Self-adaptive Security at Application Level: a Proposal

Alberto Ferrante *, Antonio Vincenzo Taddeo *, Mariagiovanna Sami † *, Fabrizio Mantovani *, Jurijs Fridkins *
* ALaRI, Faculty of Informatics, University of Lugano
Lugano, Switzerland
Email: {ferrante, taddeo, mantovaf, fridkinj}@alari.ch
† Dipartimento di Elettronica e Informazione,
Politecnico di Milano,
Milano, Italy
Email: sami@elet.polimi.it

***Abstract*—Self-adaptive systems have the ability to adapt themselves to mutating external or internal conditions without requesting any intervention of the user; the security of such systems is influenced by those adaptations. Therefore, also the security mechanisms that are put in place by the operating system, should adapt to maintain the desired security level.**

This paper proposes a self-adaptive framework for the system security. This adaptation scheme allows the system to choose the best set of security policies at every given time; this set is determined by considering the system internal and external conditions as well as the application requirements. The proposed framework deals with self-adaptation at system level in order to provide both a domain-independent and a flexible solution.

I. Introduction

Self-adaptive computational systems have the ability to adapt themselves to mutating internal and external conditions without requesting any intervention of the user. Self-adaptation can happen at different levels and can be triggered by a change in the environment (e.g., office vs. operational field), a change in the tasks that are assigned to the system, or a change in the system's operational conditions (e.g., a battery operated system detects a change in the battery status). In fact, these changes in the operational conditions may require hardware and/or software adaptations to keep fulfilling the requirements of the applications; the ability to self-adapt software can provide a great degree of flexibility that can be even further increased by equipping the devices with reconfigurable hardware. This allows the system to map specific functions on the hardware to optimize the system performance. Creating a system as a network of self-adaptive elements also introduces the ability of dynamically sharing the computational resources of the different elements so as to achieve the best cost-performance ratios, adapting to workload and/or environment variations.

Sample applications of this technology have been shown in [1]. For example, Section VI of that document [1, pp. 29-33], describes a self-adaptive portable-device based system for helping officers (firefighters, policemen, rescue teams, ...) in their work. The portable devices can be network connected by using different communication means. A set of sensors, that can be used for sensing both environmental and personal officer conditions, is connected to each device. For example, health monitoring (both of the officer or of other people) and face recognition can be performed by means of the system sensors. Self-adaptivity is very useful for this kind of devices: it can be used for allowing the system to adapt to mutating environmental conditions (e.g., in or out of office) or to mutating set of tasks to be executed (e.g., facial recognition or health analysis). Portable devices usually have limited computational resources as well as limited battery life. Therefore, resource sharing among devices can be used to overcome these limitations: by distributing tasks to other devices, as best requested by operating conditions and requests, each device can in turn access more computational resources and can perform more computations while achieving a balanced use of battery power.

Security might also be affected by changes in the internal or external conditions of the system. In fact, these changes may trigger hardware or software reconfigurations which, in turn, may change the level of security of the system. A simple example can be in the transition from a protected wireless network to an unprotected one: the user may not even notice it as long as a connection stays alive. While this transition may be transparent to the user, it is certainly not for the system security. Such a change may require a change in the security settings to keep the system safe or, anyway, to keep the level of security constant. Thus, for self-adaptive systems, security need to be reconsidered in a new way: a target security level should be defined and the system should change the security settings accordingly with the system settings to match the target security level. At the same time, mutating system settings may require to adopt more conservative security settings. For example, if the system is low on battery, it may try to change the security algorithms used to save power. This can be done only if the newly chosen algorithms are compatible with the target security level. All of these security self-adaptations can be done by changing the security policies adopted by the system. Thus, a mechanism for security policy self-adaptivity need to be studied.

In this work we present the main concepts related to security self-adaptivity. Security self-adaptation should be included in a proper part of the operating system. Policies self-adaptation will be performed by re-deploying the policies in a standard way. Information about the hardware of the device and on the environment, can be easily accessed at operating system level through proper device drivers.

The applications presented in [1], in particular the one mentioned above, have been used as a reference case.

Section II explains the related work that has been presented in the literature; Section III explains self-adaptive security concepts and outlines the possible solutions for its management. Section IV presents a possible scenario in which to apply the self-adaptive security concepts.

II. Related Work

Aim of this section is to give an overview of previous researches in the field of adaptive security system. The most difficult challenge is to find a match point between the usual *static* approach adopted in the design of security mechanisms and the *dynamic* behavior of adaptive systems. In the literature there are not many publications about this topic.

