errors, etc. would be seldom if at all reported.

The other approach is to use an existing language. Java is not an AN specific language itself but covers reasonably our requirements. Java is a very popular language designed especially for mobile code and most important with security in mind. It supports dynamic code loading, concurrency, communication between networking applications (http, sockets, RMI, etc.) and security services. Java nowadays is used extensively not only in research domain but also in industry. Therefore bugs, errors are found and reported fast. As it is a commercial product, it advances and day-by-day new features/libraries are added. Furthermore, Java offers platform independence which is a significant factor as it assures portability within a heterogeneous environment such as the AN infrastructure. That in addition with the support of object oriented concepts like polymorphism and inheritance make the development of active components easier, as these components are seen as an abstract object of code to be transported and installed in an environment no matter of the underlying architecture.

Of course Java is not panacea. It has several problems, e.g. performance weaknesses, security holes due to false implementations, etc. However, these weaknesses can be addressed reasonably in the future, e.g. the performance can be enhanced with just-in-time compilers and HotSpot technology [16] coupled with ever-faster processors, and therefore they should not be an obstacle to the deployment of java-based approaches. Bottom line is that the Java language is good enough for our needs and maybe will get even better in the future.

4. Dynamic virtual private network deployment with nomadic user support

A VPN is a communication environment in which access is controlled to permit peer connections only within a defined community of interest and is constructed through some form of partitioning of a common underlying communications medium, where this underlying communications medium provides services to the network on a non-exclusive basis. Such kind of networks are deployed within a public network and aim at providing a private working environment to its users while also taking advantage of the efficiencies of the underlying infrastructure. Current VPNs are designed mostly with static users in mind and little has been done to easy integrate mobile users or to provide mobile user support after their deployment.

Nomadic users are wanderers, people on the move from place to place. The goal is to make information services and applications ubiquitous and flexibly available for such individuals as well as to small groups of them. The problem that arises is that the need for—and the availability of—information and communication services vary from place to place and from time to time. Key requirements are the (a) rapid service adaptation and customization and (b) security. We are mostly interested in two categories of nomads:

An individual or a group of individuals moving together. The aim here is to maintain the local context as the group as a whole moves. Services provided to the group should be the same even though the group or the individual (group with one member) is away from the home environment. A military squad in a battlefield falls within this category. Here the connections between the group members may be

intact and only the underlying infrastructure changes. Thus, the services provided to the group have to adapt to environmental changes and this should be done without any noticeable changes to the upper levels.

A distributed context with autonomously moving members. The aim here is to keep a virtual community and its context intact by rapidly adapting to the new environmental parameters that are generated by the move of its members. A multi-conference between mobile users falls within this category. In this scenario the matters get more complicated as two parameters change (i) the underlying infrastructure (ii) the virtual connections among the members of the group.

Later we will try to address the requirements that need to be satisfied, and then we will demonstrate how agent-based AN in combination with the security architecture we have presented, offer an open evolutionary approach to the support of secure nomad-aware VPNs.

4.1. Portability requirements

From the portability perspective we have user, home environment and serving network requirements:

User requirements. The user wants to freely move in heterogeneous environments and be able to customize the services offered, to personalize the user interfaces based on terminal's capabilities, have ubiquitous access to all services offered to him independent of his location, the ability to modify his profile, activate and deactivate services from any location, to be able to discover the additional services in the new environment and have all of the above in an optimized and cost-effective way.

Home environment requirements. The home environment wants to provide a high customization of its services to the users it hosts. It wants also to provide an easy way to make these services available even when the users roam in third party networks. Access to the services offered should be policy-based and fine-grained according to different parameters (or a combination of them) e.g. user's credentials, current location, or even based on the foreign network's ID.

Serving network requirements. The serving network may offer to its temporary visitors access to some of its capabilities or merely provide a tunneled connection to the home environment. The challenge here is to provide the visitor with transparent access to the services he subscribed in his home environment and additionally offer him new services not available in the home location. Billing and management of the visitor users is non-trivial issue especially when the end-user services (or the generic components that they are composed of) spawn different provider domains.

