A Security Policy System for Mobile Autonomic Networks

Mohamad Aljnidi
CNRS - UMR 5141 (LTCI)
TELECOM PARIS - INFRES Department
37/39, rue Dareau - 75014 Paris - France
+33(0)1 45 81 71 56
mohamad.aljnidi@enst.fr

Jean Leneutre
CNRS - UMR 5141 (LTCI)
TELECOM PARIS - INFRES Department
46, rue Barrault - 75013 Paris - France
+33(0)1 45 81 78 81
jean.leneutre@enst.fr

ABSTRACT

An autonomic security system is indispensable for the operation of an autonomic network. Policies are basic stones in building autonomic systems. In this paper, we introduce our model of mobile autonomic networks. Accordingly, we propose a security framework for building autonomic security systems. In this framework, we discuss a trust model based on node communities, an authentication model based on node categorization, and a secure relation model based on both trust and node capabilities. A network evolution model is eventually presented as the working context of an autonomic security system. Afterwards, we explain our vision of autonomic policy systems, and relevantly present solutions for security policy representation and manipulation. Finally, we define an authorization model for mobile autonomic networks, before elaborating an example of the implementation and the negotiation of a relevant access control policy.

Keywords

Autonomic Computing, Ad-Hoc Networks, Autonomic Networks, Security Models, Security Policy Management.

1. INTRODUCTION

In most of nowadays complex systems and environments, traditional human-driven management is not efficient any more for reaching certain required performance or behavior. For instance, in mobile ad hoc networks, it is often desired to have a continuous adaptation to unpredicted changes in the topology, the population and the high-level reconfiguration performed by end-users. It is virtually impossible for humans to traditionally manage the different systems of the mobile ad hoc network or its components, to get such behavior. This is due to the complex nature of mobile ad hoc environments, represented by many aspects, such as the heterogeneity of devices and technologies, the lack of infrastructure and the decentralized structure. In such cases, relevant systems should be able to reconfigure themselves, and when necessary to optimize themselves. Moreover, they should be able to protect themselves, and in case of damage, to survive by themselves. In other words, they should be self-managing systems, which are able to monitor their contexts and environments, detect unpredicted changes in them, analyze the detected events and the related gathered information, identify the sources of anomalies, and accordingly adapt their configuration, immunity or functionality, or eventually repair themselves.

The need for self-management solutions had already been recognized by several specialists in different relevant domains, and many corresponding initiatives were launched, as elaborated in [5]. In our research, which is in conformity with the initiative of IBM [13], we study the realization of Autonomic Computing properties as defined in [6], depending on high-level policies, to build an autonomic security system for mobile ad hoc networks. We believe that seeking autonomic security solutions is a necessary step towards autonomic networks, rather than dealing with security as an afterthought, as usually happened in most of the research efforts of computer networks. As explained above, a mobile ad hoc network can be complex enough to achieve its desired functionality depending on human administrators only. Therefore, it would rather depend on autonomic systems as well, if not exclusively, at different communication levels, and in terms of hardware, middleware and software. Nevertheless, complexity is not our only motivation for autonomically securing mobile ad hoc networks. We consider as well specific characteristics, such as the potential large-scale nature, the mobility, and the possibility of assuming high-level administration tasks by non-expert users.

Different aspects of autonomy, such as self-organization [1, 19], self-adaptation [18] and spontaneous behavior [7], were already addressed in certain types of networks. These latter were not originally created as autonomic networks. Environment requirements, application goals or network eventual behavior were, among others, the reasons behind implementing autonomic solutions in later optimizations. Relatively recent studies tried to define what an autonomic network is, such as [17]. In general, the field of Autonomic Communications [14] raises challenges for interesting future research efforts. We intend to contribute to the future solutions in this domain, and as a basis for our research, we propose the following definition for an Autonomic Network: it is a network that can evolve autonomously, in terms of population and topology, and its evolution is self-controlled by a set of autonomic systems, which constitute together the autonomic manager of the network. An autonomic system of an autonomic network is supposed to detect unpredicted

AUTONOMICS 2007, 28-30 October 2007, Rome, Italy

DOI 10.4108/ICST.AUTONOMICS2007.2205

changes to population or topology, which are relevant to its context, and eventually adapt itself and the network to cope with those changes. Our ultimate goal is to develop an autonomic security system for mobile autonomic networks.

Our model of mobile autonomic networks is based on a specific type of wireless mobile ad hoc networks. According to this type, a network is created without a preexisting infrastructure, and evolves in an ad hoc manner in terms of population and topology. Its nodes are not supposed to be homogeneous devices in computing performance, storage capabilities or electrical power availability. It can employ a multitude of underlying networking technologies. It is not supposed to be managed by expert administrators. Subnetworks can be exported for certain periods, and reintegrated in the mother network when they are back. We consider this as a centralization functionality that we impose in a supposedly-decentralized type of networks, and we call it semi-centralization. In brief, we call MAutoNet (Mobile Autonomic Network) a semi-centralized, wireless, mobile, ad hoc, autonomic network of heterogeneous nodes used by non-expert users. This can be for example a home wireless network, a SOHO wireless network, a business meeting spontaneous network, an emergency service ad hoc network, a mobile sensor network or a military tactic ad hoc network.

MAutoNets have the same security requirements as those of conventional networks, in addition to the security needs specific to wireless mobile ad hoc networks. Therefore, we can say that security solutions proposed for wireless, mobile and ad hoc networks might be implemented in MAutoNets, with the necessary adaptation wherever needed. However, existing security solutions, such as those proposed in [8, 3, 2], do not appear to be compatible with the behavior required in autonomic networks. Even if certain of the existing security solutions have self-management aspects, such as those proposed in [4, 16], they generally aim at realizing an autonomic behavior in certain components or services, rather than being developed for a network built on autonomic basis. We are therefore working on new security models, architectures and protocols for MAutoNets [10, 9, 11], to propose a security framework for designing autonomic security systems, to be implemented in the future autonomic networks.