We focus here on papers covering both aspects of adaptation and security. They are presented in a top-down view according to the security level they refer to: theoretical aspects ([2], [3]), application and service level ([4], [5]), protocol level ([6]), algorithm level ([7]), and primitives level ([8], [9]).

A first theoretical study is presented in [2]; this paper proposes to design an Adaptive Security Infrastructure (ASI) composed of three conceptual components: sensor, analysis, and response. Such a design includes several problems: activity coordination and synchronization between its components; global and local nature of all components; adaptive security policy specification and enforcement; the specification, derivation and verification of the response triggered by a complex detection and analysis system.

In [3] the theoretical aspects concerning adaptive security imply the use of Control Theory and Dynamic Systems Theory. In general, adaptation is considered as the solution of an optimal control problem. The author underlines also the lack of reasonable implementations due to difficulties arising from the high complexity of the security components, their exact identification, and the restrictions on the response time of the adaptation algorithm.

In [4] the security problems of collaborative distributed systems are addressed by dividing them in three logical domains: client domain, tasks repository, and low-level control device domain. The authors propose a solution at application/service level by introducing a security framework to control the execution of client's tasks by the use of a Security Control Gateway.

Another proposal at application/service level is presented in [5] which describes an adaptive security schema for denial of service (DoS) threats. The framework delegates to a fuzzy feedback controller the task of selecting the suitable security level of a node. The fuzzy feedback controller receives a set of vulnerability metrics and evaluates the vulnerability of a node.

Unlike the previous works, in [6] the authors describe a self-adaptive security framework at protocol level. The mechanism selects the optimal set of security protocols with the best security/performance ratio. A Security Index (SI) and a Performance Index (PI) are computed for each security protocol. These indexes are used later to dynamically change the security protocol set depending on the malicious level of a node's neighbors.

An algorithm adaptation mechanism is explained in [7]; in this paper an Adaptive Cryptographic Engine (ACE) has been implemented on a FPGA board. The ACE changes "on the fly" the cryptographic algorithm used to dynamically adapt to different security parameters of the IPSec protocol. The controller module of the ACE satisfies the incoming configuration request by loading the suitable cryptographic algorithm bit-streams from a crypto-library.

Adaptation of cryptographic primitives is instead presented in [8], where the basic idea is to change the AES cryptography key length according to the confidentiality level exposed by the user.

The research presented in [9] deals with the implementation of some security primitives on reconfigurable hardware; reconfiguration decisions are based on external events that alert on incoming attacks. Several monitors are used to detect attacks and some controllers are used to drive the reconfigurations.

The above papers address the adaptation of system security by proposing a solution focused on a single security aspect (e.g., DoS attacks). We propose instead to adopt a system-level approach which is general and centered on the concept of security policy. Once a security policy is defined, it may be used within our self-adaptive schema described in the following section.

III. Self Adaptive Security

In this section we discuss the security self-adaptation mechanism. Self-adaptation requires the ability to enforce different sets of security policies at different instants of time. A security policy includes all the security settings of the system: access control as well as communication security settings. Moreover, each policy have an associated cost that depends on the resources required to enforce it. For example, the cost of a policy can be computed as follows:

$$P_{x,cost} = P_{x,fix\_cost} + P_{x,usage\_cost} \tag{1}$$

where P_{x,fix_cost} represents the fixed costs of the policy x; $P_{x,usage\_cost}$ represents the variable part of the x policy cost. The former describes the costs that must be sustained for enforcing a given policy; the latter takes into account the costs associated with using an already enforced policy. For example, the fixed cost may be associated with possible additional (reconfigurable) hardware that is required for enforcing the policy; the variable cost may be associated with the power spent for enforcing the policy for different applications.

Each application, when started, states its security requirements. Each requirement can be either *hard* or *soft*. Hard requirements are to be considered mandatory conditions for running the considered application. Soft requirements are associated with desired, but non mandatory, conditions for the application. The main application requirements can be expressed as:

- security:
  - encryption algorithm (if any);
  - encryption algorithm parameters (key length, number or rounds, ...);
  - authentication algorithm (if any);
  - authentication algorithm parameters (key length, number or rounds, ...);
- speed of execution or deadlines;
- number of Functional Units required;
- priority.

