Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 310 (2015) 113–133

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Model-based Evaluation of Scalability and Security Tradeoffs: a Case Study on a Multi-Service Platform

Leonardo Montecchi[1], Nicola Nostro[1], Andrea Ceccarelli[1], Giuseppe Vella[2], Antonio Caruso[2], Andrea Bondavalli[1]

[1] *Università degli Studi di Firenze, Dipartimento di Matematica e Informatica*
Viale Morgagni 65, I-50134 Firenze, Italy
{leonardo.montecchi,nicola.nostro,andrea.ceccarelli,andrea.bondavalli}@unifi.it

[2] *Engineering Ingegneria Informatica S.p.A.*
Viale Reg. Siciliana 7275, Palermo, Italy
{giuseppe.vella,antonio.caruso}@eng.it

Abstract

Current ICT infrastructures are characterized by increasing requirements of reliability, security, performance, availability, adaptability. A relevant issue is represented by the scalability of the system with respect to the increasing number of users and applications, thus requiring a careful dimensioning of resources. Furthermore, new security issues to be faced arise from exposing applications and data to the Internet, thus requiring an attentive analysis of potential threats and the identification of stronger security mechanisms to be implemented, which may produce a negative impact on system performance and scalability properties. The paper presents a model-based evaluation of scalability and security tradeoffs of a multi-service web-based platform, by evaluating how the introduction of security mechanisms may lead to a degradation of performance properties. The evaluation focuses on the OPENNESS platform, a web-based platform providing different kind of services, to different categories of users. The evaluation aims at identifying the bottlenecks of the system, under different configurations, and assess the impact of security countermeasures which were identified by a thorough threat analysis activity previously carried out on the target system. The modeling activity has been carried out using the Stochastic Activity Networks (SANs) formalism, making full use of its characteristics of modularity and reusability. The analysis model is realized through the composition of a set of predefined template models, which facilitates the construction of the overall system model, and the evaluation of different configuration by composing them in different ways.

Keywords: Performance evaluation, scalability, web-services, security evaluation, security tradeoffs.

1 Introduction

The increased mobility of devices, pervasive connectivity, and multiple devices per user, produced a shift towards a "thin client" approach, where a large part of the required storage and computational power is demanded to servers [3]. The recent cloud computing paradigm extends this approach with an additional layer of abstraction, which separates physical resources (i.e., hardware) from logical resources (e.g., applications, storage, computational power) which are provided to users.

http://dx.doi.org/10.1016/j.entcs.2014.12.015

In the Software-as-a-Service (SaaS) paradigm, software applications are hosted on a central server, and provided to users on-demand. This is often accomplished by means of web-based interfaces, so that clients do not need any other application than a web browser. Social networks and online storage facilities are prominent examples of this paradigm. Due to its advantages in terms of resources, costs, and convenience, this kind of paradigm is often used also within organizations, to provide services to employees or internal users.

However, this approach also introduces several challenges. One of the main problems consists in the scalability of the system with respect to an increasing population of users and applications, so that resources need to be carefully dimensioned. Another challenge consists in the additional security threats originating from exposing applications and data to the Internet, thus requiring stronger security mechanisms to be implemented within the system. Security and performance are often in contrast with each other [17]: mechanisms to improve the security of the system often prescribe constraints on resource usage, or require additional computations to be performed in order to guarantee that security policies defined at design time are actually applied at runtime. Moreover, a large part of security mechanisms relies on cryptography algorithms, which are typically resource-intensive. Therefore, the addition of security mechanisms can produce a negative impact on system performance, which needs to be carefully quantified and evaluated.

In this paper we adopt a stochastic modeling approach in order to evaluate the scalability of a multi-service web-based platform, and the impact of introducing security mechanisms. The evaluation focuses on the OPENNESS platform, a web-based platform providing different services, to different categories of users. The evaluation aims at identifying the bottlenecks of the system, under different configurations, and assess the impact of security countermeasures.

The model is constructed using the Stochastic Activity Networks (SANs) formalism [16], which can be considered an extension of the well-known Stochastic Petri Nets (SPNs) [5] formalism. The key characteristic of our approach is in the modularity and reusability of the model: the analysis model is defined as a composition of a small set of "template" SAN models, which are then composed to form the overall system model. By composing them in different ways, the same templates can be used to evaluate different system configurations.

The rest of the paper is organized as follows. The OPENNESS framework is described in Section 2, while related work are discussed in Section 3. The stochastic model is described in Section 4, while evaluations and results are described in Section 5. Finally, conclusions are drawn in Section 6.