2. SECURITY FRAMEWORK

In this section, we introduce in brief some of the features and components of the security framework that we propose for MAutoNets. This will help understanding the security policy system and example, elaborated later in this paper.

We believe that a specific security architecture, designed on basis of autonomic computing, must be embedded in a MAutoNet node. We propose here an Autonomic Security Architecture, which is designed to be compatible with the heterogeneity of nodes, transparent to the end-user and irrespective of the underlying networking technologies. Its main components, as illustrated in figure 1, are the following: 1) *Security Agents:* a set of software agents providing security services, such as data confidentiality and integrity, and self-management support, such as a security policy negotiation module. 2) *Security Management Kit:* a set of management modules that can be used either by a human administrator to perform ordinary planned management tasks, or in an autonomic context to perform self-management tasks. 3) *Autonomic Security Manager:* the autonomic security engine, which is responsible of self-management tasks on the node

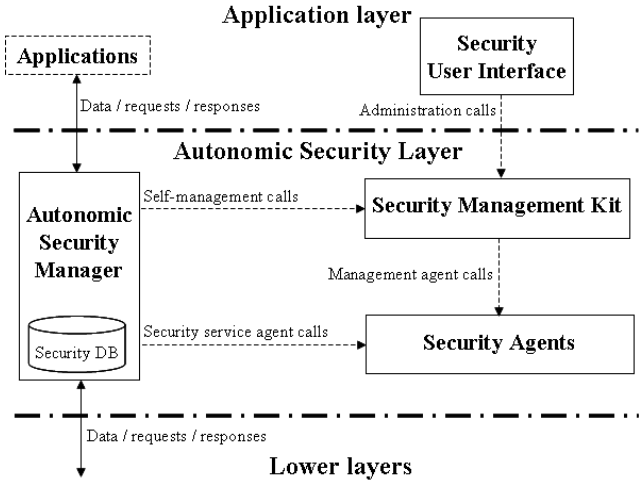

Figure 1: Autonomic Security Architecture

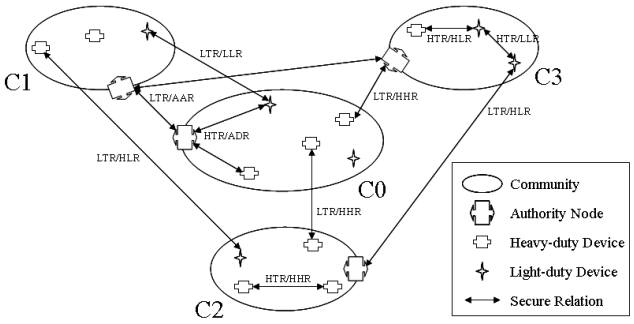

Figure 2: Virtual Security Structure

level, and on the network level in certain nodes having special roles. It is also responsible of the secure data exchange between applications of communicating nodes. Moreover, it manages the security database of the node, which provides its device capability parameters, such as storage capacity, and saves security materials, such as key information. In certain special nodes, this database stores also network security management data, such as network evolution management policies. 4) *Autonomic Security Layer:* an application-support layer encapsulating the previous three components. 5) *Security User Interface:* a set of user-friendly configuration and specification languages, to be used by security expert administrators, and to a certain extent, by non-expert end-users in the form of high-level tools.

We propose for MAutoNets a trust model built on a mutual trust between each couple of nodes. This trust is established once a secure relation is created between two nodes, and remains as long as this secure relation lasts. MAutoNet secure relations are presented later in this section. The nodes of a MAutoNet are virtually distributed on a set of communities, so that the level of trust between two nodes of the same community is the highest in the network. Different levels of inter-community trust are then defined, so that the level of trust between two nodes of two different communities is the same as the level of trust between their communities. A variable set of MAutoNet nodes represents a board of security managers. They are together

responsible of the self-management of the network in the security context. Besides, they represent the authentication and authorization board of servers of the network. We call them authority nodes. Each community has one authority node, but one authority node can be assigned to many communities. This is because the authority role implies certain capabilities, and a given community may not include nodes having these capabilities. The set of authority nodes is supposed to vary in an autonomic manner, in terms of the involved nodes and their number. The first action to take to set up a MAutoNet is to designate a qualified node as the authority node of the first community. Afterwards, we can use this authority node to insert other nodes in the first community. We can similarly create other communities and integrate them in the network using the existing authority nodes. The network evolves in terms of population when communities are integrated or revoked and when each community evolves in terms of node membership. Obviously, this evolution of population might be decided and managed by network administrators, and sometimes by network end-users, but it rather takes place in an autonomic context as we will see later in this paper.

A node is within the security perimeter of a MAutoNet when it belongs to one of its communities, where it might also belong to the set of authority nodes of the network. Each MAutoNet node has at least one secure relation with another MAutoNet node. Otherwise, we will have a single node, which is the first authority node, and this does not represent a network. In other words, for a non-authority node, there is at least one secure relation with the authority node which was used to insert the former in a community. For an authority node, there is at least one secure relation with another authority node, created after the integration of a community, and in case of a MAutoNet composed of one community, there is at least a secure relation with a non-authority node in this single community, created after the insertion of the non-authority node. We will see later in this section other types of MAutoNet secure relations. Figure 2 illustrates what we call a MAutoNet virtual security structure, which is characterized by a security perimeter and the encapsulated communities, nodes and secure relations. Geographically speaking, because a MAutoNet is a mobile wireless ad hoc network, this security perimeter is delimited by the union of the coverage areas of all the MAutoNet nodes, considering a multi-hop routing context.