In each system there are some *critical applications*; those applications are either necessary for the system to run, or fundamental for the system role. Non-critical applications can be denied running if their hard requirements cannot be met; critical applications cannot be denied running and the system must perform all the operations necessary for feeding the resources required to satisfy their hard requirements. This may include suspending or terminating non-critical applications.

The application requirements will be mapped to a set of security policies; some of these policies will be compatible with the current system status, others will not be.

The *system status* identifies both a hardware and a software configuration. Each system status defines a set of supported security policies.

At each given time a *threshold* cost for the current system status is also defined. This threshold represents the maximum cost that the system can afford for enforcing a set of policies. Such cost depends on the system status (e.g., battery status).

As shown in Figure 1, three different units are contributing to the adaptation algorithm: a *Monitor* (M), an *Analyzer* (A), and a *Reconfiguration Manager* (RM). The *Monitor* senses the changes in the parameters that are defining the system status. These parameters are related to the hardware, the software, and the environment. Requests for starting new applications or removing running applications are also received by the monitor. When a change in the system status or an application request is received by the *Monitor*, such information is sent to the *Analysis* module; this block analyzes this piece of data and computes a set of security policies that are compatible with the system status and with the application requirements. The cost of this set must be below the cost threshold defined for the system. The set of selected policies is then sent to the *Reconfiguration Manager* which has the ability to evaluate if a reconfiguration of the policies should be done or not. The following subsections describe the behavior of each module.

A. Monitor

The Monitor has to check the status of the system parameters. Their values are provided by a set of sensors and physical monitors. The values collected from the sensors are compared to specific thresholds defining the different system states. Thus the current state of the system is determined. The state of the system should be updated with proper frequency, based on the system status itself and on the policies. For example, if the battery level is low, it may be useful to raise the checking frequency of the battery level. Defining the system status may not be trivial due to the mutual influence of the different parameters. Let us suppose that only two parameters, X and Y, completely define the status of the system. For each one of them we can define a set of thresholds: $\{x_0, x_1, \ldots, x_M\}$ and $\{y_0, y_1, \ldots, y_N\}$ for X and Y, respectively. Thus, there are $M \times N$ different systems status. In many cases, only a subset of these status will be possible due to state cross-correlations. For example, it may happen that for certain values of X (e.g., in the range $\{x_i, x_j\}$) only a subset of values of Y are possible (e.g., Y need to be in the range $\{y_k, y_l\}$). This property can be exploited to reduce the set of system states to be considered during the analysis phase.

In summary, for designing the *Monitor* the following information are required:

- the list of all parameters that can influence the state of the system;
- the list of the possible correlations among parameters;
- the definition of the status thresholds of the parameters.

B. Analysis

The *Analysis* module receives the notifications about system status change and the requests related to newly started or terminated applications. The *Analysis* module analyzes the requests received and tries to find the best match between the security policies that can be implemented and the application requirements. The available policies are firstly analyzed to find the set of *applicable policies*; these policies are the ones that are compatible with the present system status (i.e. they require resources that can be provided by the system). Finally, the *Analysis* module finds a match between the applicable policies and the applications.

For example, let us suppose that the twelve security policies shown in Figure 2 are available. Let us also suppose that the system status determines the set of applicable policies named *older-system-policies* in the same figure.

Later, a change in the system status produces a new set of applicable security policies (*newer-system-policies* in the figure) that is composed by:

$$S_{sys-old} = \{P_1, P_2, P_3, P_4, P_9, P_{10}\};$$
$$S_{sys-new} = \{P_1, P_2, \ldots, P_6\};$$

For example, a set of running or incoming applications can be:

$$app = \left[\underline{A_1}, \underline{A_2}, A_3, A_4, A_5\right];$$

where underlined applications are considered critical. Each application comes with its own security requirements which can be either hard or soft. Security policies that satisfy all the hard requirements are named *essential* (P_x). Policies that

