Self-Optimization of Secure Web Services

Valentina Casola [a] Emilio P. Mancini [b] Nicola Mazzocca [a]
Massimiliano Rak [c] Umberto Villano [b]

[a] *Dipartimento di Informatica e Sistemistica, Università degli Studi di Napoli "Federico II", Via Claudio 21, 80125 Napoli, Italy*

[b] *RCOST-Dipartimento di Ingegneria, Università del Sannio, Via Traiano 1, 82030 Benevento, Italy*

[c] *Dipartimento di Ingegneria dell'Informazione, Seconda Università di Napoli, Via Roma 29, 81031 Aversa (CE), Italy*

Abstract

Service-oriented architectures (SOA) and, in particular, Web Services designs are widely adopted for the development of interoperable systems. In a dynamic scenario, a composed service may exploit component services in order to complete its task; composed services are variously distributed, and offered by different providers in different security domains and under different load conditions. So, the development of services and their integration entails a huge number of design choices. Obtaining optimality for all of the involved parameters for composed services is a challenging and open issue. In this paper we present MAWeS, an autonomic framework that makes it possible to auto-configure and to auto-tune the composition of services, guaranteeing optimal performance and the fulfillment of given security requirements. We will illustrate the framework architecture and how it is able to support the development of self-optimizing autonomic services on the basis of two evaluation services, the first one able to predict the performance of different services execution, the second one able to evaluate the security level provided by a service.

Key words: self-optimization, Web Services, performance prediction, security evaluation

Email addresses: Corresponding Author: valentina.casola@unina.it (Valentina Casola), epmancini@unisannio.it (Emilio P. Mancini), nicola.mazzocca@unina.it (Nicola Mazzocca), massimiliano.rak@unina2.it (Massimiliano Rak), villano@unisannio.it (Umberto Villano).

1 Introduction

Service Oriented Architectures (SOA) enable the integration of loosely-coupled and reusable services for on-demand applications [1, 2]. Services are independent software components that can be accessible through standard interfaces and protocols, such as SOAP. Services can be atomic, if they are able to offer some functionalities without invoking other services. Otherwise, they are composed. Composed services may be made up of both atomic and composed services provided by different providers, according to a specific workflow. The providers are distributed over the networks and, in general, different providers may offer the same service with different functional and non-functional properties (e.g., performance, throughput, security, trustability, availability and fault-tolerance). Indeed, services can be developed with different technologies, hosted on different platforms and possibly in untrusted domains. In light of the above, the development of services and their integration entails a huge number of design choices. Obtaining optimality for all of the involved parameters for composed services is a challenging and open issue. Optimal performance is related not only to the specific services, but also to the systems that host them and to the way the mapping between service and systems is performed [3–7]. Performance-oriented management of services requires the ability to dynamically change service configuration and in a way transparent to the service offered to the end-user.

As for security issues, different methodologies and architectural solutions are available in the literature, to define and to assess security and trust [8–10]. Providing a secure service implies that all operations and the infrastructure itself are secure and trusted. However, the definition of trusted domains in a dynamic context as SOA, where a composite service can dynamically change its atomic services by finding them in a public registry, is a very complex and challenging task. A common approach is to describe security issues by means of security policies that express rules and constraints on technical and organizational aspects [11]. Policies are adopted to define secure communications among different domains and to establish trusted domains [12–14]. Available specifications let the user to accept or to refuse a communication with a provider, but they do not support him in dynamically choosing the most secure or the "optimal" services among available ones, against the requester's security requirements. This is primarily due to the lack of automatic tools that evaluate security quantitatively. In a dynamic scenario, a web application exploits component services in order to complete its task; composed services are variously distributed, and offered by different providers with different *security levels* and under different load conditions. So, different choices imply both different performances and different security levels provided by the composed service. The object of this paper is to present an autonomic [15–18] framework that makes it possible to auto-configure and to auto-tune the com-

position of services, guaranteeing optimal performance and the fulfillment of given security levels.

In previous papers, we have introduced MAWeS (MetaPL/HeSSE Autonomic Web Services) [19, 20], a framework whose aim is to support the development of self-optimizing autonomic systems for Web service architectures. It adopts a simulation-based methodology, which allows to predict system performance in different status and load conditions [21, 22]. The predicted and evaluated results are used for a feedforward control of the system, which self-tunes for optimal performance *before* the new working conditions are actually observed. In this paper we describe an extension of the MAWeS framework and its use for the development of autonomic SOA applications that optimize themselves in a proactive way, *both for performance and security*. This is an original approach, as customarily performance and security evaluations are considered separately in the literature. As for performance evaluation, in Web Services designs, abstraction layers completely hide the underlying system to all users, and classical techniques for system optimization (such as ad-hoc tuning, performance engineered software development, ...) are not applicable. As for the security evaluation some methodologies have been developed that aim at expressing security by formal means and defining a security metrics to evaluate the security level offered by a service [23–25]. So the system autonomically self-tunes before the new load conditions are actually observed, settling in a configuration that guarantees optimal performance without decreasing the system security level. In addition to the existing tools, adopted by MAWeS to simulate multiple system configurations and to choose the performance-optimal one, a new security evaluation component has been developed that provides information on the security level of the various system configurations. This component is based on a a policy-based approach, and relies on the methodology previously published in [26].

The remainder of this paper is structured as follows. Sections 2 and 3 describe the security and performance evaluation issues in Web Services Architectures and introduce the HeSSE/MetaPL prediction methodology and the security evaluation methodology. Section 4 describes the MAWeS framework with the new security extension, explaining how it can be used to choose the best configuration in terms of both performance and security requirements. In particular, a detailed description of the framework services is presented. In Section 5, the MAWeS autonomic approach is dealt with, describing how to build autonomic services by extending a MAWeS client. In Section 6, a case study for the proposed framework is shown, describing the optimization process of a secure web services application. Finally, some conclusions are drawn and the directions of our future work are outlined.

2 Self Optimization and Security

In a typical real-world scenario, a web application exploits component services in order to complete its task. Component services could be atomic or composed of other services, and they could be variously distributed and, possibly, offered by different providers with different *security levels* and under different load conditions.

Figure 1 tries to summarize the problem that we wish to address. It shows two main actors, a service requester (R) (the web application or the compound service) looking for a specific component service, and a set of service providers (P_i) offering such service. Each service provider offers the service with different security levels, exposing them through published policies, the *Offered Policies (OPs)*. Performance parameters are not included in the policies, but need to be evaluated or predicted with external tools. The requester searches for a service that fulfills a set of desired security requirements expressed by means of a policy, the *Requested Policy (RP)*, and that optimizes the overall response time.