We categorize the MAutoNet nodes according to their capabilities in terms of computing performance, storage capacity and electrical power availability. This should help, as we will see later, to identify the possible types of secure relations that a node can assume. This categorization is configurable, but a default one is used, according to which, a node can be a heavy-duty device or a light-duty device. A heavy-duty device is capable of performing asymmetric cryptography and storing the associated cryptographic materials, and it is relatively always on. A light-duty device is capable of performing only symmetric cryptography and storing only the corresponding security information, and it might be off or in a sleep mode at any time. As for an authority node, it is a heavy-duty device which has server capabilities, such as multithreading support and the storage capacity required to assume the designated server role. Moreover, even that we work on self-healing mechanisms to efficiently replace a leaving, a lost or a damaged authority node, it is always advised to assign the authority role to a node of a limited mobility and a long-life membership. Other categories can be defined later, like for example a category of average-duty devices that cannot run a complete asymmetric encryption scheme, but can at least verify signatures in a light version of the RSA public-key system.

A mutual authentication should take place before a secure relation is established between two MAutoNet nodes. Authentication between a new node and an authority node takes place implicitly during the node insertion operation. Node insertion takes place through a single-hop communication between the new node and the authority node of the selected community, and using a protected short-distance channel. The result is an ADR (Authority-Device secure Relation). According to the category of the new node, it is assigned either a public-key certificate, or a secret key shared uniquely with the authority node of its community. In the first case, the new node generates its key pair, and the authority node sends its public key to the new node, and assumes the role of the certification authority. In the second case, the authority node generates the ADR secret key and sends it to the new node. Similarly, authentication between two authority nodes takes place implicitly during a community integration operation. Community integration takes place through a single-hop communication between the authority node of the new community and an existing authority node, and using a protected short-distance channel. The result is an AAR (Authority-Authority secure Relation). The involved authority nodes exchange public keys and public-key certificates, assuming each the role of certification authority for the other. As for authentication between two non-authority nodes belonging to the same community, either they have certificates assigned by the authority node of the community and they use them in a certificate-based mutual authentication protocol, or one at least could not be assigned a certificate and the authority node intervenes as an authentication server. Finally, for authentication between two non-authority nodes belonging to two different communities, either both nodes were assigned certificates by the relevant authority nodes and a mutual authentication protocol based on chaining of certificates is used, or one at least could not be assigned a certificate and relevant authority nodes can intervene and assume together the role of an authentication server. We are currently working on different authentication protocols in this context, and specify them in conformity with the autonomic behavior. In other words, their specification methods will allow certain autonomic modules to adapt them to unpredicted changes in the security environment. For example, they can be optimized by a self-healing module after a successful attack revealing a vulnerability in the authentication system.

A MAutoNet secure relation is a contract between two MAutoNet nodes. It establishes an agreement on securing data exchange in terms of confidentiality and integrity. It starts when a mutual authentication takes place between the two nodes as belonging to the same MAutoNet. It ends when one of the two nodes does not belong to the MAutoNet any more. Once created, the secure relation is based on a mutual trust between the two nodes of a certain level as explained above. According to the level of trust, there could be authorization rules controlling the communications in a secure relation. Such access control rules are based on the roles of the both nodes and on the level of trust between

them. Moreover, the choice of the cryptographic materials used to secure the data exchange depends on the categories of the both nodes. Hence, secure relations are classified depending on node roles and categories and on the trust levels. For example, figure 2 illustrates a virtual security structure that we already defined for a home MAutoNet [10]. In this MAutoNet, default node categorization is used and only two trust levels are defined: either a high trust between nodes of the same community or a low trust between nodes of different communities. This is why a first classification of secure relations defines two relation types: LTR for a Low-Trust Relation and HTR for a High-Trust Relation. A second classification based on the roles of the MAutoNet nodes defines three relation types: ADR for an Authority-Device Relation, AAR for an Authority-Authority Relation and DDR for a Device-Device Relation. A DDR secure relation class is an abstract one. It is not used directly. A third classification based on the categories of the home devices (MAutoNet nodes) derives three more relation types from DDR: HHR for a relation between two heavy-duty devices, HLR for a relation between two non-authority nodes of different categories and LLR for a relation between two light-duty devices. As for relations classified as ADR or AAR, the cryptographic materials shared with or between authorities will be used by default, so there is no need for a further classification based on device categories in these two cases.

The autonomic security system is automatically set up when the initial virtual security structure of the MAutoNet is created. In other words, when the following initial steps are done: 1)The autonomic security architecture is installed in the different nodes before inserting them in the MAutoNet. 2)The initial trust levels, node categories, cryptographic material types and security policies are specified, or the default ones are utilized. 3)The first communities are created, populated and integrated. 4)The first secure relations are established, in addition to those resulting from node insertion and community integration operations. We suppose that all those initial steps are driven by end-users through a simple set up operation based on well-configured initial software and hardware components. Expert administrators may also participate in the accomplishment of these steps, such as for specifying security policies other than the default ones. Nevertheless, once the initial security perimeter is materialized, the autonomic security system assumes the self-management of the virtual security structure, and the employed security components. It should be able to detect the network evolution events related to security, and respond to them by adapting the virtual structure or its components in terms of reconfiguration, protection, optimization or repair. As a result, we will have an autonomic system that manages the MAutoNet evolution of population and topology in a security context. It manages its evolution as well, which makes it also responsible of a set of security evolution events. We mention hereinafter the different evolution events to be handled by the autonomic security system.