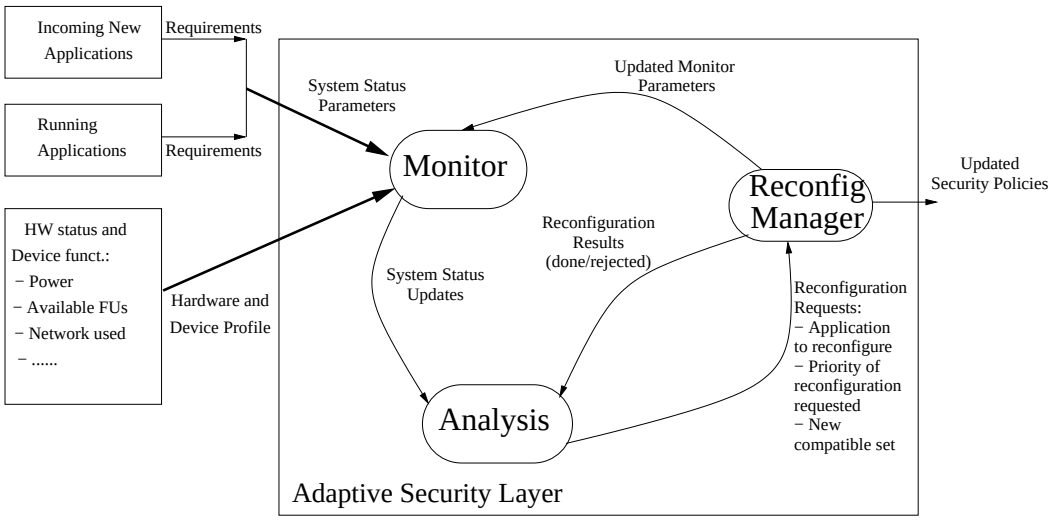

Fig. 1. Security policy adaptation scheme.

are also satisfying at least one of the soft requirements are named *desirable* ($\overline{P_y}$). The following expressions formalize the description of the essential and the desirable policies:

$$P_{essential} \rightarrowtail \{HardReq\}^{*};$$
$$\overline{P}_{desirable} \rightarrowtail \{HardReq\}^{*} \oplus \{SoftReq\}^{n};$$

where $\rightarrowtail$ represents the relation: P_x *satisfy* $\{AppReq\}$; $*$ means that all requirements are satisfied; n is a positive exponent indicating the number of soft requirements covered by the corresponding security policies. For example, let us assume that the *Analysis* module found the following sets of security policies for the five applications above:

$$A_{1,req} \Rightarrow \left[P_1, \widehat{\overline{P_2}}, \overline{P_5}\right];$$
$$A_{2,req} \Rightarrow [P_1, P_4];$$
$$A_{3,req} \Rightarrow \left[P_5, P_6, \widehat{P_{10}}\right];$$
$$A_{4,req} \Rightarrow \left[\widehat{P_4}, \overline{P_5}, P_6\right];$$
$$A_{5,req} \Rightarrow \left[P_7, \widehat{P_9}\right];$$

The current active application security policy used by each application, within the older set of supported policies ($S_{sys-old}$), is marked with $\widehat{P_x}$ (e.g., $\widehat{P_9}$ for the application A_5).

Looking at the $S_{sys-new}$ set of Figure 2, we can see that P_9 and P_{10} are not compatible anymore with the system status. Also P_7 is not in the set of applicable policies. Looking at the application requirements, we can determine that P_3 is useless, as it does not satisfy any requirement of the present applications.

By considering the above data, the *Analysis* module simply computes the following sets of application security policies satisfying the applications requirements, defined as:

$$S_{satisfy} \subseteq S_{sys-new}$$

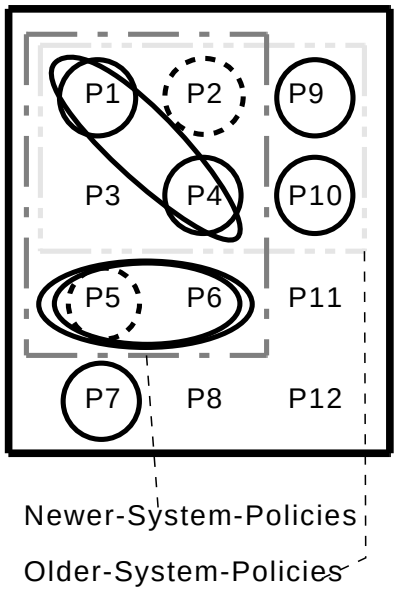

Fig. 2. Different sets of policies depending on the system status. Desirable policies are denoted by continuous lines; dashed lines represent desirable policies.