2 The OPENNESS Platform

The OPENNESS (OPEN Networked Enterprise Social Software suite) platform is the framework conceived within the research project VINCENTE [18], which aims at defining, realizing, and experimenting a technological platform for sustainable entrepreneurship. It optimizes the resources, enhances the sharing of knowledge,

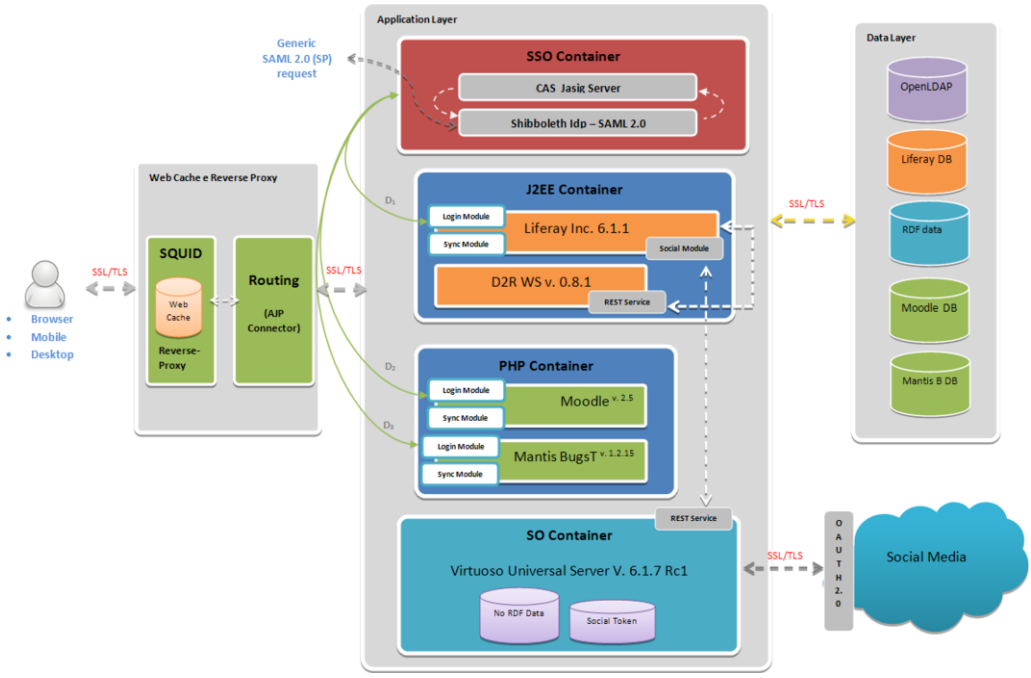


Fig. 1. Logical architecture of the OPENNESS platform.

and supports social discussion, while supporting modern technological standards. Moreover, such platform guarantees secure exchange of data between several services of heterogeneous frameworks. The design of the platform has thus been strongly characterized by the risk evaluation on the whole system, where the need to realize a secure system was of paramount importance.

The logical architecture of the OPENNESS platform adopts a classical three tier model, where every tier is independent from the others. The Three-Tier Architecture exposes a first layer, or *Web Layer*, where a web server manages requests coming from Clients, either Desktop, Web or Mobile, and delivers the content to the *Application Layer*, which in turn interacts with the *Data Layer*. Such layer is in charge to provide data to services in order to aggregate them and finally to satisfy Clients requests.

The Web Layer, which is built on top of the Apache Server, is also composed by a Web Cache, operating as a *Reverse-Proxy*, in order to reduce bandwidth and to improve response time. The Apache Web server communicates with the Application Layer components (e.g., Tomcat and PHP application servers), using AJP (Apache JServ Protocol), which is an optimized protocol for the J2EE and PHP containers. Communication between layers is based on the SSL (Secure Socket Layer) protocol.

The access to the platform is managed by an identity provider system, and a specific module is in charge to manage user accounts, which are centralized on an LDAP (Lightweight Directory Access Protocol) server.

The other components of the platform are integrated with the LDAP repository with a full synchronization of the user accounts; at the same time the modules for

the users management within the individual components are disabled, in order to make OPENNESS the only entry point for user registration and management.

Account centralization and OPENNESS specifications allowed the integration of two of the most efficient SSO Single-Sign On solutions. One of them is CAS (Central Authentication Service) , an open source framework implementing a SSO mechanism that provides a centralized authentication system on a single server. When a request is sent to an application, it is redirected to CAS, which deals with the authentication. The other Identity Provider implemented by OPENNESS, Shibboleth, sets up his SSO (Single Sign-on) logic on SAML protocol (Security Assertion Markup Language) allowing users to sign in to various systems using just one identity. CAS can be integrated with the Shibboleth federated SSO platform to serve as the authentication provider for Shibboleth.