We identify six events of population evolution: the insertion, removal, banishment and reinsertion of a node, and the integration and revocation of a community. Node banishment might be needed when a node is out of the security perimeter for a period greater than an allowed maximum, and node reinsertion is applied to cancel the banishment of a node.
We identify two events of topology evolution: the merging and splitting of a community.
We identify four events that affect population and topology together: the exportation and reintegration of a subnetwork, and the merging and splitting of entire MAutoNets. Exportation of a subnetwork might be needed when a set of MAutoNet nodes are supposed to leave the security perimeter for a predefined period which might be greater than the allowed maximum period of absence, and reintegration of a subnetwork is needed when such a set of nodes is back, given the fact that a subnetwork of MAutoNet may evolve as freely as an independent MAutoNet during its absence. See [11] for more details about subnetwork exportation.
All the previous events might represent a security evolution as well, but we identify six more events that explicitly have such effect: the acquisition, dispossession, delegation and retirement of the authority role, and the establishment and termination of a secure relation. Dispossession of an authority role is needed when an authority node is normally removed or banished. Authority delegation is needed in the exportation of a subnetwork, and authority retirement is needed to end an authority delegation during the reintegration of a returning subnetwork. See [11] for more details about authority delegation.
Note that a human action, either normal or malicious, might be the trigger of any of the previous eighteen events, which may imply an autonomic reaction. However, the ad hoc mobility of MAutoNet nodes, and the ad hoc nature of the MAutoNet in general, are mainly the triggers of those events in an autonomic context. This is why the Autonomic Security System must monitor the valid functionality of the network, the mobility of its nodes, the actions of its users and administrators, and the signs of attacks in its environment. We are currently working on evolution event detection and handling. In fact, it is up to the authority nodes to perform the cycle of monitoring, detection, event analysis, decision and finally autonomic reaction to a detected evolution event. For certain evolution cases, authority nodes collaborate to achieve the required autonomic reaction. For example, they might negotiate optimizing one of the security policies in response to merging two communities as we will see in a later section in this paper.

3. POLICY SUBSYSTEM

A policy subsystem is a main component of an autonomic system. It translates high-level objectives, usually through several intermediary rule specifications, into low-level elementary rules, upon which autonomic decision are based. We introduce in this section our design of MAutoNet security policy subsystem.

An autonomic system necessarily has initial policies, which are supposed to be provided by default. Default policies would be already enforced at low level, but also they will have copy representations in form of rule specifications at the administrator level and high-level objectives at the end-user level. This is to provide a current view of the autonomic system functionality to the both types of users. Usually, default policies are modified later, in response to the evolution or self-maintenance of the encapsulating autonomic system or its environment. For instance, we identify in our design of the MAutoNet security policy subsystem, which is elaborated in figure 3, three sources of modification for a security policy in an autonomic security system: end-users, administrators and authority nodes. End-users may want to make

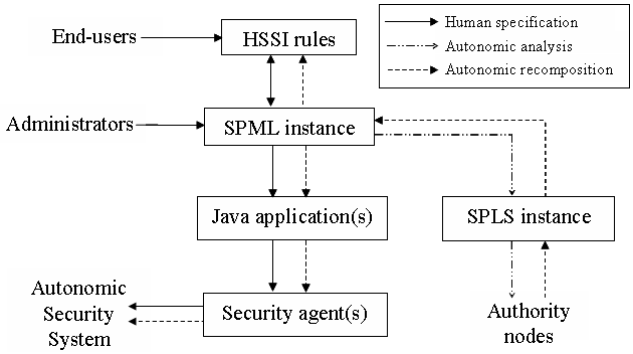

Figure 3: Security Policy Subsystem

changes to their high-level security objectives, which may affect security policies. Administrators may want to explicitly optimize or repair some part of the autonomic security system by means of security policy management. Authority nodes may need to analyze, negotiate and recompose a security policy, or a certain part of it, in the context of an autonomic operation. The last functionality is the one we expect to take place so often. In an autonomic system, human intervention is expected to be as reduced as possible. It is like the autonomic systems of the human body, where the owner of the body (end-user) is not supposed to intervene in their functionality, and doctors and human body specialists (administrators) may be called only when those systems are not able to manage themselves in some cases.

Figure 3 illustrates the design and functionality of the MAutoNet security policy subsystem. Policy manipulation after initial enforcement might go through different tracks before modifications are enforced in low-level forms. The human specification track may be launched by an end-user or an administrator as already explained. This kind of manipulation is transparent for the end-user. In other words, he is not necessarily aware of the fact that security policies are being changed. All he knows is that he is acting on his security objectives. He does so by means of a high-level language, through a user-friendly interface, which is in relation with the configuration of the security environment as a whole. We call this language HSSI for Human / Security System Interface. As for administrators, they are completely aware of what they are changing. This is why they use a specific language for manipulating policies. For now, we work on this language in the security context and we call it SPML for Security Policy Management Language. In a human specification track, policies are modified through HSSI or SPML, and when the user confirms his modifications, a built-in interpreter transforms the corresponding SPML instance into one or more Java applications, which in their turn will be compiled into security agents (see figure 1), and distributed on the nodes where the relevant policy should be enforced. Obviously, an HSSI/SPML translator is needed when the human specification is driven by an end-user. Nevertheless, the system state should be always represented through high-level objectives, so an SPML/HSSI translator is also needed in a human specification driven by an administrator. The autonomic analysis track is launched automatically by an authority node which needs to negotiate policies with other authorities to modify them in the context of an autonomic operation. An authority node can access SPML instances, search them for specific information needed in the negotiation process, and reformat the extracted data in a logic language specific to autonomic manipulation. The use of such language is necessary for representing the logic of the policy rules, which is not possible by SPML, because the latter is an XML language used to specify syntax and actions. On the other side, SPML can not be replaced by the logic language used by authorities, because SPML is easier to interpret into Java code, and because the logic language is not supposed to represent every aspect of a policy. For now, we work on a logic language in the security context and we call it SPLS for Security Policy Logic-based Specification. After reformatting the needed information in SPLS, an authority node uses them in a negotiation process aiming at the modification of the relevant security policy as an autonomic reaction to some detected event. At the end of a negotiation process, an autonomic recomposition track is launched, aiming at gathering the SPLS-formatted modified policy parts, reformatting them in SPML and reintegrating them in the original SPML instance of the policy. From one side, the built-in SPML interpreter is called to enforce the new version of the policy, and from the other side, the SPML/HSSI translator is called to update the state of the security system configuration at the end-user high level. Autonomically modified policies are eventually enforced on the involved authority nodes, and distributed to the nodes needing them. We are working on policy distribution considering node availability in a multi-hop routing context.