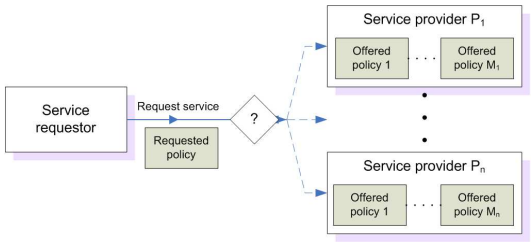


Fig. 1. The problem of trusted security requests

The main questions to be answered are:

(1) Does a service exist that respects the security requirements expressed by R?
(2) If multiple configurations are available for such service, which is the best in terms of performance and security?

In the literature these questions have been almost separately discussed; our proposal is to address the above issues together with the use of the MAWeS framework. In the security-extended version described in this paper, MAWeS is able to support the development of self-optimizing autonomic systems for Web Service architectures on the basis of services able to predict the performance (response time) of different services execution and to evaluate the security level provided. So, the proposed approach is based on three main elements:

(1) a policy-based methodology to evaluate security and a service to implement it (*Security evaluation service*);

(2) a simulation-based methodology to evaluate the performance and a service to implement it (*Performance evaluation service*);
(3) a framework to choose the best service configuration on the basis of evaluation results (*Core service*);

Both security and performance evaluation methodologies have been already presented by the authors in some previous papers [21,22,26], and they will be summarized in Section 3. This paper, instead, is focused on the architecture of the framework and its practical use. The MAWeS framework gives autonomic features to services thanks to two important features:

(1) It provides services to evaluate security and performance (*Evaluation Service*) and, furthermore, it provides a decision service (*Core Service*) that is able to look for available target service configurations, evaluate their performance and security features, choose the best one and give back to the target application/service the correct configuration parameters to optimize itself.
(2) The optimization process, and the system itself, is completely transparent to the target application/service thanks to a *Front End* that acts as a client for the core service on behalf of the application; this lets the application/service be autonomic.

To this aim, the proposed framework is structured in three layers, named *Front End*, *Core* and *Evaluation*. *Core* and *Evaluation* are in charge of evaluation and decisional tasks, and are implemented by a service-based architecture. The *Front End* is the part of the framework that must be integrated into the user application and allows to access the framework services, thus enabling autonomic behavior.

3 Evaluation Methodologies

Before showing the detailed description of the architecture and its services, we will illustrate here the two methodologies on which the MAWeS framework is based.

3.1 Evaluation of security

The methodology implemented to evaluate security is the Reference Evaluation Model (REM) [26], its goal is to provide an automatic mean to state the security level provided by an infrastructure. The methodology defines how to express in a rigorous way the security policy, how to evaluate a formalized

policy, and how to state the provided security level. Any policy is represented through a tree, which contains all the policy provisions (intermediate nodes and leaves).

In Figure 2 the three methodology phases are shown: **Policy Structuring**, **Policy Formalization** and **Policy Evaluation**:

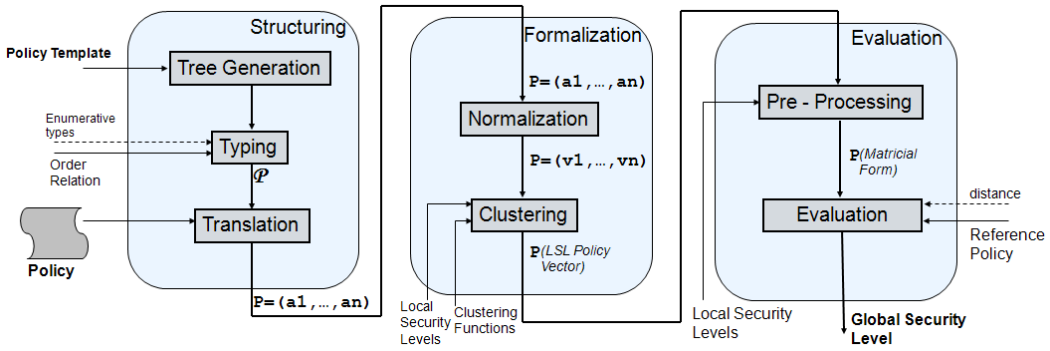


Fig. 2. Phases of the evaluation methodology

(1) The goal of the **Structuring** phase is to associate an enumerative and ordered data type K_i to the n leave-provisions of the policy. A policy space "P" is defined as P $= K_1 \times K_2 \times \ldots \times K_n$, i.e. the vectorial product of the n provisions K_i. The space is defined according to a policy template that strongly depends on the application context.

(2) The main goal of the **Formalization** phase is to turn the policy space "P" into an homogeneous space "PS". This transformation is accomplished by a normalization and clusterization process which allows to associate a Local Security Level (LSL) to each provision; after that the provisions may be compared by comparing their LSLs.

(3) The main goal of the **Evaluation** phase is to pre-process the "PS" vector of LSLs in order to represent it by a $n \times 4$ matrix whose rows are the single provisions K_i and the number of columns is the chosen number of LSLs for each provision. For example, if the number of LSL is four and the LSL associated to a provision is l_2, the row in the matrix associated to the provision in the matrix will be: (1,1,0,0). Finally, a distance criteria for the definition of a metric space is applied. REM adopts the Euclidean distance among matrices:
$d(A,B) = \sqrt{(\sigma(A-B, A-B))}$
where $\sigma(A-B, A-B) = Trace((A-B)(A-B)^T)$

To define the Global Security Level L_{Px} associated to the policy P_x, we have introduced some reference levels and adopted the following metric function:

$$L_{Px} = \begin{cases} L_0 \ iff d_{x0} \leq d_{10} \\ L_1 \ iff d_{10} < d_{x0} < d_{20} \\ L_2 \ iff d_{20} < d_{x0} < d_{30} \\ L_3 \ iff d_{30} < d_{x0} < d_{40} \\ L_4 \ iff d_{40} \leq d_{x0} \end{cases}$$

where $d_{i,0}$ are the distances among the references and the origin of the metric space (denoted as $\emptyset$). This function gives a numerical result to the security; the idea is to evaluate the security associated to an infrastructure through the evaluation of its security policy.

The GSL is a measure of the security provided by an infrastructure according to its security policy; it is obtained by formalizing the process that is manually performed by security experts while trying to extend trust to other domains. The details of the methodology are out of the scope of this paper, and they can be found in [26].