4.2. Operational context for nomadic users

The concept structure of the Agent DPE has to be applied into the functional framework of the nomadicity. Each user can be considered as acting within his own EE. This EE (that is under the total control of the user) hosts one or more intelligent cooperating agents that keep track of the user's needs and current status. Additionally these agents are responsible for mediating and managing the services that are offered by the underlying infrastructure to the user as an individual or as member of a group. When the user moves from one network point to another, the agents are responsible for providing optimal adaptation to the new environment and reconnect/reconfigure the services that the user needs, in order to provide the same (not only in look and feel but also in functionality) working environment as before. The whole process should be transparent as possible to the end users.

A VPN with nomadic users constitutes a graph with changing nodes (due to mobility requirements). The challenge is to re-assign the connections between the nodes of the graph in order to provide the same services in a higher level despite of the fact that the underlying infrastructure continuously changes as the nomadic users move.

We believe that agent technology in combination with the active/programmable networks is the right step to this direction. Agents can also be intelligent, which means that they can adapt easier to non-deterministic environmental changes, learn while they are active and act proactively in order to satisfy their internal goals. In the following scenario, we will firstly show how agents can be used to deploy the initial VPN among the end users and subsequently we will consider two users as nomads and explore the infrastructure adaptation while these users change location and terminals.

4.3. Initial VPN provisioning

Our objective is the dynamic and flexible provision of legacy VPNs, allowing the deployment of a VPN a) in minimal time and b) with minimal user interference. Therefore, agents take the roles of the customer, the service provider and the network provider. The agents are digitally signed by their users, a proof that they are delegated to perform the specific tasks. There are multiple network and service providers in the infrastructure on top of which we want to build our VPN. We assume that the Group Agent (GA) interacts with the VPN Service Provider Agent (SPA) and with the Network Provider Agent (NPA). The SPA has to negotiate with all User Agents (UAs) and NPAs. If a specific service provided by a service provider, spawns multiple domains, then the service provider makes the negotiations concerning the service with those domains. The stages to follow are:

Common requirement definition stage. In this stage the UAs negotiate and come up with a common set of requirements for the underlying infrastructure and desired services. The UAs assign as responsible for the further negotiations a GA and all sign the common set of requirements.

VPN Network negotiation stage. At this stage the GA negotiates with NPAs the possible topology of the VPN and the requirements in the networking infrastructure.