According to [12], policies define choices in behavior in terms of the conditions under which predefined operations or actions can be invoked rather than changing the functionality of the actual operations themselves. Security policies are required in the autonomic security system of a MAutoNet to reflect the different functionalities encapsulated in the virtual security structure and how it should evolve, which allows for monitoring, controlling and managing the MAutoNet security. In terms of functionality, we define for each node of the MAutoNet an *Authentication* policy to specify rules of mutual authentication between two nodes before being bound by a secure relation, an *Authorization* policy to specify rules for access control between two nodes bound by a secure relation, and a *Communication* policy to specify security materials employed during a communication session between two nodes in the context of a secure relation. In terms of evolution management, we define for authority nodes exclusively a *Certification* policy to specify rules for assuming the authentication server role during the establishment of secure relations, a *Collaboration* policy to specify when and how to inter-operate with other authority nodes, a separate *Evolution* policy for each evolution type to specify when and how to respond to the relevant detected event and how to achieve and manage the required evolution, and a *Self-healing* policy to specify the actions to be taken when detecting security violations or signs of a successful attack on the MAutoNet. It is useful to specify policies relating to groups and nested groups of entities, and to group the policies pertaining to the rights and duties of a role or position [12]. Therefore, we distinguish, in terms of policy specification, between policies having the same specification in all the network, such as those used for self-management purposes (the self-healing policy for instance), and policies that may vary from a community to another, such as those used to control secure relations (the authorization policy

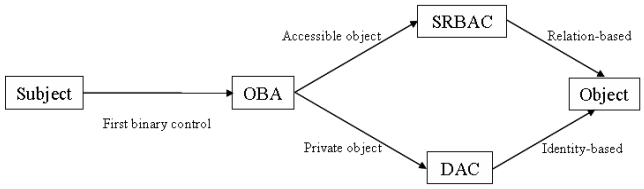

Figure 4: Access Control Model

for instance). Besides, we distinguish between policies used by all nodes and those used by authority nodes exclusively, as could be noted in the policy definitions above. Policies employed exclusively by authority nodes can also be called obligation policies [12], because they specify event-triggered rules, which are conditioned by certain actions and used to define adaptable management operations.

Modification of security policies might be needed for synchronization, enhancement or optimization purposes after a certain network evolution, or in the context of self-healing operations. A need for negotiation in this case might arise, either from a possible conflict with the properties of the security environment of certain communities in case of changing network-level policies, or from possible conflicts with pre-existing policies in general. Nevertheless, there are cases, other than conflicts, that imply negotiating security policies, of which we can mention the following network evolution types: 1) Community integration (a new community might have different network-level policies). 2) Delegation termination (the returning subnetwork might have changed its policies). 3) Merging communities (merged communities might have different community-level policies). 4) Importing parts from other MAutoNets (potential differences in any of the security policies). 5) Merging two MAutoNets (it is necessary to negotiate the best policy specification for the resulting network). To accomplish a security policy negotiation, each authority node will compute the SPLS instance of the needed information from the specified policy and inputs it to the security agent which is responsible of security policy negotiation in the autonomic security layer (figure 1). SPLS-formatted information will be then exchanged between authority nodes in the context of a Security Policy Negotiation Protocol (SPNP). After having negotiated the specified policy data, and decided about its new specification, the autonomic recomposition track is followed. We are currently working on the different issues of the security policy negotiation. We study the potential assets and components of this operation, define different specification languages, develop the corresponding translators and interpretors, design the security policy negotiation algorithm and specify and validate the SPNP protocol.

4. ACCESS CONTROL

A MAutoNet node may operate out of the security perimeter in the context of what we call external applications. An external application does not use the autonomic security system, and it might involve components from outside the network. It might be secure or not according to its context, but it is anyway non-secure for the MAutoNet. Therefore, it should be isolated by the MAutoNet security perimeter. In other words, although a MAutoNet node is allowed to host objects that can be accessed in the context of an external application, the objects that are handled within the security perimeter, which we call classified objects, must not be available in such context. Nevertheless, a node may classify an unclassified object to restrict its availability to internal applications only, which are the applications involving MAutoNet secure relations. On the contrary, a classified object can never be changed to be unclassified. This is to guarantee the confidentiality and the integrity of the data handled within the security perimeter. The MAutoNet access control model (figure 4) concerns internal applications exclusively. In other words, it represents authorization rules for a MAutoNet node willing to access classified objects hosted by another MAutoNet node, within a session of a secure relation already established between the both nodes. In such a communication session, remote objects are accessed on their hosts or through a data exchange. In both cases, the access aims at performing a certain action on the remote object. According to our access control model, a right is a permission given to a node to perform an action on an object hosted by another node, in the context of a secure relation binding the both nodes, provided that the required access is already authorized. So, we need first to make a binary check to see if the access is authorized or not, and if authorized, we can then verify the access rights.