that is equal to:

$$S_{satisfy} = S_{essential} \cup S_{intermediate} \cup S_{desirable}$$

The set $S_{essential}$ contains the application security policies P_i marked as essential; $S_{desirable}$ contains the policies that are defined as desirable; $S_{intermediate}$ is composed by the policies that are essential for some applications and desirable for others. The intersection of these sets is empty. We define the *fair* set as:

$$S_{fair} = S_{intermediate} \cup S_{desirable}$$

In our example, the previous definitions give the following

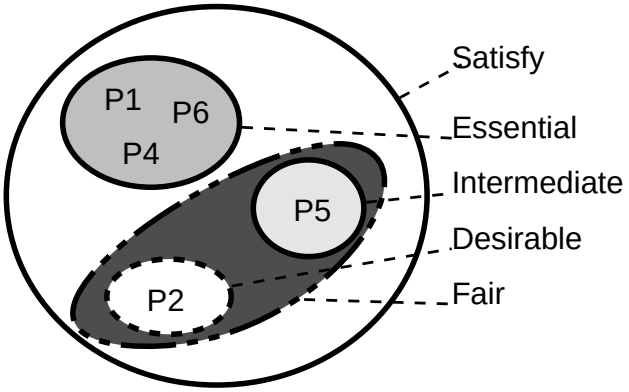

Fig. 3. Example of policies sets computed by the *Analysis* module.

TABLE I
THE INITIAL COVERAGE TABLE FOR THE S_{fair} SET (TABLE 1) AND FOR THE $S_{satisfy}$ SET (TABLE 2)

I.1:

	A_1	A_2	A_3	A_4
P_2	x			
P_5	x		x	x

I.2:

	A_1	A_2	A_3	A_4
P_1	x	x		
P_2	x			
P_4		x		x
P_5	x		x	x
P_6			x	x

sets:

$$S_{essential} = \left\{P_1^2, P_4^2, P_6^2\right\};$$
$$S_{intermediate} = \left\{P_5^3\right\};$$
$$S_{desirable} = \{P_2\};$$
$$S_{fair} = S_{intermediate} \cup S_{desirable} = \left\{P_2, P_5^3\right\};$$

The exponents are used to count the number of applications that are compatible with the associated policies. Figure 3 shows a graphical representation of the sets of our example. Once the $S_{essential}$, $S_{intermediate}$, and $S_{desirable}$ sets have been determined, a coverage algorithm similar to the Quine-McCluskey coverage table [10, pp. 220, 229] is run on the S_{fair} set. The coverage table corresponding to the previous example is shown in Table I.1. The rows of the table correspond to the different policies; the columns correspond to the different applications. A mark in the cell (i, j) means that the policy P_i is suitable for the application A_j (i.e., the policy P_i covers the requirements of the application A_j). If a possible solution is found by applying the coverage algorithm and its global cost is lower than the desired threshold, the selection process terminates. If a possible solution is not found, the coverage method is applied on the $S_{satisfy}$ set (that is $S_{essential} \cup S_{desirable} \cup S_{intermediate}$). If the coverage algorithm applied on the S_{fair} set provides a solution, but this solution has a cost that is over the threshold, a branch and bound method [11] is applied on the same set. If a certain set of policies is not able to provide a suitable solution (i.e., at least one application is not covered), the coverage table will have