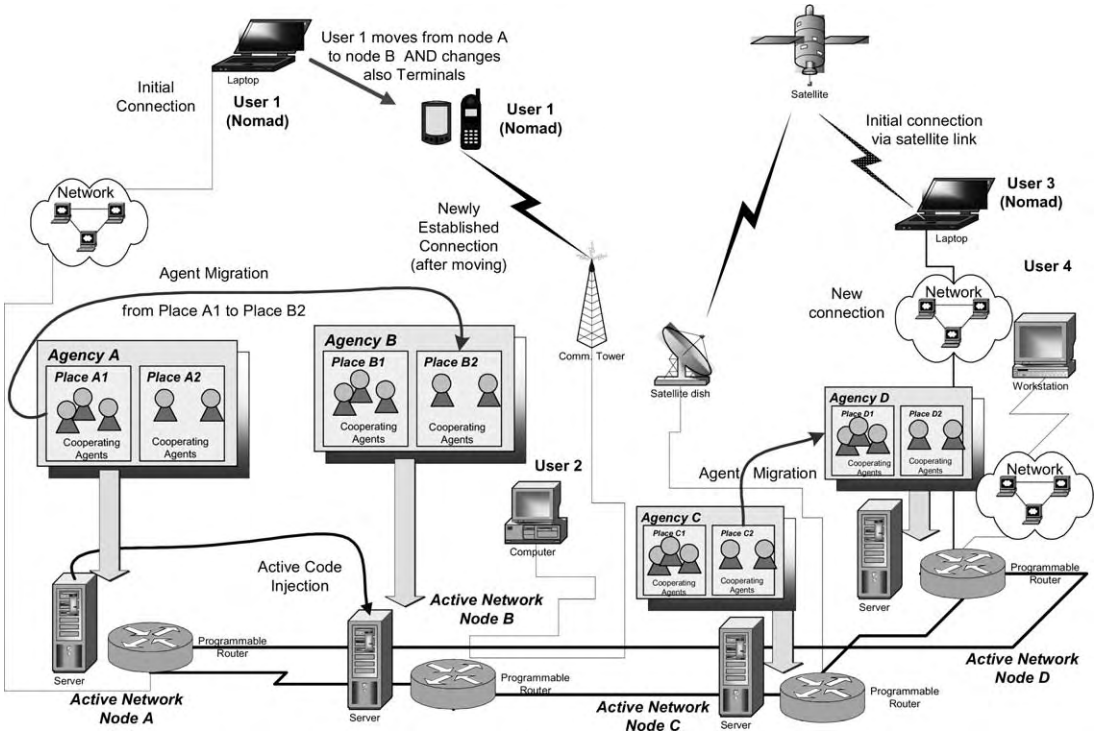

Fig. 4. Nomadic user support in an agent-based AN infrastructure.

This is done in order to accommodate user specific requirements concerning network infrastructure e.g. routing of flow via distinct nodes.

VPN service negotiation stage. In this stage the GA has multiple network topologies that fulfill group's requirements. Subsequently the GA negotiates with SPAs in order to see which of the possible topologies support the desired services. The service provider, can also offer alternative solutions concerning the network topology, etc. At the end of this stage the GA has some network topologies that support the services he requires and a set of service level agreements (SLAs).

VPN selection of the final network topology. The GA either returns and reports the possible solutions to the group of UAs or decides by itself for the best of the offered solutions. The final decision is made based on various facts, e.g. connectivity bandwidth, error statistics, reputation of nodes contained in the topology, cost, security, etc. either by a human or the GA itself assuming that he is intelligent enough. Finally, a network topology is selected and the final phase of agreement has to follow.

Deployment of the VPN and its services. The GA requests that service and network providers set-up the services and the network connections, respectively. When everything is done, the GA informs the UAs that the VPN is deployed and fully functional (all services at group/individual level are instantiated). The GA either terminates at this stage or can be used as a central authority for future requests regarding the VPN's services, topology and status.

4.4. Dynamic virtual private network adaptation

After setting-up our VPN, group communication services can be deployed. For example, such an infrastructure is depicted in Fig. 4 where users on domains A, B, C and D are connected with each other. VPN members may span various network providers as some of them rely on the home network but others are roaming in foreign networks.

We will now examine what happens when the members are on the move. Lets suppose that a user that resides in domain A moves to domain B and also changes his terminal from a laptop (device with advanced capabilities) to a PDA (a low capability device). The user had a teleconference in his laptop, which he wants to continue with the least possible disturbance in his new location (domain B).

The user's move typically in our infrastructure means that he has to be registered within the new domain and be provided at least with the same service quality as before taking into account the equipment change. For that specific user, domain A is the home network and domain B is the serving network. The user movement implies that the agents providing the user with all the services move from execution place A1 in node A to execution place B2 in node B and resume their execution there after of course registering with the local node and adapt to the new environment. Although the steps are not strictly defined in such a scenario, generally the following abstract actions take place:

- The user is ready to move. This can be an automatic event (e.g. in a mobile device because the signal of the nearby

communication tower is stronger) or a result of broken communication (e.g. the connection was terminated because of a communication hole or satellite technical problem). In any case, the user's agents receive an 'operations stop' signal from the system's agents.
- After receiving this 'operations stop' signal, the agents shutdown the services they provide to the user. They also notify the GA (which acts as a central information registry) that the current user will change his network position and all communication is temporarily suspended.
- The new destination address of the user in domain B is available to the local agents (still in domain server A). This can be done in advance (if the user move is apriori known) or is sent to the agents the moment the user tries to log into the new visiting network B.
- Having obtained the new destination address, the agents migrate to the appropriate active host in domain B where they are subject to the authentication control of the domain provider. Since each active node features a security architecture as the one described in Fig. 2 the actions that follow in order to allow the agents to authenticate themselves and enter the EE of the foreign network are similar to the ones described in Fig. 3.
- Having successfully authenticated themselves they resume execution in the new node B in the visiting network.
- The user profile is consulted (this can be carried by the agents or even retrieved from the home network) in order to see what are the services the user is subscribed to and how they should be personalized.
- Subsequently user's agents co-operate with the agents of the local node in order to retrieve the services supported by the node B for the visiting domain. The services that are the same with the home network are configured with the user's preferences and are activated. For the services that do not exist locally, the agents of node B are asked to tunnel services from the home network. That assumes agent-to-agent communication and cooperation between the two domains A and B. If it is allowed by the policy of node B, the agents can download the AC from a code server in the home environment that implements the missing services in domain B and install them on the fly on the active node B. This is a very important step as it truly demonstrates the power and dynamicity of the VPN that is based on agent technology and active/programmable nodes. The possibility of downloading and installing code on the fly directly into the visiting environment is possible via the active networking technology in a flexible and secure way. The code is stored in the component manager (Fig. 2) and is instantiated. Further authentication and authorization requests are handled by the security architecture and the service itself.
- After having set-up everything they announce the new user place as the new part of the VPN network and inform the GA. Subsequently the GA multicasts the new VPN node to all affected UAs so that they reconfigure the local services to comply with the new topology of the VPN.