An access may be authorized or not, after a self-organizing control operation, which is based on what we call OBA model (Object-Based Authorization). Objects hosted by a MAutoNet node are classified in OBA model as:

- *Private* (inaccessible objects): access is unauthorized by default.
- Accessible objects:
  - *Protected*: either the hosting node has no authority role and access is authorized only for the authority node or a delegated authority node of the community, or the hosting node has an authority role and access is authorized for all the nodes it manages in its community.
  - *Friendly*: access is authorized only for the nodes belonging to the same community.
  - *Administrative*: the hosting node has an authority role, and access is authorized only for other authorities and delegated authorities.
  - *Public*: access is authorized for all the MAutoNet nodes.

We inspired the OBA model from the scope management in Java object oriented programming. We use an analogy, in which we consider the whole MAutoNet as a Java application, a community as a package and a node as an instance of one of three classes representing the three possible roles for a node: authority, delegated authority and device, and they are respectively related by inheritance. Moreover, in order to make this analogy useful in implementation terms, we see node categories as interfaces and node relations as attributes which are instances of classes representing the different relation types. A correspondence is then possible between the autonomic control of network evolution and the event-driven self-management methods in a relevant Java application. Actually, we assume that *Protected* and *Administrative* objects are created and classified during the autonomic manipulation of the network evolution and their classifications

must not be altered, while the other objects are by default *Public* when created, and their classification can be reconfigured later if necessary. The three object classes *Public*, *Friendly* and *Private* are considered respectively as ordered security levels. The *Public* object class represents the lowest security level. A node can change the ordered classes of the objects it hosts upwards only. For example, a node can reconfigure a *Public* file as a *Friendly* one in order to restrict its access to the nodes of its community only. This is necessary to protect the confidentiality and integrity of accessed objects with regards to data flow between communities within the security perimeter.

As illustrated in figure 4, once the access is authorized according to the OBA model, if the access concerns a private object, the hosting node can apply its own discretionary access control based on the identity of the accessing node. Otherwise, if the access concerns an accessible object, rights are determined depending on the roles of the accessing node and the node hosting the object, and on the trust level of the relevant secure relation. For example, in an authority delegation session, an authority node has the permission to change the role of a node of its community (protected data of the node) to make of it a delegated authority, and then to copy administrative data on it. However, a node has the permission to verify the role of the authority node of its community (protected data of the authority node), but not to change it. We are developing SRBAC (Secure Relation Based Access Control) out of RBAC for modeling this phase of MAutoNet access control. Actually, SRBAC is an RBAC applied in the context of a secure relation, and depends on its characteristics. Because the role of a same node might differ from a secure relation to another, we can make a correspondence with the varying role of the subject from a session to another in the flat RBAC model. We also might be able to use the hierarchy support in the hierarchical RBAC model to represent the relation between permissions assigned to a delegated authority and permissions assigned to an authority. Nevertheless, according to our model, the trust level of the secure relation should be taken into account, and the role and the class of the object should be considered as well. So, we have more differences between access sessions to care about than in RBAC model. Maybe, the use of constraints in the constrained RBAC model would give a solution by introducing conditions related to the characteristics of the secure relation encapsulating the access session. So either we will be able to utilize the symmetrical RBAC model (hierarchical and constrained at the same time), or we will have to introduce an enhanced model of RBAC. We are still working on SRBAC model formalization, but we can currently expect, to a certain extent, how an SRBAC policy specification should look like, as elaborated hereafter.

We will consider in the following the example of the home MAutoNet [10] (see figure 2) to discuss the implementation and negotiation of an SRBAC policy. The configuration of the autonomic security system depends mainly upon the following parameters: node categories (heavy-duty or light-duty) - trust levels (high or low) - node roles (authority, delegated authority or device) - duty-based relation classification (HHR, LLR and HLR) - trust-based relation classification (HTR and LTR) - role-based relation classification (AAR, DDR and ADR) - object classes (private, accessible or unclassified). In terms of access control, we are not concerned about node categories or duty-based relation classification. On the other hand, SRBAC deals with objects of the abstract class "Accessible" only. This is why only certain parameters are taken into consideration in the following explanation.