As a multichannel platform OPENNESS will integrate mobile devices like smartphones and tablets either with iOS and Android. In order to allow a federated authentication a SSO mechanism has been implemented through a CAS service authentication using the REST protocol (Representational State Transfer) thus guaranteeing the persistence of the authentication even among different kinds of applications using the platform.

The OPENNESS architecture is summarized in Figure 1. The module in the left part of the figure shows the Web Layer, which includes the reverse-proxy. The central part of the figure highlights the core of the platform, which is represented by the Application Layer, which is in turn composed of different submodules. Based on the offered functionalities, it is possible to identify different submodules; in this paper we will consider the following main blocks:

- *SSO Container* (or *Authentication*). Comprises the SSO modules, i.e., CAS and Shibboleth, and is in charge of managing the authentication procedure.
- *J2EE Container*. Represents the web container in which all the services of the platform relying on Java 2 Enterprise Edition are installed. Within this modules it is possible to identify two main applications: the Liferay Portal and the D2R WS [1]. Liferay is a web portal framework written in Java based on a Service-Oriented Architecture (SOA). D2R WS is a framework that can be used to map different kinds of information to an ontology model, and retrieve them using SPARQL queries. Such service can be used both to expose the information on a certain database, and to perform SPARQL queries on external data providers.
- *PHP Container*. This block comprises all services that rely on PHP for their execution. In particular, we consider the Moodle and Mantis applications [2]. Moodle is a modular e-Learning platform. It allows teachers to organize lectures, and provides social features like forums, blogs, and chats to students. Mantis is a popular open source bug tracking system written in PHP.
- *SO Container*. This container provides access to the most popular social network platforms. This block is realized by means of the Virtuoso Universal Server [3], a

[1] Liferay: http://www.liferay.com/. D2R WS: http://www.d2qr.org/. Accessed: 2014-03-10.
[2] Moodle: http://www.moodle.org/. Mantis: http://www.mantisbt.org/. Accessed: 2014-03-10.

middleware which combines features of traditional relational databases with other data models, providing uniform access to them. Within OPENNESS, Virtuoso is used to manage the access to social media, through the Oauth2 protocol.

The right part of Figure 1 also depicts the Data Layer. Within such layer we identify a set of databases used to store application data (e.g., Moodle DB, Liferay DB) or user data (OpenLDAP), and the social media data provider.

3 Security, Scalability, Performance: Related Work

While precise definitions of security [2] and performance [8] properties exist in the literature, to the best of our knowledge, no unique definition of "scalability" has been established. With some variants, scalability is however usually intended as a metric that links the size of the system with the performance that it is able to reach [11,7]. Therefore, in our evaluation we will focus on performance metrics, and their sensitivity with respect to the size of the system.

In the literature, the impact of security mechanism on performance metrics, and thus on system scalability, is a well-known problem for different application domains. As an example, it is an important problem in the Wireless Sensor Networks (WSNs) domain, where the scarcity of resources and the large number of nodes raise the need to reduce the computational cost of security mechanisms and protocols. In [22] the overhead introduced by three different mechanism for secure communication in WSNs is evaluated by experimental means. In the worst case, among the kind of messages considered by authors, an overhead of up to 50% in message size has been measured, with a subsequent increase of more than 10% in transmission time.

Similar results have also been obtained in completely different application domains, e.g. e-commerce [6,1]. In particular, the authors of [1] perform an experimental evaluation of the impact of the TLS/SSL protocol on the performance of an application server in a business-to-business (B2B) setup. The evaluation performed by the authors of this work compares key performance indicators obtained with and without a secure connection based on the SSL protocol, at varying the number of clients that are concurrently using the system. Also in this case, the results obtained by the authors show that using a SSL channel introduces a performance degradation between 5% and 10%. In is interesting to note that, according to authors themselves, such results are optimistic and that with different workloads an even greater impact should be expected.

The impact of security on performance is even higher on web services which rely on XML for communication. Even though the use of XML for communication guarantees properties like interoperability and flexibility, its usage for implementing security mechanisms has a great impact on performance, mainly due to its excessive verbosity. Several works in the literature highlight the high performance cost due to the adoption of WS-Security, a standard based on XML to provide security mechanisms to web services. For example, results in [9] show an increase of network

[3] Virtuoso: http://virtuoso.openlinksw.com/. Accessed: 2014-03-10.

traffic up to 690% with the introduction of WS-Security if compared to web services without any security mechanism. Even worse, results in [17] highlight an increase of transmission time up to 2100% for response messages of small dimensions.

In addition to experimental approaches, model-based evaluation has also been used as a method for relating performance and security aspects. The adoption of a model-based approach allows analysts to i) obtain useful insights on the system from the early phases of system design, and ii) perform "what-if" analyses in order to estimate the impact deriving from architectural changes.