3.2 Evaluation of performance

Frequent reconfigurations are one of the peculiar characteristics of Distributed Heterogeneous Systems (DHS). This is a very common situation in GRID or in Web Services architectures where target systems are very likely to be expanded, as in high performance clusters. As far as performance prediction is concerned, in all the cases mentioned above, the target architecture is highly variable, while the software is fixed. In other cases, both software and hardware may change. Aiming at managing this complexity, the HeSSE/MetaPL performance evaluation methodology tackles the problem of assessing automatically the effects of different hardware/software configurations on the overall performance of a given application (usually measured in terms of response time). The methodology adopts a two-level modeling approach:

System level models the DHS architecture. Hardware components, operating systems and middleware are usually modeled as system components; the system level is modeled through simulation objects, able to reproduce all the actions that have performance effects during system evolution (i.e., when applications are executed). System simulations are performed in the HeSSE simulation environment.

Behavioral level models the system evolution. Application and process cooperation are usually modeled at this level. The methodology adopts the MetaPL prototype language [22, 27] for the modeling of this level.

HeSSE (*He*terogeneous *S*ystem *S*imulation *E*nvironment) is a simulation framework that allows the user to simulate the performance behavior of a wide range of distributed systems for a given application, under different computing and network load conditions.

The HeSSE compositional modeling approach makes it possible to describe Distributed Heterogeneous Systems by interconnecting simple *components*. HeSSE uses traces to describe applications behavior. A trace is a file that records all the actions of a program that are relevant for simulation. Each trace is the representation of a specific execution of the parallel program.

Trace file are usually obtained through application instrumentation, when the application is still being developed. Moreover, they can be generated using prototypal languages, such as MetaPL [22, 27]. This defines a program description notation language-independent, and can support different programming paradigms or communication libraries. The MetaPL core can be easily extended to introduce new constructs into the language. Starting from a MetaPL program description, a set of extensible filters enables to generate different program *views*, and among the others, the trace files that can be used to feed the HeSSE simulation engine [21, 22].

MetaPL and HeSSE are independent modeling tools, but can be easily integrated through a three-steps methodology:

System Description in which we model the system to evaluate:
- MetaPL metacode production (Application Description);
- system architecture model definition (System Architecture Modeling);
- evaluation of timing parameters (Model Tuning).

Generation MetaPL filters generate HeSSE traces, following users indication about the parameters to evaluate

Evaluation HeSSE simulates the traces, predicting the performance parameters.

During the *System Description* phase, users are involved mainly in application description, which mainly consists of MetaPL prototypes development to generate the trace files needed to drive the simulated execution, and in system architecture model definition, which consists of the choice (or the development, if needed) of the HeSSE components useful for simulation, that are later on composed through a configuration file. The model tuning step, consists of running benchmarks on the target system, in order to enrich the simulator and the MetaPL description with parameters typically related to timing information. During the *Generation* Phase, users have just to give the tools a set of information useful to define the actual execution conditions to be evaluated (application parameters, system loads, ...). Note that the tool fully automatizes this step. The last phase (*Evaluation*) gives the user an evalua-

tion of the chosen execution condition in terms of the performance parameters chosen (e.g., response time, throughput).

Repeating the *Generation* and *Evaluation* phases, and so performing several trace generation and system simulations (each of them needs few milliseconds), we can compare the different results, to ascertain which is the best configuration and which are the optimal parameters that lead to highest performance.

4 MAWeS Architecture

In this section we describe the architecture of the MAWeS framework, which has been enriched to deal with both self-optimization and trusting in an integrated way. As illustrated in Figure 3, it is logically structured in three layers:

Frontend Layer: made up of the software modules or client services used by the requester (both web applications and other services) to access the MAWeS Core service;
Core Layer: composed of the core service that makes optimization decisions;
Evaluation Layer: made up of the services for performance and security evaluation, which support optimization decisions.

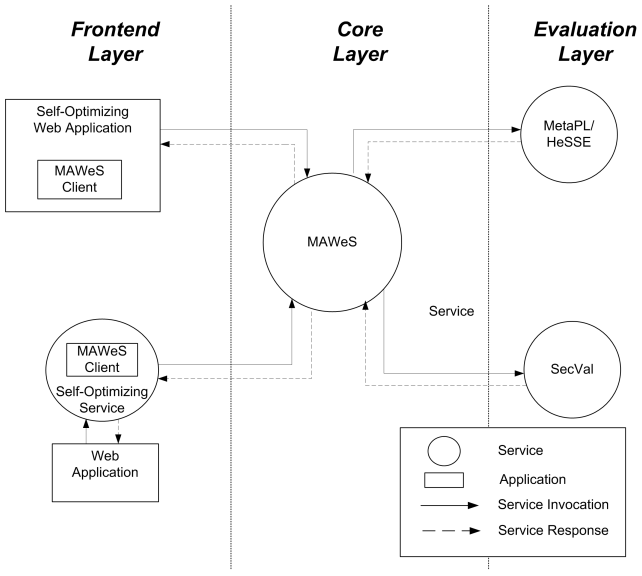


Fig. 3. MAWeS main Services

In the following we will describe in details the Core and Evaluation services, while the Frontend layer will be illustrated in the next section.

The MAWeS Core exploits environment services to monitor the system status and the available service configurations and it asks the evaluation services to find out the optimal execution conditions (in terms of both performance and security).

The MAWeS Core is subdivided in three logical components: **Interface Unit**, which receives the MetaPL document and retrieves the optimization information, the **Decision Unit**, which controls the execution of the evaluation processes in order to make decisions, and the **Evaluation Clients**, which invoke the Evaluation Services.

The MAWeS core can perform optimizations on-line, i.e. on application/service requests, according to strategies that will be discussed later. Furthermore, publication of available configurations or new available services, information needed to perform the optimizations, takes place off-line.

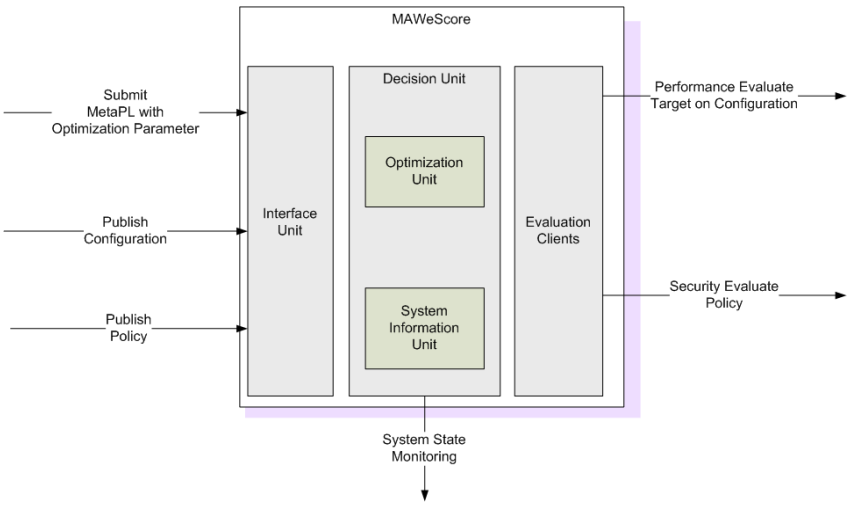


Fig. 4. MAWeS Core

Interface Unit The task of the Interface Unit (IU) is to find the parameters that may affect the application performance and to pass this information to the Decision Unit. In order to do so, it extracts from the MetaPL application description all the information that may possibly be used to optimize the application. The MetaPL description explicitly declares the optimization target by means of the `Target` element, whose attribute `kind` specifies if the target is to maximize a `performance` metric (default), or to guarantee a desired minimal security level (`GSLmin`). A single MetaPL description may contain more than one `Target`. Furthermore, the service requester (web application or composed service) needs to indicate explicitly the type of security credentials that it intends to adopt. This information will also be used to choose the correct simulation configurations. To include the information on credential, MetaPL