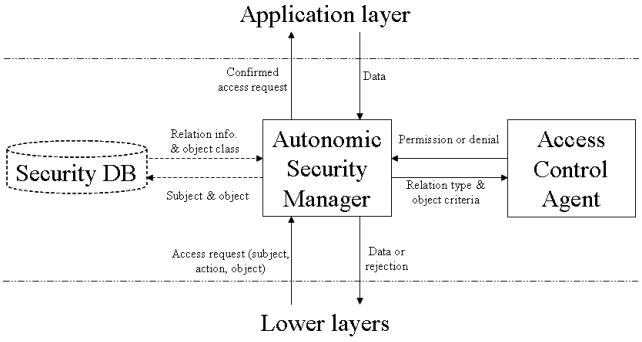

Figure 5: Access Control Implementation

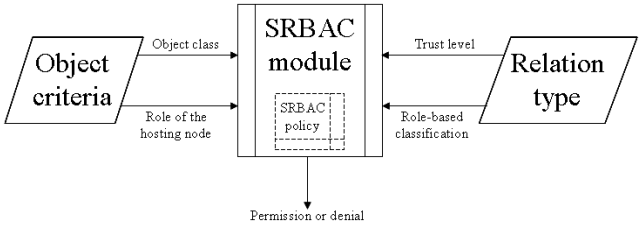

Figure 6: SRBAC Policy Enforcement

In the context of a given secure relation, the autonomic security manager (see figure 5) receives a request for accessing an object. The request would indicate the identifiers of the subject (requesting node) and the object, in addition to the desired action. The autonomic security manager uses its security database to look up the class of the object, the role of the hosting node in the context of the relevant secure relation, and the trust-based and the role-based classifications of the secure relation. It then calls the security service agent that is responsible of the access control, passing the retrieved information to it as parameters. The agent is composed of three main modules representing the two phases of the MAutoNet access control model. The OBA module is launched first, and then it launches the DAC module or the SRBAC module according to the object access class. The second-phase module (DAC or SRBAC) uses its access control policy to verify if the asking subject has the right to perform the desired action on the specified object. Figure 6 illustrates how the SRBAC module receives the necessary parameters for consulting the SRBAC policy, which is implemented as one of its components, before giving a decision of permission or denial. The relevant thread of the autonomic manager waits meanwhile for the answer. Once arrived, either it sends a rejection to the subject in case of access denial, or it forwards the access request with a confirmation to the relevant application in the higher layer.

A part of the SRBAC policy, which controls the access in the context of an ADR relation, may have the following default specification, represented in SPML language:

```
<SRBAC>
  <relation category=ADR level=HTR>
```

```
    <object role=device>
      <access class=any>
        <permission action=all />
      </class>
    </object>
    <object role=authority>
      <access class=public>
        <permission action=all />
        <denial action=delete />
      </class>
      <access class=any>
        <permission action=read />
      </class>
    </object>
  </relation>
</SRBAC>
```

An end-user is generally not able to interpret it, and he needs a corresponding HSSI representation in order to understand it. We are still working on the definition of HSSI and the development of the corresponding tools, which we will keep out of the scope of this paper. As for a MAutoNet administrator, he should be able to interpret the above specification and understand the following: in a secure relation between an authority and a device (ADR), based on a mutual trust of the high level (HTR), if the accessed object is hosted by the device, the accessing authority node is allowed all actions whatever the class of the accessed object is, while if the accessed object is hosted by the authority node, the accessing device is allowed all actions except delete if the accessed object is of class public, and it is allowed to read the accessed object whatever its class is.

We will not talk here about an operation performed by an administrator. We will try to clarify how and when authority nodes might negotiate an SRBAC policy. As explained earlier, SPLS instances are used in security policy negotiation. In fact, we still study the best use of the existing logic-based languages, with necessary adaptation, to define the SPLS language. Anyway, we can use one of the studied languages to continue our example. It is the ASL language (Authorization Specification Language), which is a stratified first-order logic language [15]. Given a predicate called *cando* used for representing an access rule, taking respectively as parameters the object class, the accessing subject and an action preceded by a sign indicating permission (+) or denial (-), and given that relation types and trust levels are handled as sets to which a relation may belong or not, the above SRBAC policy could be represented by the following SPLS rules:

```
cando(any, authority, +all)
<- in(r, ADR) & in(r, HTR)

cando(public, device, +all)
<- in(r, ADR) & in(r, HTR)

cando(public, device, -delete)
<- in(r, ADR) & in(r, HTR)

cando(any, device, +read)
<- in(r, ADR) & in(r, HTR)
```

Let us suppose now that SRBAC policies differ between communities, and that the above policy is enforced in the community C1. Suppose that there is another community C2, in which the authority has more restricted access rights, so that the corresponding parts of the SPML specification looks as follows:

```
..
<object role=device>
  <access class=any>
    <permission action=read />
  </class>
</object>
..
```

Suppose now that for a certain reason C1 and C2 should be merged. By some technique, which is still under research, one of the two involved authority nodes detects the conflict between the SRBAC policies of C1 and C2. It reacts by contacting the other authority for launching a security policy negotiation process to resolve the detected conflict. We will not go into the very details of the negotiation protocol. We just want to emphasize here the steps that are interesting for us in the scope of this paper. Each authority will extract the needed information from the SPML instance of its community, reformat it in SPLS, and send it to the other authority. Each one will have then the both following SPLS rules representing the conflict:

```
cando(any, authority, +all)
<- in(r, ADR) & in(r, HTR)

cando(any, authority, +read)
<- in(r, ADR) & in(r, HTR)
```

On each authority node, the conflict will be resolved by first unifying the both rules:

```
cando(any, authority, (+all)&(+read))
<- in(r, ADR) & in(r, HTR)
```

And then by applying the two conflict resolution rules

```
(+x)&(+y) = +(x&y)
all&read = read
```

which should make part of the configuration of the security policy negotiation module, each authority will decide to use the resulting rule:

```
cando(any, authority, +read)
<- in(r, ADR) & in(r, HTR)
```

which is in this case identical to the rule used in the SRBAC policy of C2. The two authorities exchange and validate their decisions, and as a result the authority of C1 recompose and distribute the SRBAC policy of its community, according to the functionality of the security policy system (see figure 3), before executing the community merging.

5. CONCLUSION

We defined our model of mobile autonomic networks that we call MAutoNet. Through this definition, we explained our paradigm for autonomic networks, and we elaborated the relation between such environments and the autonomic computing systems. We also explained why mobile ad-hoc networks are the most likely to behave as autonomic networks, and accordingly cited MAutoNet application fields.