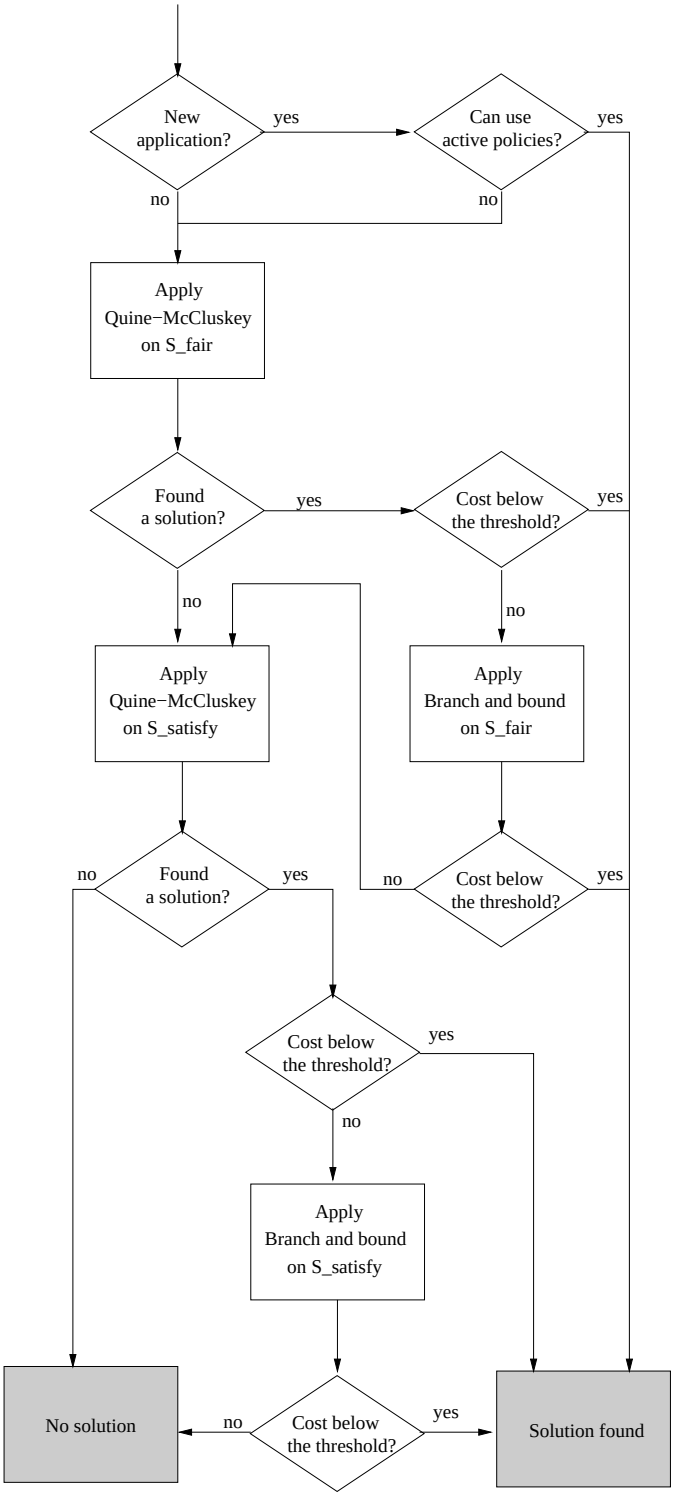

Fig. 4. Application of different algorithms to select a set of policies.

at least one column without any mark. Therefore, by verifying that no empty column is present in the table, the analysis module is able to determine if a solution for the considered set of policies is allowed. If the set of policies identified as a solution, has a cost over the system threshold, a new solution is computed by using a branch and bound method. The cost of the different policies and the cost threshold will be considered in the branch and bound method to optimize the search of the minimum. Figure 4 summarizes all the possible steps to determine the set of policies to be used.

The coverage method selects the policies that are associated with the highest number of applications. Thus, it allows saving resources in implementing the policies. The provided solution may not be the optimal one in terms of global cost. The branch and bound method is slower, but it provides an optimal solution in term of cost. Whenever a solution is not found and at least one application has no policy associated with it (i.e., there is at least an empty column in the table of coverage when the $S_{satisfy}$ set is considered), the related applications are discarded and the search of a solution is restarted. If one of the discarded applications is critical, the system should go in an error state. If a solution is not found due to a cost above the threshold for of all the possible solutions, one of the non critical application has to be suspended or removed from the system. To study a way to select which application to remove is beyond the scope of this paper.

In our example, the coverage algorithm applied on S_{fair} does not find a solution (i.e., not all the applications can be covered by the policies contained in that set); therefore, the coverage algorithm is applied on the larger set ($S_{satisfy}$). P_1 and P_5 or P_4 and P_5 are selected by applying this procedure.

If for some reason the system threshold is changed, the *Monitor* will inform the *Analysis* module and will provide the new system threshold. The *Analysis* module will check if the current set of policies still satisfies the new conditions. If not, the set of policies will be recomputed by following the algorithm described above.

When a request related to a new application is received, the *Analysis* module computes the sets of security policies that are satisfying the new application requirements; then, if these requirements are satisfied by the active policies and the updated security cost (see Equation 1) is below the system threshold, no reconfiguration is necessary. If any of these conditions is not met, the *Analysis* module will recompute the set of policies by applying the algorithm described above. When a terminate application request is received, a new set of policies is computed to optimize the system performance.

At the end of *Analysis* module operations, the set of selected policies, along with the cost of the new configuration and the strength of the request is sent to the *Reconfiguration Manager* (RM). A reconfiguration request can be weak or strong. Weak reconfiguration requests are typical for changes in the system conditions (e.g., the battery is draining); these request lead to reconfigurations in a short amount of time. Strong reconfiguration requests usually happen when new applications are added to the set of currently running applications. This kind of requests should be satisfied immediately. A request for updating the cost of the currently used set of policies can be also issued to *RM*. In fact, such a situation can happen when an application is added or removed from the set of running application, but no change in the set of policy is required.