has been enriched with a new element, `credential`, whose attribute `kind` specifies the type of credential. At the state of the art, our working prototype supports login/password credentials (`pwd` value), digital `certificate`, or no credential (`nothing`).

Evaluation Clients Evaluation Clients are a set of clients that let the MAWeS core to interoperate with the Evaluation Services, which perform performance predictions and policy evaluations. They are the MH Client and the SEC Client.

The MH (MetaPL/HeSSE) Client is a software unit that implements a client for the MetaPL/HeSSE web service. Its implementation is threaded, in that each client instance runs as a new thread. This makes it possible for the MAWeS core to start in parallel as many MH clients as necessary. Each client invokes the services needed to perform the simulation and analysis for each different configuration, and gets the corresponding results. The Decision Unit collects all the results, compares them and makes its decisions.

The SEC Client is a software unit that implements a client for the Security Evaluation web service; it is usually invoked off-line to evaluate the GSL of each published service that will be stored in the System Information Unit (SIU).

Decision Unit

The Decision Unit (DU) contains all the framework autonomic intelligence. It applies the optimization rules defined by the framework administrator to optimize the target applications by means of the feedforward approach.

The DU is made out of two different units: the System Information Unit (SIU) and the Optimization Unit (OU). The former maintains information about the system status, i.e., the list of valid configurations with their GSLs. The latter requests to the MH client to simulate all admissible configurations and compare them.

In particular, the **Optimization Unit** analyzes all possible configurations after their evaluation, and chooses the one with maximum performance among the ones that meet the minimal security requirements. The OU requests to the SIU all available configurations that are accessible with the user-provided credentials, and whose GSL is greater than the requested GSLmin. It asks to the MetaPL/Hesse WS (through the MH client) the evaluation of all admissible configurations and, after that, it chooses the best configuration. It should be noted that in the current version of the framework we assume that all service policies are published and evaluated off-line by the System Information Unit. This exploits the Security Evaluation Service to perform the GSL of each available service, as explained later in this section.

The **System Information Unit** maintains all system and status information, as follows:

- available simulator configurations (which correspond to the different system configurations);
- security policies related to different configurations and the corresponding security level (GSL);
- associations between policies and configurations.

Furthermore, the SIU includes a **Monitoring and Discovery Unit** that updates the timing parameters for the simulator. It should be noted that the security monitoring problem is currently a particularly hot topic; we assume that the compliance of policies with systems is assured by a Trusted Third Party, and that they are periodically verified by humans.

4.2 Evaluation Services

The evaluation Layer includes a set of Web Services that make it possible to obtain performance predictions and security evaluation of the submitted configurations. The performance prediction services that exploit the HeSSE simulator and apply the MetaPL filters are described in detail in [19, 20]. As already discussed, the security Evaluation service implements the security evaluation methodology proposed in [26] and allows to evaluate the Security Level provided by a service or by a system from its published policy. It receives the policy, described according to WS-Policy language, and applies the structuring, formalization and evaluation processes to return the GSL that will be stored in the SIU.

5 The MAWeS Autonomic Approach

The MAWeS services can be used for building predictive load-balancing system, to optimize, both dynamically and statically, services and applications. They are used to give autonomic features to any web application or web services. The basic idea is to provide to developers a set of tools which can be (almost) transparently integrated into their application in order to made them autonomic. The MAWeS Frontend is the tool which enables the autonomic self-optimization feature in the target system. The MAWeS Frontend can be integrated in Web Services based applications at two different, non exclusive, levels:

Application Level: the autonomic system is able to affect user applications

by modifying the order of actions and their resource usage; at this level, the framework Frontend is integrated into the target application and it is able to adapt the application services invocation.

Service Level: the autonomic system affects the service, i.e., the optimization actions have impact on the tuning of parameters for composing services invocation (i.e. the invocation sequence, the load request to each server, ...); at this level the framework Frontend is integrated into the service and optimizes its configuration.

In both cases the MAWeS Frontend is a base class that implements the MAWeS client. Any new autonomic application or service inherits the class obtaining the autonomic features. Developers have to load the application/service MetaPL description in MAWeS Frontend (i.e., the MetaPL description File path is given to the MAWeS client base class), and the tool autonomically will provide application optimizations.

We have considered three different strategies for triggering the optimization process:

- *Deploy*: optimization takes place only at service deployment;
- *Service Call*: each time a client calls the service, it self-optimizes itself according to the current system status;
- *Reactive*: optimization is triggered by some particular events, such as a timer expiration or a status notification from a system monitor.

The first strategy, *Deploy*, makes the tool unable to react to status changes in the environment: once the service has been deployed and optimized, it is impossible to re-tune it. Hence it is suitable only for static systems, where the status conditions do not rapidly change during the whole application execution cycle.

The second strategy, *Service Call*, imposes high overhead to the service execution: every time the service starts, a new set of simulations are performed. It can be useful in the case of long-running services, where the overhead is dwarfed by computation times or, in general, when the advantages of performance optimization are higher than the introduced overhead.

The most flexible solution is the *Reactive* strategy. In this case, optimization process is performed only when it is really necessary, as a consequence of "significant" system status changes. The optimization trigger is a notification event defined by the application developer. It can be generated, for example, using a *workload profile*, that makes it possible to predict when the system workload *should* changes. From here onwards, we will adopt only the *Service Call* strategy.