The authors of [21] analyze the tradeoffs existing in a key distribution centre, using the Markovian process algebra PEPA. Other existing approaches in the combined evaluation of performance and security are reviewed in [20]. In the same paper, the authors describe a general process using the example of choosing an appropriate key length for encryption.

For a successful deployment of a multi-service web-based system like OPENNESS, means to evaluate its scalability with respect to the expected workload are needed. Moreover, the evident impact of security mechanisms on the planned architecture needs to be assessed as well, in order to find the proper balance between security, performance, and flexibility of the system. In this paper we describe a model-based approach for performance and scalability analysis of OPENNESS.

4 Stochastic Model of the OPENNESS Platform

4.1 Modeling Approach

The modeling approach adopted in this paper is based on a compositional approach, where the overall system model is built by composing together a set of submodels, each addressing a specific aspect or component of the system.

In performing such decomposition, particular attention is devoted to the identification of the interfaces between the different submodels. Clearly defining the interfaces between submodels before their implementation improves the reusability, maintainability and modularity of the obtained submodels. Taking this concept to its highest level leads to a modeling paradigm that recalls object-oriented programming: the implementation of each submodel is independent from the other interacting submodels, and it only depends on the defined interfaces. Submodels obtained in this way are modular i.e., they can be easily replaced or refined as needed, provided that the input and output interfaces remain the same. This approach also eases the integration with external tools: a given submodel, implementing a specific function, may be replaced with an ad-hoc external tool, either directly or through a wrapper model. An example of such integration is described in [4], where adopting this approach allowed SAN submodel implementing a mobility model to be replaced with the output produced by a vehicular mobility simulator.

Another dual aspect that enhances the modularity of submodels is the identification of their parameters. In complex systems like OPENNESS, different components may have a similar behavior, only differing by some numerical parameters that are specific of a particular instance of the component, depending on its role in the

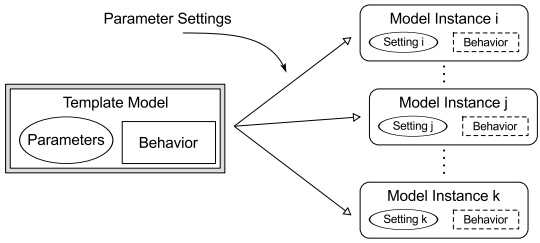


Fig. 2. Template models and parameterization.

system, or on the environment in which it is operating.

This process leads to the definition of "template submodels", which are composed of two parts: a part defining its behavior and a part defining its parameters (Figure 2). In the construction of the overall model these templates are then instantiated multiple times, with different parameters settings. This approach saves the modeler from manually creating (and maintaining) multiple models for components having a similar behavior, which is a very time-consuming and error-prone task. Also, any change in a template model is automatically propagated to all the instances of that template.

Instances of templates are then composed according to precisely defined rules, in order to obtain the overall model for the desired scenario. The ability to easily create different instances of the same model makes it also easier to evaluate the system under different conditions and different scenarios, which requires only adding or removing model instances or changing their parameters. When coupled with model-transformation techniques, such approach can greatly reduce the effort needed to create and assemble large stochastic models [12].

In this paper, the model is defined using Stochastic Activity Networks (SANs) [16], which provide useful features for the concrete application of such approach.

4.2 Assumptions and Metrics

The system architecture used as a reference for constructing the analysis model is the one described in Section 2. In order to precisely define some aspects of the system, some assumptions have been introduced on the behavior of users and on the deployment of services provided by the system.

The OPENNESS platform is used by users having different *profiles*, e.g., "teacher", "developer", "project manager". Each user of the system may perform a number of high-level *actions* on the platform, e.g., "manage an e-Learning course", "engage the communication and collaboration services". The kind of actions that are available to each user, and the pattern followed for their execution depend on the user profile.

Each action requires the use of one or more services of the platform (e.g., Moodle, Virtuoso, etc.). We also assume that the behavior of a generic user of the OPENNESS platform can be outlined as follows:

- The user u remains inactive for a mean time $T^u_{activation}$, according to an exponential distribution with rate $\lambda^u_{activation} = 1/T^u_{activation}$.

- After becoming active, a user executes, in a probabilistic way, a number of actions among those available to him, before returning into an inactive state.
- After the execution of an action has been completed, the beginning of the subsequent one is delayed by a physiological reaction time of the user (*think time*) T^u_{think}, in which the user processes the output of the platform and decides whether he will perform further actions or return into an inactive state. Such delay is distributed following an exponential distribution with rate $\lambda^u_{delay} = 1/T^u_{think}$.