We introduced a framework for building autonomic security systems in MAutoNets, in which we presented an autonomic security architecture and a virtual security structure.

In terms of architecture, we proposed a design for an autonomic security layer to be integrated as a secure session layer right under the application layer in the communication stack. In terms of security models, we showed that the virtual security structure reflects a trust model based on node communities, an authentication model based on node heterogeneity, and a secure relation model based on both trust and node categorization. These security models eventually allowed us to propose a MAutoNet evolution model as a working context for the desired autonomic security system.

Because handling policies is a must in autonomic systems, we focused on the autonomic security policy subsystem and its specification languages and autonomic components. We showed how we made use of existing technologies in addition to a set of new specification languages. As a contribution to modern solutions dedicated to autonomic systems, we introduced one of our ongoing works, by which we intend to propose a Security Policy Negotiation Protocol (SPNP).

We opted for working on a MAutoNet authorization model as a basis for studying the autonomic handling of security policies. After defining our authorization system and explaining a part of its implementation, we explored in this paper an example of an autonomic operation which implies a negotiation of an access control policy after merging two MAutoNet communities. Through this example, we introduced a new access control model, we employed some instructions of the initial definitions of our policy specification languages, and we proposed some preliminary ideas about how the security policy negotiation algorithm could work.

We thought about an autonomic security system for the first time during a work on home network security [10], due to a need for self-management based on high-level policies. We wanted then to expand our work to similar networks in this perspective, so we started to study security in autonomic networks in general [9]. We directed then our main efforts to a couple of fields, namely security policy autonomic processing and collaboration between nodes in MAutoNets [11]. In this paper, we introduced the different concepts of our research in general and presented the latest work in the first field using the case of authorization policies.

We are currently working, according to specific applications and scenarios, on a representative set of high-level security policies specified in HSSI. Those policies should give a wide view of autonomic network needs, which would help reaching a formal definition of SRBAC with respect to existing access control models. SPLS language can be then fully elaborated on a good basis, in addition to a preliminary definition of SPML. At that point, we will implement a MAutoNet prototype and use it to test the autonomic security system on the most challenging events of the evolution model, such as merging two communities or two networks.

6. REFERENCES

[1] A.Datta and K.Aberer. The challenges of merging two similar structured overlays: A tale of two networks. In *the First International Workshop on Self-Organizing Systems (IWSOS 2006)*, 2006.

[2] D.Balfanz, D.Smetters, P.Stewart, and H.Wong. Talking to strangers: Authentication in ad hoc wireless networks. In *Symposium on Network and Distributed Systems Security (NDSS '02)*, 2002.

[3] F.Stajano and R.J.Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *the 7th International Workshop on Security Protocols*, 1999.

[4] H.Luo, P.Zerfos, J.Kong, S.Lu, and L.Zhang. Self-securing ad hoc wireless networks. In *the 7th IEEE Symposium on Computers and Communications*, 2002.

[5] J.O.Kephart. Research challenges of autonomic computing. In *the 27th International Conference on Software Engineering*, 2005.

[6] J.O.Kephart and D.M.Chess. The vision of autonomic computing. *Computer*, 2003.

[7] L.M.Feeney, B.Ahlgren, and A.Westerlund. Spontaneous networking: an application-oriented approach to ad hoc networking. *Communications Magazine, IEEE*, 39(6):176–181, 2001.

[8] L.Zhou and Z.J.Haas. Securing ad hoc networks. *IEEE Network*, 1999.

[9] M.Aljnidi. Sécurité des réseaux mobiles autonomes. In *Premier workshop GET sur les réseaux spontanés*, pages 17–18, Rennes, France, November 2006.

[10] M.Aljnidi and J.Leneutre. Autonomic security for home networks. In *the First International Workshop on Self-Organizing Systems (IWSOS 2006)*, pages 239–242, Passau, Germany, September 2006.

[11] M.Aljnidi and J.Leneutre. Towards an autonomic security system for mobile ad hoc networks. In *The Third International Symposium on Information Assurance and Security (IAS 2007)*, Manchester, United Kingdom, August 2007.

[12] N.Damianou, A.Bandara, M.Sloman, and E.Lupu. A survey of policy specification approaches, 2002.

[13] P.Horn. Autonomic computing: Ibm's perspective on the state of information technology. Technical report, IBM Research, 2001.

[14] S.Dobson, S.Denazis, A.Fernandez, D.Gaiti, E.Gelenbe, F.Massacci, P.Nixon, F.Saffre, N.Schmidt, and F.Zambonelli. A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems*, 2006.

[15] S.Jajodia, P.Samarati, and V.S.Subrahmanian. A logical language for expressing authorisations. In *IEEE Symposium on Security and Privacy*, Oakland, USA, 1997.

[16] S.L.Keoh and E.Lupu. Towards flexible credintial verification in mobile ad-hoc networks. In *the 2nd ACM Annual Workshop on Principles of Mobile Computing (POMC'02)*, 2002.

[17] S.Schmid, M.Sifalakis, and D.Hutchison. Towards autonomic networks. In *the First International IFIP TC6 Conference on Autonomic Networking(AN 2006)*, 2006.

[18] S.S.Yau, Y.Yao, Z.Chen, and L.Zhu. An adaptable security framework for service-based systems. In *the 10th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS'05)*, 2005.

[19] T.Messerges, J.Curkier, T.Kevenaar, L.Puhl, R.Struik, and E.Callaway. A security design for a general purpose, self-organizing, multi-hop ad-hoc wireless network. In *the First ACM Workshop on Security of Ad-hoc and Sensor Networks*, Fairfax, Virginia, 2003.