As already mentioned, MAWeS self-optimization services may be offered at application or service-level. In the *Application-level optimization*, the web application searches for a set of atomic services that minimize its response time and meet given security requirements. This means that the application adapts itself on the basis of the prediction of the composed service performance. At Application level, the MAWeS Frontend works as a standard client integrated in the web application. It sends to the MAWeS core all the application-related information (performance metrics, behavior, security requirements, ...) needed to perform the simulations. In *Service level optimizations*, the MAWeS core aims at minimizing the response time of a self-optimizing service. In this case, the MAWeS frontend is integrated in the service, which invokes the MAWeS core in order to optimize its execution. The details about these optimizations can be found in [19, 20]. Figure 5 illustrates how MAWeS works "on-line": (1) the MAWeS Frontend submits the request to the core service, attaching the MetaPL document containing the GSLmin, the type of credentials and the application description. (2) The IU extracts from MetaPL the GSLmin, the credentials, the performance optimization target, the application parameters and the application description and forwards them to the Optimization Unit of the DU. (3) The Optimization Unit gets the parameters then it requests the available configurations to the SIU. The SIU extracts from the local database the list of valid configurations whose GSL, evaluated off-line, is greater than GSLmin. (4) the SIU returns the list of valid configurations. (5) the Optimization Unit starts the cycle of simulations, submitting the configuration and the MetaPL application description to the MH clients, which generate the traces and simulate them on the target configuration (5a and 5b). (6) The MH client returns the set of performance measurements for valid configurations. (7) The Optimization Unit chooses the best configuration and returns the corresponding parameters to tune the application to the IU. (8) The final results are sent to the Frontend. The grey box describes the "off-line" publication of configurations and policies, together with the policy evaluation process. When the new pair configuration/policy is published, the SIU forwards the policy to the SEC client, which performs the GSL evaluation and returns the value. The SIU stores the GSLs ordered by growing GSL values.

6 A case Study

The MAWeS framework is able to perform both security and performance optimizations on the basis of the MetaPL description of the application and of the security policy adopted by the composed services. In this section we illustrate the main features of the proposed framework, showing how it works

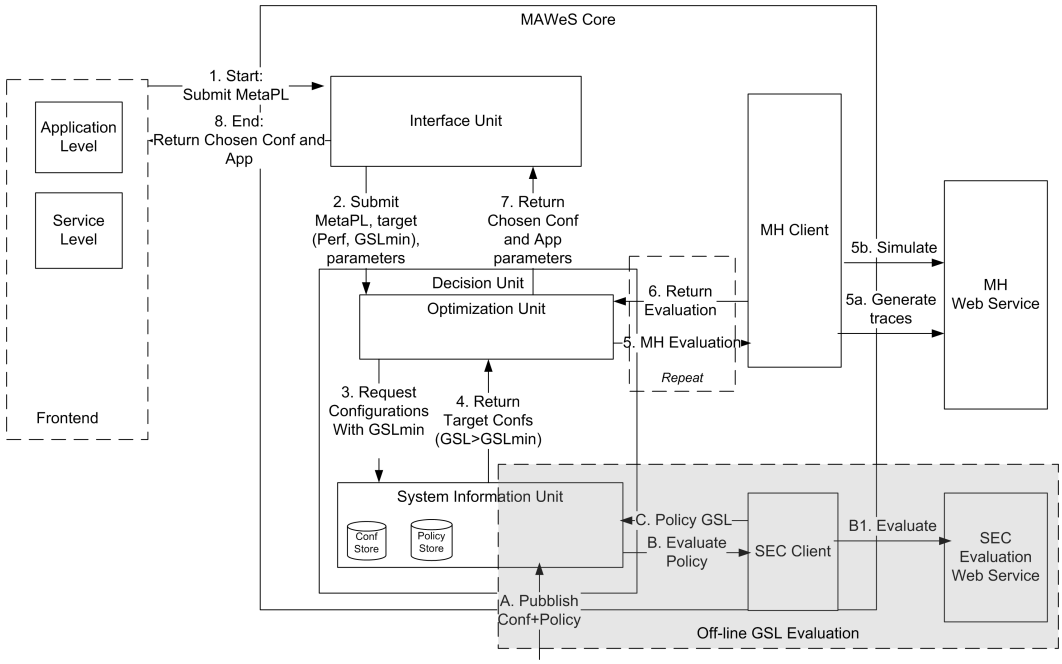


Fig. 5. MAWeS Execution Flow

while performing an *Application-Level* optimization. As case study, we have chosen an application for log files analysis, where the files may contain confidential data. Even if the application is relatively simple, it carries out complex tasks (analysis of large log files) and has to meet desired security requirements, according to user credentials and data confidentiality.

The application is composed of a client and of a set of services to store and to elaborate files, allowing the adoption of authentication mechanisms. The services enforce different security policies and implement different security mechanisms. According to the definition given previously of security level, we can say that the service can operate at different security levels. In other words, it is possible to search for, to access and to use a service with different security features depending on the user Requested Policy (RP). The RP contains information on the credentials the user has, on the confidentiality of the log file data and/or on other service level constraints.

Figure 6 describes the service-based application, without any security feature. The application sends the log file to a *storage service*, which returns an unique *FileID*. Every time that the application invokes the *elaboration services* in order to "evaluate" the log file (e.g., to search for DoS attacks, to search for statistics on a given IP address, and so on), it passes the *FileID* to the elaboration services. These access the original file, analyze it and return the results.

The invocation of multiple services is needed (the target application/service usually composes service in order to offer complex interactions) and can be

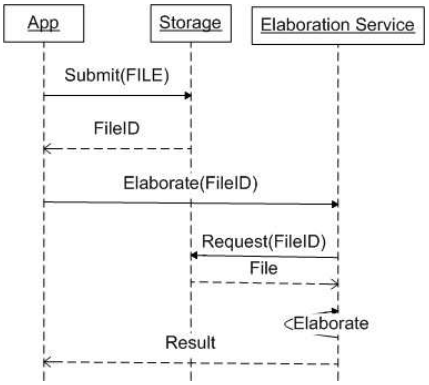


Fig. 6. Log Analysis Service Usage

performed in two ways: in sequence (Figure 7a), where a single thread performs in sequence the service invocations, or in parallel (Figure 7b), where one thread for each invocation performs the service call.

The three services (`Storage`, `El1`, `El2`) can be offered in three different versions (*basic*, *pwd*, *sign*), depending on the enforced security mechanism:

basic: data are not confidential, and they are sent as clear text on the communication channels. There is no authentication mechanism to access them on the storage;

pwd: data are confidential, and they are encrypted on the channels. Users and processes are authenticated by a username-password mechanism;

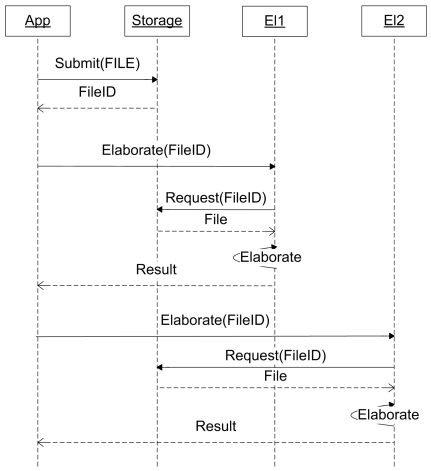
sign: data are confidential, and they are encrypted on the channels. Users and processes are authenticated by means of digital signatures.