The assumptions concerning the services provided by the platform are instead the following:

- Requests received by service s are served with an exponential rate $\lambda^s_{serve} = 1/T^s_{serve}$.
- With a certain probability p^s_{proxy}, requests for service s can be handled directly by the proxy.
- When a service request is served by the proxy, the service time is reduced by a factor γ^s_{proxy}, i.e., the average service time in this case is $\gamma^s_{proxy} \cdot T^s_{serve}$, with $\gamma^s_{proxy} < 1$.

The metrics of interest that will be evaluated by the analysis are mainly performance indicators, and are described in the following.

U_s**:** *Utilization of service s.* This metric provides an indication on the dimensioning of resources allocated to each service. It is evaluated as the probability that, at a certain instant of time, there are requests of service s waiting to be served.

T_s**:** *Mean waiting time for service s.* This metric provides an indication of quality of service received by system users. The metric is evaluated as the mean time that elapses from the instant to which a service request for s is issued, to the instant in which the request is served.

The scalability of the OPENNESS platform is highlighted through the evaluation of such metrics at varying parameters related to the size of the system, e.g., the number of users and the frequency with which they request services provided by the platform. The impact of security mechanisms on system performance, and thus on its scalability, can be evaluated by assuming an increase of service delays, based on experimental analyses available in literature (e.g., see Section 3).

4.3 System Model

The compositional modeling approach, the predefined assumptions, and the measures of interest to be evaluated, led us to identify three template models, each addressing a specific aspect or component of the system. The overall OPENNESS model is built by composing together such submodels. For the sake of brevity, in the following we provide a detailed description of two of the three template models, *UserBehavior* and *Service*; while the third one, *Action*, is only briefly described. After that, a description of the composed model and a specification of the previously defined metrics of interest are provided.

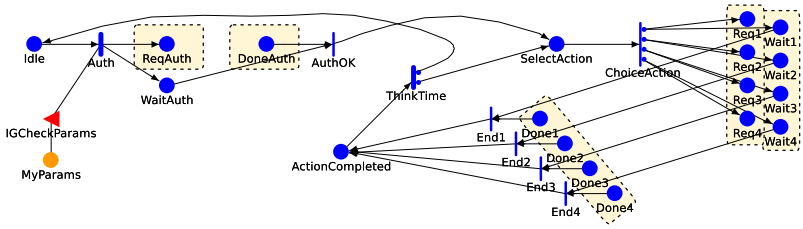


Fig. 3. `UserBehavior` template SAN model, representing a generic OPNENNESS user.

4.3.1 `UserBehavior` Template Model

According to the introduced assumptions, a user of the OPENNESS platform periodically becomes active, performs a number of actions, and then returns in an inactive state. The selection of an action by the user is modeled as a probabilistic choice. The actions that are available to the user, as well as the probability of being selected, depend on the user profile.

The SAN model for a generic user behavior u of the OPENNESS platform is depicted in Figure 3; in this one and in the following figures, interfaces to other template models are highlighted with a dashed yellow box. The user is initially in a waiting state, modeled as a token in place `Idle`. The SAN activity `Auth` represents the beginning of a new user session, and the subsequent request to the authentication module, i.e., the event for which a user becomes active. The firing time of such activity is exponentially distributed with rate $1/T_{activation}$.

When the `Auth` transition fires a token is added in place `ReqAuth` and in place `WaitAuth`. Place `ReqAuth` is an interface to the model of the authentication service (see Section 4.3.2), and it holds the total number of requests that are currently waiting to be served. Conversely, place `WaitAuth` is local to the user model, and is used to keep track that the user is waiting for the authentication to be performed.

Similarly, place `DoneAuth` is shared with the model of the authentication service, and contains a token when a new request has been fulfilled. When there is a token both in `DoneAuth` and `WaithAuth` places, the activity `AuthOK` is enabled and fires, representing the completion of the authentication procedure. In this case, a token is added in place `SelectAction`.

The `ChoiceAction` activity is then enabled and fires, representing a probabilistic choice between the actions available to the user. The activity has a case for each action X available to the user profile modeled, and each case has a different probability of being selected, pX. When the case X is probabilistically selected, a token is added in place `ReqX` and in place `WaitX`, which represents the beginning of the corresponding action performed by the user. Similarly as to what already described for the authentication service, each place `ReqX` is shared with the corresponding model of the user action (Section 4.3.3), while place `WaitX` is local to the user model. Once the action is completed, a token is added in place `DoneX` by the corresponding `Action` model, thus enabling the corresponding `EndX` activity. When the activity `EndX` fires a token is added in place `ActionCompleted`, representing the completion of the action performed by the user.