At the end, all this functionality is presented to the user (via the form of the services and automatic configuration). The effort is to have everything transparent and with minimal human intervention. For the end-user in our scenario, it means that he can continue his teleconference uninterrupted as the agents on the back ground have taken care of this environment change. Code injection to a foreign network is not a trivial issue and is the driving force behind the AN community. However, by mixing both the advantages the agents provide (e.g. AC carriers, service implementers, intelligent handling of non-deterministic events, etc.) with those of AN (e.g. dynamic code injection, reconfiguration of the routers via open interfaces, etc), we can have an open and flexible approach to the problem of nomadic user support in VPNs. The earlier scenario is abstract and demonstrates only one of the many ways that the agent-based AN infrastructures can be dynamic and proactive. Finally, implementing the earlier functionality is not a trivial issue, especially due to the fact that the standardization activities have not advanced that much, in order to fully support via standardized interfaces all the interactions described earlier.

5. Evaluation

We have presented an approach that tries to integrate the advantages of the agent technology and those of the AN. This can offer numerous advantages to the telecom providers as explained below:

Rapid service deployment and customization. Current network elements are closed to third party entities, therefore introduction of new services is equipment dependent and is additionally slowed down by the lengthy standardization process. However, ANs feature open standardized interfaces at various levels, which can be used to deploy market-driven services. Furthermore, these services can be modified on runtime. This on-the-fly modification of the service behavior can be exploited in order to offer customizable services to the end users.

Decentralization and autonomy. Many tasks/applications require a continuously open connection and a fixed network topology. Agents do not have that requirement and therefore ANs can benefit from it. Agents are able of working autonomously and in a decentralized manner. They exploit the locality and achieve optimization in the usage of resources that are offered in that location. Thus, problems such as unpredicted network latencies in critical real-time systems (e.g. robots in a manufacturing process) can be avoided. In addition, by using agents we don't have to develop new transport mechanisms for the deployment of active components to the nodes.

Flexibility. Users are able to launch agents and customize

services easily. Most important, the user does not have to be online all the time since he can send his agent and then disconnect from the network. The agent carries certificates from the originator, acts autonomously and tries to satisfy the user request. The agent handles all interactions intelligently. Furthermore, agents provide mechanisms for monitoring, logging, updating, etc. which can ease tasks like administration/management of an AN node.

Adaptivity. Changes in the environment in which an agent operates trigger possible changes to agent's behavior. An agent is capable of sensing the environment, analyzing the new data and acting accordingly. For instance, a group of agents monitors the consumed resources in a network. These agents can exchange information and traverse the network in order to get a global view of the network's state. Depending on their goals as well as their capabilities, they can interfere and, e.g. change the routing tables of hosts in order to provide better exploitation of the available bandwidth. This adaptivity promotes the optimal network performance and the handling of non-deterministic events.

Interoperability. Current network infrastructures are heterogeneous both in hardware and in software matters. Agents are computer and transport independent entities (they depend only on the execution environment) and therefore promote interoperability among systems and software. It is possible with agents to implement interactions with any legacy systems and currently existing services, and make it available to other heterogeneous agents or applications in a standardized way. Standardization efforts in the agent domain exist within the Object Management Group[1] and the Foundation for Intelligent Physical Agents[2].