Services can access a stored file only if they are invoked adopting the same authentication mechanism used for file storing.

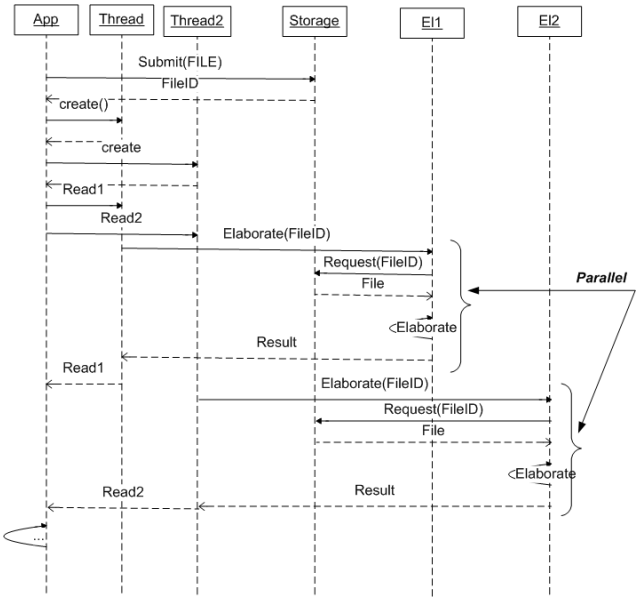
Authentication mechanisms are enforced by external authentication services that manage user credentials and attributes to perform a valid access control procedure, as an user directory with login and password or a digital certificate directory.

The secure-service version needs that the web application performs token acquisition as a preliminary task, before storing the file. Security tokens are built following WS-Security specifications, and may contain username and password or may be signed, depending on the authentication method chosen.

The application can request the token to an external server, or can produce it on its own. Once the token is available, it is attached to all service invocations, both for file storage and for elaboration requests. Every secure enhanced service checks the security token before performing its tasks. Each secure service holds its own security token; when a secure service invokes another service (for example, when the elaboration service invokes the storage service), it at-



(**a**)



(**b**)

Fig. 7. Multiple Service Invocations

taches its own security token to the request. Multiple services invocations can be performed in parallel or in sequence, in a way similar to the unsecure service shown in Figure 7. Figure 8 shows the token acquisition phase and the sequential invocation.

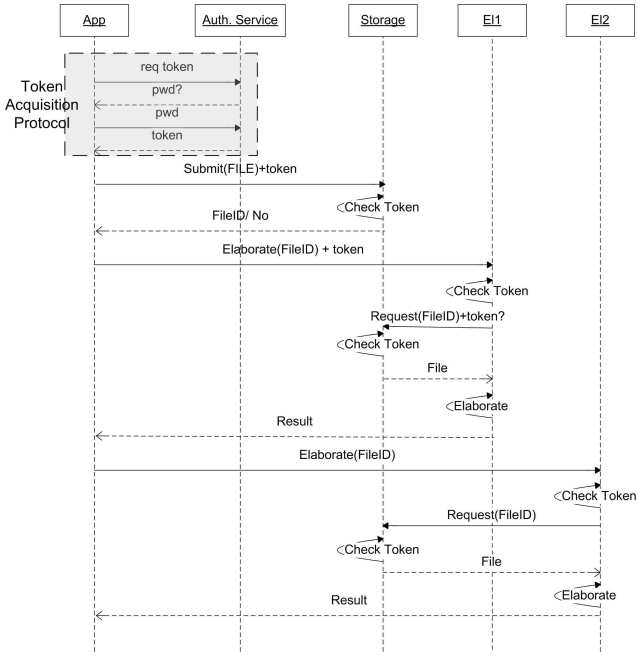


Fig. 8. Log Analysis Service Usage with Security Tokens

6.1 Autonomic LogAnalysis

In order to show the effect of the use of MAWeS framework, we will apply an application level self-optimization with a *Service Call* optimization strategy. This means that we will enrich the log analysis client with the MAWeS frontend, in order to perform optimization each time the client is started.

The changes to be done on the client for using MAWeS are very minimal. In the Java version, the logAnalysis class has to inherit the MAWeS frontend. The `main` procedure invokes a load method, which sets the path to the MetaPL description of the application behavior. It should be noted that the application has two possible implementations: single- or multi-threaded. The chosen behavior heavily affects the overall performance. In this scenario, MAWeS can act to predict the effect of parallelism on performance. Figure 9 shows the MetaPL description for the unsecure version of the application.

As for security parameters, they are specified in the service policy. For example, the policy in Figure 10, formalized according to the WS-Policy standard, specifies that the service implements security mechanisms to authenticate the sender of the SOAP message and to ensure the integrity of the SOAP message (i.e., to ensure that the SOAP message is not altered while in transit). In particular, the service can offer Integrity and Confidentiality services by encrypting the body message, and the illustrated policy has both Integrity and Authentication assertions. The details of the mechanisms are described in the corresponding tags (*KeyLength*, *KeyLocation*, ...).

```
<Code>
  <Task name="main">
   <Block>
    <Switch variable="Mode" >
      <WSCall ws="Storage" method="store" />
      <Case condition="MonoThread" />
        <WSCall ws="LogAnalysis" method="Elaborate1" />
        <WSCall ws="LogAnalysis" method="Elaborate2" />
      </Case>
      <Case condition="MultiThread" />
        <Spawn spawnedname"Thread" param="Elaborate1"/>
        <Spawn spawnedname"Thread" param="Elaborate2"/>
        <Wait />
      </Case>
     </Switch>
   </Block>
  </Task>
  <Task name="Thread">
   <Block>
     <WSCall ws="LogAnalysis" method="@param" />
   </Block>
  </Task>
</Code>
```

Fig. 9. MetaPL description of the Web Services client

Moreover, the interconnection network may have different configurations. We can introduce a network layer security protocol (TLS/SSL, for example) or distribute the nodes in different network LAN segments with different security features. For example, a set of services can be directly accessible from an open network, while others are accessible only inside a private network. Depending on the adopted security mechanism, the composed service provides a different security level, which is described by its policy. In Figure 11, the configurations and the corresponding policies are denoted as *Policy null*, *Policy pwd* and *Policy sign*, and they have been published in the MAWeS service SIU.

The last issue to be addressed is the definition of the available physical configurations. In fact, services may be deployed on a number N of nodes, and each of them may offer all or just a subset of the available services. For example, we may assume the existence of three nodes, each of them offers different services but accessible through a homogeneous security mechanism.

We have stored in the SIU the different configurations for the case study as different simulator configurations. Table 1 summarizes the configurations to be compared.