The reaction time of the user is modeled by the `ThinkTime` activity, which fires

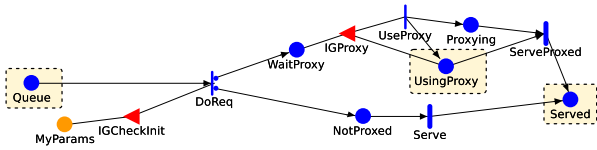


Fig. 4. `Service` template SAN model, representing a generic service of the OPENNESS platform.

with an exponentially distributed delay T_{think}. The `ThinkTime` activity has two cases, one corresponding to the choice to end the user session and return to an inactive state (selected with probability $pEnd$), and the other corresponding to the choice to perform a further action (selected with probability $1 - pEnd$).

4.3.2 `Service` Template Model

The SAN template model for a generic service s of the platform is depicted in Figure 4. Place `Queue` is an interface to the corresponding `ReqY` of each `Action` model. Thus, at any time this place contains a token for each pending service request. The `DoReq` activity models the beginning of the processing of a user request; with probability p^s_{proxy} the service request will be handled by the proxy (case 1), while with probability $1 - p_{proxy}$ the service is provided by the machine hosting the service itself (case 2). In the first case a token is added in place `WaitProxy`, while in the second case a token is added in place `NotProxed`.

Activity `Serve` represents the fulfillment of a service request; its firing time is exponentially distributed with rate λ^s_{serve}. When the activity fires, it removes a token from place `NotProxed` and adds a token in place `Served`.

Place `UsingProxy` is shared between all the instances of the `Service` template model, and it contains a token when the proxy is being used to satisfy a service request. When there is a token in place `WaitProxy` and the proxy resource is free (i.e., the place `UsingProxy` is empty), the activity `UseProxy` is enabled and fires, adding a token in places `Proxying` and `UsingProxy`, thus representing the utilization of the proxy. Activity `ServeProxed` represents the fulfillment of the service request by the proxy; its firing delay is thus distributed according an exponential distribution with rate $\gamma^s_{proxy} \cdot \lambda^s_{serve}$. Similarly to the `Serve` activity, when the activity `ServeProxed` fires it removes a token from places `Proxying` and `UsingProxy`, and adds a token in place `Served`, to represent the fulfillment of the service request.

4.3.3 `Action` Template Model

The execution of a user action generates a number of service requests to the platform. Once the action has started, the involvement of the different services is modeled as a probabilistic choice, in a similar way as in the `UserBehavior` template model. The platform services that are needed for the different actions depend on the kind of action itself. For example, the action "manage an e-Learning course" will perform a number of requests to the "PHP Container" and "Database" services. The SAN template model for a generic action is depicted in Figure 5. Due to its similarity to the `UserBehavior` template model, and to reasonable limits of space, we do not provide a description of such model, whose details can be found in [19].

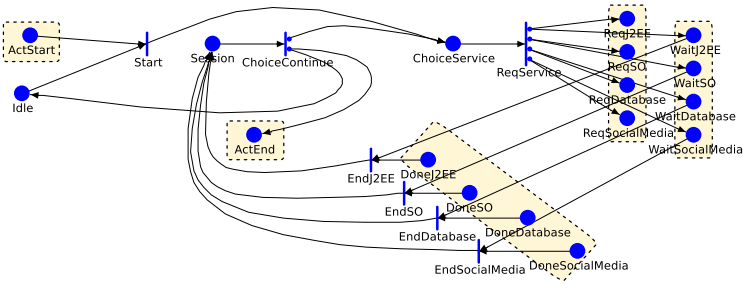


Fig. 5. `Action` template SAN model, representing a generic user action.

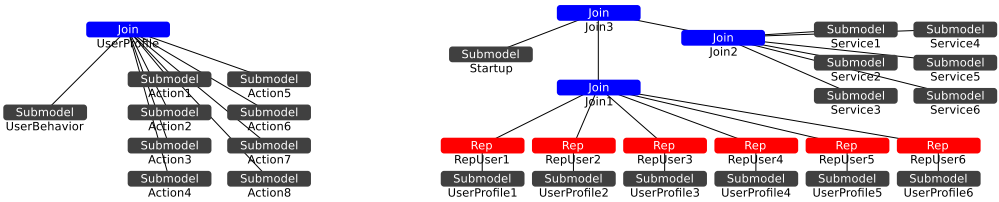


Fig. 6. Composed model of the overall OPENNESS platform. In a first stage the `UserBehavior` model is composed with instances of the `Action` model (a). In a second stage the different instances of the `UserBehavior-Action` composed model are replicated and then composed with instances of the `Service` template model (b).