Security. ANs allow users to inject their code to the node, which is a security critical activity. At a certain level, these issues have been addressed and solution exist within the agent community. Agent technology is capable of providing authentication, authorization, integrity-check and privacy mechanisms to AN managers so that they can have control over the network and its resources. Security is generally provided by exploiting application level services. Agents act on behalf of a user/entity/enterprise, etc. and carry some credentials (e.g. signed by the owner or the originator, etc.) and based on these credentials and the local policy a security manager could have control on the agents themselves and their payload. In addition, as the research area of security in agent systems is a hot research domain, we expect in the near future more and better security solutions, which we expect it will be easy to import in the security architecture for ANs we have presented in this paper.

Scalability. This AN-based architecture is a decentralized one and can scale easily. AN nodes can host an agent system that could be low or high populated by agents that act and interact with each-other providing services and advanced features.

Safety. The usage of Java as an implementation language offers some security and safeness level. Furthermore, the idea of places acting as sandboxes avoids unwanted interactions within an agent system.

Performance. Our approach is based on Java. Java based systems for the moment lack performance. We expect this to change in the near future as efforts are being made in this direction. Nevertheless performance can be achieved/enhanced from our side, with other techniques such as component (agent code, protocol, algorithm, etc.) caching, etc. From AN point of view intelligence is added at a network level offering the ability for exciting network wide applications that can configure each end every node for optimal application and overall performance. ANs may perform actions that on first glance appear to degrade network performance (e.g. lower packet throughput) but actually, they bring improvement to the application and to the network itself by reducing demand of bandwidth at endpoints, reducing network congestion, etc.

Robustness and fault tolerance. Mobile agents are programmed with their internal goals and logic. Taking also into account their ability to react to the changing environment and unpredicted situations, it makes easier to design and implement robust and fault tolerant systems.

Software independence and evolvement. Current distributed systems exchange data via a standardized way (protocols). Each node owns the hardware specific code that implements the protocol needed to communicate with the outside world. However, the protocols and their supported features evolve and often we face the problem of outdated protocol versions, which are inefficient and insecure. Agents on the other hand can help effectively with this problem. They can move to remote hosts and establish 'channels' based on protocols that are task specific and not even standardized. Furthermore, they can update node's components automatically, therefore keeping our infrastructure always up-to-date since all network components will be updated in parallel shortly after the announcement of a component update by the manufacturer. In addition to that, the AN concept has a fundamental difference with current telecommunication networks: in AN the functionality to be implemented is prescribed via standardized APIs but not the way this functionality is implemented. This decouples the network platform provisioning from the network software provisioning and therefore makes telecom operators less dependant to a specific vendor; a fundamental issue in a multi-vendor competitive environment where a software vendor A develops innovative services for vendor's B network platform.

Deployment of both active and programmable network models. The agent-based approach can be complementary technology in the effort to integrate [19] active and programmable networks. Furthermore, it allows a high-level system design including business aspects. The market is moving towards a service orientation and agents can fit well as service-oriented software. Agents have a natural place in the application model as (i) wrappers of legacy

[1] OMG Web Site: http://www.omg.org/.
[2] FIPA Web Site: http://www.fipa.org/.

systems or as embedded smart systems, (ii) as powerful middleware that glues together distributed components, (iii) as intelligent and adaptable interfaces that support online/offline user interaction.

Active and programmable networking is a good area to apply the agent technology as they will benefit on all the earlier mentioned sections.

6. Conclusions

An agent-based active node architecture has been presented. This approach uses agents with different features, e.g. mobile, stationary, intelligent, goal oriented, etc. to empower the current passive routers and to transform them to AN nodes. We have showed that agent technology is a promising candidate for the development of AN. It offers benefits to the users as well as the developers of ANs including flexibility, adaptivity, decentralization and autonomy, interoperability, scalability, security, etc. Furthermore, their double status that of AC actors or of AC carriers can ease matters such as service deployment, authentication and authorization issues, network safeness, etc.