The number of available configurations can be fairly high. Being the simulation time independent of the configuration to be evaluated, the overhead linearly increases with the number of configurations to be compared. Our future work will focus on techniques for a *clever* automatic choice of *interesting* (i.e., possibly optimal) configurations.

Now the application is an autonomic self-optimizing application: each time it

```
<?xml version="1.0" encoding="UTF-8"?>
<wsp:Policy>

  <wsse:SecurityToken>
    <wsse:TokenType>wsse:SAMLAssertion</wsse:TokenType>
    <wsse:TokenIssuer>server</wsse:TokenIssuer>
    <ConfirmationMethod>
    urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
    </ConfirmationMethod>
    <KeyLength>2048</KeyLength>
    <KeyLocation>SmartCard</KeyLocation>
  </wsse:SecurityToken>

  <wsse:Integrity>
    <wsse:Algorithm Type="wsse:AlgCanonicalization"
          URI="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
    <wsse:Algorithm Type="wsse:AlgSignature"
          URI="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />

 <MessageParts
         Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
         wsp:Body()</MessageParts>
    <KeyLength>2048</KeyLength>
    <KeyLocation>SmartCard</KeyLocation>
</wsse:Integrity>

<wsse:Confidentiality>
 <wsse:Algorithm Type="wsse:AlgEncryption"
        URI="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
 <MessageParts
         Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
       wsp:Body()</MessageParts>
    <KeyLength>1024</KeyLength>
    <KeyLocation>Floppy</KeyLocation>
</wsse:Confidentiality>
</wsp:Policy>
```

Fig. 10. Policy corresponding to Level 3

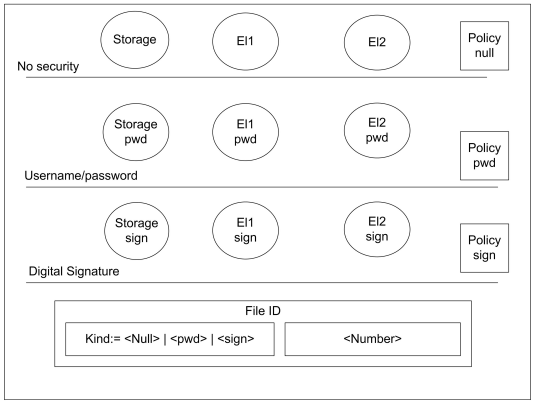


Fig. 11. Log Analysis Services Set and the file identifier

starts, the configurations that grant the requested security level requested are chosen, their execution are simulated (under the actual workload conditions) and the application is started in the optimal execution condition (the best configuration).

Config Name	Security	Servers	Description
Basic Distributed	none	3	one node hosts Storage and one node hosts Elaboration services
Basic Concentrated	none	1	a single node hosts both Storage and Elaboration services
Pwd Distributed	pwd	3	one node hosts Storage and one node hosts Elaboration services
Pwd Concentrated	pwd	1	a single node hosts both Storage and Elaboration services

Table 1
Configurations to be compared

6.2 Effects of self-optimization

In order to illustrate the effect of the adoption of the MAWeS framework in the case study application, we have analyzed the execution of the proposed application on the cluster Callisto at Second University of Naples. It is composed of an IBM x346 front-end (two Xeon 3.06 GHz, 4 GB RAM, 6x72 GB HD) and 40 nodes (two Xeon 3.06 GHz CPU and 2 GB of memory) connected with two Gigabit Ethernet and a Myrinet network.

At first we have performed a static analysis of the application, under known load conditions, varying the log file dimension and comparing real measurements and the simulation results adopted internally by the framework. This analysis is complex and out of the focus of this paper. We would just like to point out that the simulations can predict the mean response time of the application with a relative error under 20%. This is a sufficiently accurate information to enable the performance comparison of alternative configurations.

Then, we have run the target application instrumented with the MAWeS frontend. The framework is configured to compare all possible configurations with the *Service Call* strategy. It should be noted that this is the worst condition as far as overhead is concerned, since all the possible configuration are compared every time that the application is started.

Table 2 shows the results of the optimizations granting the *Pwd* authentication level. The first two table rows show the response time of two static choices (concentrated and distributed), the last row shows the response time of the autonomically-enhanced version of the application. The columns contains the dimension of the log file.

It should be noted that the autonomic version has always an additional overhead of about 0.5s, but it is always executed in the best possible configuration.

Application Optimization	1KB	10KB	40 KB	70 KB
Concentrated	0.92s	4.29s	8.5s	12.24s
Distributed	0.78s	4.25s	7.5s	10.01s
Autonomic	1.1s	4.7s	8s	10.5s

Table 2
Application response time

The overhead is independent of the log file dimension. The self-optimization features are not useful for files of small dimension, but they can be very effective for big files.

It should also be pointed out that the statical *Distributed* configuration has always the best response time. This is not a general rule, but depends on the system status (workload condition, node configuration, node hardware). The MAWeS framework always chooses the best configuration, while hiding completely this information from application users/developers.

7 Conclusions and Future Work

In this paper we have proposed an innovative approach for the development of self-optimizing autonomic systems for Web Services architectures with security requirements. The optimization process is based on the adoption of simulation for performance prediction and on a security evaluation methodology to guarantee the security level of the adopted services. The approach relies on the MAWeS autonomic framework, now enriched with security features. We have illustrated the two methodologies that are at the base of our approach and the new framework architecture and how it can be integrated in an application or service to make them autonomic.

The proposed framework opens a new way for the development of autonomic and security systems; it lets services and applications to self-optimize, guaranteeing a given security level, it is transparent to users and easily manageable by developers.

We are currently working on the definition of a system monitor to assess the *state* of all services and resources, in order to be able to optimize also resource allocation. Another open issue is the problem of monitoring the compliance between the security policy and the enforced security mechanisms; we are looking for quantitative parameters that could be measured by external tools. Furthermore, it is a fact that we have focused our attention on system functionalities, and not on the reliability and availability of the whole infrastructure. These topics should be addressed to avoid DoS or other attacks.

As for the efficiency of our framework, we argue that the time spent to choose and evaluate different configurations can be high when lots of alternative configurations are available. We intend to provide the system with a knowledge-based research functionality for a more clever selection of configurations to be evaluated.

References

[1] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard, Web Services Architecture, W3C Web Services Architecture Working group, February 2004.

[2] V. Balasubramanian, A. Bashian, Document management and web technologies: Alice marries the Mad Hatter, in: Commun. ACM, Vol. 41(7), ACM Press, 1998, pp. 107–115.

[3] S. Chandrasekaran, J. A. Miller, G. Silver, I. Arpinar, A. P. Sheth, Performance analysis and simulation of composite web services, in: Electronic Markets, Vol. 13(2), Routledge, USA, 2003, pp. 120–132.