4.3.4 Composing the Overall System Model

In order to obtain the overall model of the OPENNESS platform, the template SAN models described in Sections 4.3.1, 4.3.2, and 4.3.3 are instantiated several times with the appropriate parameters, and connected together – through the identified interfaces – using the ***Replicate/Join*** composition formalism [15].

In a first composition step models for the different user profiles are created; each of them is obtained by composing an instance of the `UserBehavior` template (Section 4.3.1) with a number of instances of the `Action` template (Section 4.3.3), based on the actions that are available to the corresponding user profile. In performing the composition, for each action X, interfaces `ReqX` and `EndX` of the `UserBehavior` model are connected with places `ActStart` and `ActEnd` of the `Action` model. This step is then repeated for each user profile that should be modeled. The number of `ActX` submodels depend on the number of actions that are available for the involved user profile. An example model corresponding to a user profile for which 8 actions are available is shown in Figure 6a. This kind of composed model will be referred to as the `UserProfile` composed model in the following.

The complete system model is then created in a second composition step, by replicating the models of user profiles, and composing them with instances of the `Service` template model. Multiple instances of the `UserProfile` composed SAN model are added, one for each of the user profiles that are supported by the system. Each instance of the `UserProfile` model is replicated through the *Replica* operator [15], in order to represent a number of identical users which operate on the platform according to the same profile. In the model of Figure 6b, each submodel "UserK" is an instance of the `UserProfile` template similar to the one depicted in Figure 6a,

while the red rectangle "Rep" represents the *Replica* composition operator.

Through the *Join* node "Join1" all the replicated models are composed together. In the composition, all the interface places `ReqX` and `EndX` corresponding to the same platform service are shared together. Then, such places are shared with the corresponding `Queue` and `Served` places of the different instances of the `Service` template model, which are shown in the upper right part of the figure. The instances of the `Service` template also share the `UsingProxy` place, as already described in Section 4.3.2. Finally, the `Startup` submodel is an helper model that is used to properly initialize the parameters of the different template instances.

The number of submodels of kind `UserProfile` and the number of instances of the `Service` template depend, respectively, from the number of different user profiles and the number of different services that should be modeled.

4.3.5 Specification of Metrics

The metrics of interest defined in Section 4.2 are specified on the stochastic model using reward structures. More in details:

U_s: *Utilization of service s.* To evaluate this metric, the reward function is defined as a function returning one unit of reward for each state in which service s is busy (i.e., `Mark(Queue)>0`), and zero otherwise. The mean reward that is obtained in a given instant of time t corresponds to the desired metric.

T_s: *Mean waiting time for service s.* This metric is evaluated as T^s_{tot}/N^s_{req}, where T^s_{tot} is the total amount of time that users spend waiting for service s, and N^s_{req} is the number of requests that have been issued for service s. In reward terms, the quantity T^s_{tot} can be obtained by defining a reward function that assigns to each state of the model the total number of users that are currently waiting for the service (i.e., `Mark(Queue)`); while N^s_{req} is obtained by defining a function that provides one unit of reward each time that service s is requested (i.e., each time that the case corresponding to service s is selected after the firing of a `ReqService` activity in an `Action` model).

In this paper, both the metrics are evaluated at steady-state. It should be noted that, using the same model, different metrics can be evaluated as well.

5 Evaluation and Results

In this section the model described in Section 4 is evaluated in different configurations, to consider the impact of some key parameters on the metrics of interest. Section 5.1 defines the scenario that will be used as a reference for the following evaluations, and introduces the default parameters assigned to the model. The scalability of the OPENNESS platform is analyzed in Section 5.2, while the impact of introducing some security countermeasures is evaluated in Section 5.3. All the obtained values have been computed by discrete-event simulation, with a confidence level of at least 99%, and a confidence half-interval of 1%.

ID	Profile Name	$T_{activation}$ (sec.)	T_{think} (sec.)	Users
1	Project Manager	1800	20	10
2	Developer	3600		50
3	Trainee	600		80
4	Teacher	3600		10
5	Public Citizen	600		100
6	Decision Maker	3600		20

Table 1
User profiles that have been considered in the reference scenario, and their default parameters.

1: Manage working groups	5: Manage an e-Learning course
2: Assign bugs and development activities	6: Generate an entrepreneurship idea
3: Manage bugs and development activities	7: Engage the communication and collaboration services
4: Access to an e-Learning course	8: Decide for the realization of an entrepreneurship idea

Table 2
Actions that are available to OPENNESS users in the considered scenario.

5.1 Reference Scenario and Default Parameters

The reference scenario considers 6 different user profiles, 8 kinds of actions, and 6 services. The user profiles and the corresponding parameters are reported in Table 1. Each of them has a different activation time ($T_{activation}$), but the same think delay (T_{think}), set to 20 seconds. The total number of users for each profile is also reported in the table.