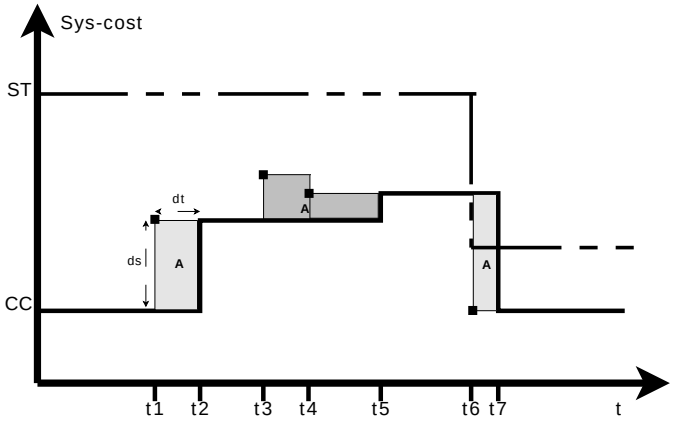

Fig. 5. Example of adaptation diagram.

C. Reconfiguration Manager

The *Reconfiguration Manager* receives the possible reconfiguration requests from the *Analyzer* module; it checks the reconfiguration request and, if necessary, it reconfigures the system. The reconfiguration requests contain the new set of selected policies, the list of applications that are associated with them, the cost of the new set, and the cost threshold. *RM* decides if a reconfiguration has to take place, depending on the current configuration and its cost. As explained before, the reconfiguration request may be strong or weak. In the first case, the reconfiguration must occur immediately. A strong request may happen when new applications, that cannot be run with the current set of policies, need to be run. When the reconfiguration request is weak, the old configuration is maintained by *RM* for a time given by a constant divided by the difference between the current cost and the one of the new policy. Figure 5 represents this behavior. In this figure, ST is the system threshold, CC is the cost of the current configuration; ds is the difference between CC and the cost of the new policy; A is a constant (to be determined experimentally) such as $ds \times dt = A$. Therefore, dt is the time for which the old configuration is allowed to be maintained ($dt = A/ds$). The higher the difference between the cost of the new configuration and the cost of the current one, the lower the time that the current configuration is allowed to be used after the reconfiguration request is received. This mechanism is valid both for reconfigurations triggered by changes in the cost threshold and for the ones triggered by changes in the system status. Reconfigurations are not applied immediately to filter out reconfiguration requests that last only for a short time.

IV. Scenario Example

In this section a case study is proposed. The methodology described in the previous section, is applied on this sample system. In this section we show how the self-adaptive security system can adapt the security settings in response to a change in the internal and/or external conditions.

The considered application is a software that requires, as a hard requirement, message authentication and, as a soft requirement, message encryption. The application is unaware of the security protocols and algorithms that the system will use to satisfy its requirements.

Let us suppose that the system supports the IPSec protocol [12]; this protocol can be used for creating a secure channel. Let us also suppose that the AES algorithm (with 128 and 256 key length) can be used for encryption and the HMAC-SHA-1 and HMAC-SHA-2 algorithms can be used for authentication. For the shake of simplicity, let us assume that a different security policy is used for each security algorithm; policy costs are computed considering the power consumption of the algorithms. In our case, the power consumption cost of HMAC-SHA-2 will be higher than HMAC-SHA-1 as well as for the AES-256 and AES-128. The cost threshold, in this simple case, will only include power consumption.

Our self-adaptive system will adapt the security algorithms based on the available battery energy. Let us suppose that in the initial status the system has the full battery energy available; let us also suppose that the system requirements are satisfied by the HMAC-SHA-1 and by the AES-128 algorithms. Here follows a list of possible reconfiguration events that may occur during the system lifetime. Example of possible events that will trigger adaptations are:

- a new incoming application may require a higher security level; the system will reconfigure to support HMAC-SHA-1 and AES-256, given that their cost is under the cost threshold;
- a change in the battery level can decrease the cost threshold; thus, the system may go back to the AES-128 algorithm;
- if, for a system failure the AES algorithms are not available any more, the applicable policies are affected. The applications that are using all the AES-based policies will be terminated. Our reference application can still run because its hard requirements are still satisfied by using only HMAC-SHA.

As shown in this simple scenario, the adaptation is transparent for the application.