We have explored a scenario where VPNs can be flexibly deployed via cooperating intelligent agents and in parallel make use of the AN technology in order to become more nomadic-user aware. Nomadic users are part of groups that have requirements that change unpredictably by the time especially due to the fact that their users move constantly and spawn heterogeneous infrastructures. This kind of groups are interested in flexible VPN's that immediate react to environmental changes. Such groups cannot use a static version of a VPN to cover their requirements as they are not built with mobility in mind and they are difficult and awkward in reconfiguration requests. Furthermore, those groups' lifetime is short and usually determined by some other external events. Such groups need to set-up and delete VPNs in minimal time. This flexibility to create and teardown such a virtual environment can be provided with the approach described in this paper.

Agents can be used to deploy new services and program the nodes according to application's needs. By updating the underlying infrastructure's components on demand and by reusing in a Lego-like way the local services, agents are able to provide more sophisticated personalized services. Stationary agents that reside on the nodes not only offer their services but also respond to the environment changes—which may be unpredicted—by reconfiguring or updating node's components.

It is very likely that agent technology will play an important role in the development and expansion of the AN. The basic characteristics of mobile agents such as mobility and autonomy can push networks to become more 'open', active and more powerful. By integrating agents we also make sure that in the future we will be able to import the state-of-the-art agent technology in our network and that simply means that our infrastructure will keep evolving as long as it is connected with this parallel developing domain.

References

[1] Active Networks at DARPA, http://www.darpa.mil/ito/research/anets/.
[2] Mobile Agent Technology, http://www.cetus-links.org/oo_mobile_agents.html.
[3] C.G. Harrison, D.M. Chess, A. Kershenbaum, Mobile agents: are they a good idea? Technical report, IBM Research Division, T.J. Watson Research Center, March 1995.
[4] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, G.J. Minden, A survey of active network research, IEEE Communications Magazine 35 (1) (1997) 80–86.
[5] H.S. Nwana, D.T. Ndumu, A brief introduction to software agent technology, agent technology, in: N. Jennings, M. Wooldridge (Eds.), Foundations, Applications and Markets, 1997.
[6] L. Peterson (Ed.), Node OS Interface Specification, AN Node OS Working Group, January 24, 2000.
[7] K.L. Calvert (Ed.), Architectural Framework for Active Networks, Draft version 1.0, July 27, 1999.
[8] D.J. Wetherall, J. Guttag, D.L. Tennenhouse, ANTS: a toolkit for building and dynamically deploying network protocols, IEEE OPENARCH'98, San Francisco CA, April 1998.
[9] D. Scott Alexander, ALIEN: a generalized computing model of active networks, PhD Thesis, University of Pennsylvania, December 1998.
[10] MASIF—Mobile Agent System Interoperability Facility, http://www.omg.org/docs/orbos/98-03-09.pdf.
[11] IEEE P1520 Project, http://www.ieee-pin.org/.
[12] S. Karnouskos, Security implications of implementing active network infrastructures using agent technology, special issue on active networks and services, Computer Networks Journal 36 (1) (2001) 87–100 ISSN 1389-1286.
[13] B. Schneier, J. Kelsey, Cryptographic Support for Secure Logs on Untrusted Machines, The Seventh USENIX Security Symposium Proceedings, USENIX Press, 1998.
[14] Y. Yemini, A. Dailianas, D. Florissi, MarketNet: a market-based architecture for survivable large-scale information systems, Proceedings of Fourth ISSAT International Conference on Reliability and Quality in Design, Seattle, WA, August 1998.
[15] S. Karnouskos, I. Busse, S. Covaci, Place-oriented virtual private networks, in: Proceedings of the 23rd Hawaii International Conference on System Sciences IEEE HICSS-33, Island of Maui, Hawaii, January 4–7 2000.
[16] Java HotSpot Technology, http://java.sun.com/docs/hotspot/PerformanceFAQ.html.
[17] A.V. Vasilakos, K.G. Anagnostakis, W. Pedrycz, Application of computational intelligence techniques in active networks, Soft Computing Journal 5 (4) (2001) 264–271.
[18] X.509 Internet public key infrastructure online certificate status Protocol—OCSP, IETF RFC2560, http://www.ietf.org/rfc/rfc2560.txt.
[19] S. Karnouskos, H. Guo, T. Becker, Trade-off or invention: experimental integration of active networking and programmable networks, special issue on programmable switches and routers, IEEE Journal of Communications and Networks 3 (1) (2001) 19–27 ISSN 1229-2370.