[4] R. Elfwing, U. Paulsson, L. Lundberg, Performance of SOAP in Web Service environment compared to CORBA, in: Proc. of the Ninth Asia-Pacific Software Engineering Conference (APSEC02), IEEE Press, Washington, DC, USA, 2002, pp. 84.

[5] F. Cohen, Discover SOAP encoding's impact on web service performance, in: IBM DeveloperWorks, IBM Corp., 2003, `http://www-106.ibm.com/developerworks/webservices/library/ws-soapenc`.

[6] S. Chandrasekaran, G. Silver, J. A. Miller, J. Cardoso, A. P. Sheth, Web service technologies and their synergy with simulation, in: Proc. of the 2002 Winter Simulation Conference, ACM, San Diego, California, USA, 2002, Vol. 1, pp. 606–615.

[7] Y. Diao, J. L. Hellerstein, S. Parekh, J. P. Bigus, Managing web server performance with AutoTune agents., in: IBM Systems Journal, Vol. 42(1), IBM Corp., 2003, pp. 136–149.

[8] V. Casola, L. Coppolino, M. Rak, An architectural model for trusted domains in web services, Journal of Information Assurance and Security, Dynamic Publishers Inc., USA, Vol. 1(2), pp. 107-118.

[9] T. Beth, M. Borcherding, B. Klein, Valuation of trust in open networks, in: D. Gollman (Ed.), Computer Security - ESORICS '94, Lecture Notes in Computer Science, Springer-Verlag, 1994, No. 875, pp. 3–18.

[10] J. Bicarregui, T. Dimitrakos, B. Matthews, Towards security and trust management policies on the web, 2000.

[11] G. Della-Libera, al., Web services security policy language version 1.1, 2005.

[12] B. Atkinson, al., Ws-security specification, web services security version 1.0 05, OASIS, April 2002.

[13] A. S. Vedamuthu, al., Ws-policy specification, web services policy framework, W3C, September 2004.

[14] J. Chung, J. Zhang, L. Zhang, Ws-trustworthy: A framework for web services centered trustworthy computing, in: Proc. of IEEE international conference on Services Computing (SCC 04), IEEE Computer Society, Shanghai, China, 2004, pp. 186–193.

[15] K. P. Birman, R. van Renesse, W. Vogels, Adding high availability and autonomic behavior to web services., in: Proc. of 26th International Conference on Software Engineering (ICSE 2004), IEEE Computer Society, Edinburgh, United Kingdom, 2004, pp. 17–26.

[16] IBM Corp., An architectural blueprint for autonomic computing, IBM Corp., USA, 2004, `www-3.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf`.

[17] J. O. Kephart, D. M. Chess, The vision of autonomic computing, in: Computer, IEEE Computer Society Press, 2003, Vol. 36(1), pp. 41–50.

[18] Y. Zhang, A. Liu, W. Qu, Software architecture design of an autonomic system, in: Proc of 5th Australasian Workshop on Software and System Architectures, Melbourne, Australia, 2004, pp. 5–11.

[19] E. P. Mancini, M. Rak, U. Villano, Predictive autonomicity of web services in MAWeS framework, Journal of Computer Science, Science Publications, Vol. 2(6), 2006, pp. 513–520.

[20] E. P. Mancini, M. Rak, U. Villano, Autonomic service oriented architectures with MAWeS, To be published in Journal of Autonomic and Trusted Computing, American Scientific Publishers.

[21] E. Mancini, N. Mazzocca, M. Rak, U. Villano, Integrated tools for performance-oriented distributed software development., in: Proc. SERP'03 Conference, Las Vegas (NE), USA, 2003, Vol. 1, pp. 88–94.

[22] N. Mazzocca, M. Rak, U. Villano, The MetaPL approach to the performance analysis of distributed software systems., in: Proc. of 3rd International Workshop on Software and Performance (WOSP02), IEEE Press, 2002, pp. 142–149.

[23] R. Preziosi, M. Rak, L. Troiano, V. Casola, A reference model for security level evaluation: Policy and fuzzy techniques, Journal of Universal Computer Science, 2005, Vol. 11(1), pp. 150–174.

[24] OSSTMM, Open source security testing methodology manual.

[25] I. C. E. Levin T. E., S. E., Quality of Security Service: Adaptive Security - Handbook of Information Security, Wiley Computer Publishing John Wiley & Sons, 2006.

[26] V. Casola, A. Mazzeo, N. Mazzocca, V. Vittorini, A security metric for public key infrastructures, Journal of Computer Security, 2007, Vol. 15(2), pp. 197–229.

[27] N. Mazzocca, M. Rak, U. Villano, MetaPL a notation system for parallel program description and performance analysis parallel computing technologies, in: V. Malyshkin (Ed.), Parallel Computing Technologies, Lecture Notes in Computer Science, V Edition, Springer-Verlag, Berlin (DE), 2001, Vol. 2127, pp. 80–93.

Authors short biography

Valentina Casola is an assistant professor at the University of Naples Federico II. She received the Laurea degree in Electronic Engineering cum laude from the University of Naples in 2001 and she received her Ph.D in Electronic and Computer Engineering from the Second University of Naples in 2004. Her research activities are both theoretical and experimental and are focused on security methodologies to design and evaluate distributed and secure infrastructures.

Emilio P. Mancini is a research assistant at the Dept. of Engineering, at the University of Sannio, Benevento, Italy. In 2004 he received his Ph.D. in Computer Engineering. His current research interests are in the area of Autonomic and Parallel Computing. He has been working on some italian and european project on Grid Computing. Actually he studies the description and simulation of parallel systems.

Nicola Mazzocca is a full professor of High-Performance and Reliable Computing at the Computer and System Engineering Department of the University of Naples Federico II, Italy. He owns an MSc Degree in Electronic Engineering and a Ph.D. in Computer Engineering, both from the University of Naples Federico II. His research activities include methodologies and tools for design/-analysis of distributed systems; secure and real-time systems and dedicated parallel architectures.

Massimiliano Rak is an assistant professor at Second University of Naples. He get his degeree in Computer Science Engineering at the University of Naples Federico II in 1999. In November 2002 he got PhD in Electronical Engineering at Second University of Naples. His scientific activity is mainly focused on the analysis and design og High Performance System Architectures and on methodologies and techniques for Distributed Software development.

Umberto Villano is full professor at the University of Sannio at Benevento, Italy, where he is Director of the Department of Engineering. His major research interests concern performance prediction and analysis of parallel and

distributed computer architectures, tools and environments for parallel programming and distributed algorithms. He received the Laurea degree in Electronic Engineering cum laude from the University of Naples in 1983.