V. Conclusions and Future Work

In this paper we have proposed a comprehensive and domain-independent methodology for system security self-adaptivity. Security policy self adaptivity allows the system to keep the same security level even when the internal and/or external conditions change.

Future work have to be put in further refining the self-adaptation scheme, in studying the parameters that have to be monitored, and in defining the precise contents of the application security policies. The security self-adaptive system requires to be simulated and its parameters require to be tuned according to the results of the simulations.

VI. Acknowledgments

This work was partially supported and funded by the European Commission under the Project AETHER (No. FP6-IST-027611). The paper reflects only the authors' view; the European Commission is not liable for any use that may be made of the information contained herein.

References

[1] L. Bisdounis, P. Bonnot, K. Nasi, S. Koutsomitsos, A. Taddeo, A. Ferrante, O. Heron, and M. Danek, "A self-adaptive embedded technologies for pervasive computing architectures; applicative scenarios and basic requirements. Metrics definition," European Project deliverable D4.1.1, 07 2006, AEther EU Project, Contract Number: IST-027611.

[2] L. Marcus, "Introduction to logical foundations of an adaptive security infrastructure," in *Proceedings of FCS'04 Workshop on Foundations of Computer Security*, ser. General Publications, A. Sabelfeld, Ed., vol. ISBN: 952-12-1372-8, no. 31, June 2004, pp. 251–266. [Online]. Available: http://www.tucs.fi/publications/attachment.php?fname=G31.pdf

[3] A. Shnitko, "Pratical and theoretical issues on adaptive security," in *Proceedings of FCS'04 Workshop on Foundations of Computer Security*, ser. General Publications, A. Sabelfeld, Ed., vol. ISBN: 952-12-1372-8, no. 31, June 2004, pp. 267–282. [Online]. Available: http://www.tucs.fi/publications/attachment.php?fname=G31.pdf

[4] Y. Xu, L. Korba, L. Wang, Q. Hao, W. Shen, and S. Lang, "A security framework for collaborative distributed system control at the device-level," in *Industrial Informatics, 2003. INDIN 2003. Proceedings. IEEE International Conference on*, 21-24 Aug. 2003, pp. 192–198. [Online]. Available: http://ieeexplore.ieee.org/iel5/9109/28887/01300269.pdf

[5] S. Alampalayam and A. Kumar, "An adaptive security model for mobile agents in wireless networks," in *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, vol. 3, 1-5 Dec. 2003, pp. 1516–1521vol.3. [Online]. Available: http://ieeexplore.ieee.org/iel5/8900/28134/01258491.pdf?tp=&arnumber=1258491&isnumber=28134

[6] C. Chigan, L. Li, and Y. Ye, "Resource-aware self-adaptive security provisioning in mobile ad hoc networks," in *Wireless Communications and Networking Conference, 2005 IEEE*, vol. 4, 13-17 March 2005, pp. 2118–2124Vol.4. [Online]. Available: http://ieeexplore.ieee.org/iel5/9744/30731/01424845.pdf?tp=&arnumber=1424845&isnumber=30731

[7] A. Dandalis and V. K. Prasanna, "An adaptive cryptographic engine for internet protocol security architectures," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 9, no. 3, pp. 333–353, 2004.

[8] M. El-Hennawy, Y. Dakroury, M. Kouta, and M. El-Gendy, "An adaptive security/performance encryption system," in *Electrical, Electronic and Computer Engineering, 2004. ICEEC '04. 2004 International Conference on*, 5-7 Sept. 2004, pp. 245–248.

[9] G. Gogniat, T. Wolf, and W. Burleson, "Reconfigurable security architecture for embedded systems," in *Proceedings of the 39th Hawaii International Conference on System Sciences, HICSS 2006 / MOCHA 2006*, vol. Mobile Computing Hardware Architectures: Design and Implementation Design Symposium (MOCHA 2006), Kauai, Hawaii, USA2, January 4 2006. [Online]. Available: http://www.ecs.umass.edu/ece/wolf/pubs/2006/hiccs2006.pdfs

[10] Edward J. McCluskey, *Logic Design Principles*. Prentice-Hall Inc., 1986.

[11] A. H. Land and A. G. Doig, *An Automatic Method for Solving Discrete Programming Problems*. Econometrica, 1960, vol. 28, pp. 497–520.

[12] R. Yuan and W. T. Strayer, *Virtual Private Networks*. Addison Wesley, 2001.