The actions that can be performed in the platform are listed in Table 2. The association between them and the different user profiles is detailed in Table 3, where for each user profile are given the available actions and the corresponding selection probability pX (see Section 4.3.1). Table 3 also lists the value of the p^u_{end} parameter, i.e., the probability that user become inactive after having completed a specific action.

A user with profile *Project Manager* may manage the working groups (Action 1) and assign bugs and development activities to developers (Action 2). The latter of the two actions is much more frequent, since development activities will change more often than working groups. A *Developer* may only manage bugs and development activities (Action 3); moreover, he will typically perform a longer sequence of actions in the same user session. This aspect is modeled with a lower probability to terminate the session, i.e., the value of the p_{end} parameter is lower.

Similarly, a *Trainee* may only access to e-Learning courses (Action 4). A user with profile *Teacher* may access to courses, but he may also manage them (Action 5). A *Public Citizen* may be involved in the generation of an entrepreneurship idea (Action 6), and may use the platform to engage the communication and collaboration services (Action 7). Finally, users with profile *Decision Maker* use the platform to decide about the realization of ideas proposed by citizens (Action 8).

As previously described, each action involves the utilization of one or more services of the OPENNESS platform. The services considered in the reference scenario are listed in Table 4. In addition to the *Authentication* service, to which a request

ID	User Profile Name	Action Selection Probability p1	p2	p3	p4	p5	p6	p7	p8	pEnd
1	Project Manager	0.2	0.8	–	–	–	–	–	–	0.5
2	Developer	–	–	1.0	–	–	–	–	–	0.2
3	Trainee	–	–	–	1.0	–	–	–	–	0.2
4	Teacher	–	–	–	0.6	0.4	–	–	–	0.2
5	Public Citizen	–	–	–	–	–	0.8	0.2	–	0.4
6	Decision Maker	–	–	–	–	–	–	–	1.0	0.5

Table 3
Available actions for each user profile of the reference scenario, and corresponding selection probabilities.

ID	Service Name	Layer	T_{serve} (sec.)	p_{proxy}	γ_{proxy}
1	Authentication	*Web*	0.5	0.05	0.5
2	PHP Container	*Application*	0.1	0.4	0.4
3	J2EE Container	*Application*	0.5	0.3	0.7
4	SO Container	*Application*	1.0	0.2	0.7
5	Database	*Data*	0.1	0.1	0.9
6	Social Media	*Data*	2.0	0.5	0.7

Table 4
Services that are considered in the reference scenario, and their default parameters.

is issued initiating a new session, other five services are provided by the architecture, corresponding to the main blocks of the architecture described in Section 2. For each of these services the table lists the mean time to satisfy a user request of such services (T_{serve}), the probability that a service request is satisfied by the proxy (p_{proxy}), and the corresponding reduction factor for the time required to satisfy a service request (γ_{proxy}).

The mapping between actions and services is described in Table 5, which lists the services involved in each action, and their probability of being selected. Actions 1–5 involve the usage of the *PHP Container* (Service 2) and *Database* (Service 5) services only, since they are all based on PHP applications. More in details, actions 1–3 involve the usage of the "Mantis" application, while actions 4–5 involve the usage of the "Moodle" application. The ratio between the selection probability of the two services (i.e., the PHP Container and the Database) depends on the kind of action. For example, managing bugs and development activities (Action 3) is assumed to have a heavier impact on the database with respect to managing working groups (Action 1).

The "generation of an entrepreneurship idea" (Action 6) involves collecting and processing different information sources, in order to correctly describe the idea with consistent information. Accordingly, such action uses multiple services of the OPENNESS platform: the *J2EE Container*, the *SO Container*, the *Database*, as well as the *Social Media* infrastructure. The J2EE container is used for accessing the Liferay platform and the D2R server. The latter allows the user to access, through the database, to the main concepts that will be referenced by the idea; the SO container provides the access to Virtuoso, which in turn supports the collection of information from social media.

Action		Service Selection Probability						
ID	Name	p1	p2	p3	p4	p5	p6	pEnd
1	Manage working groups	*	0.7	–	–	0.3	–	0.5
2	Assign bugs and development activities	*	0.6	–	–	0.4	–	0.2
3	Manage bugs and development activities	*	0.4	–	–	0.6	–	0.2
4	Access to an e-Learning course	*	0.7	–	–	0.3	–	0.2
5	Manage an e-Learning course	*	0.5	–	–	0.5	–	0.5
6	Generate an idea	*	–	0.4	0.3	0.1	0.2	0.3
7	Engage the communication and collaboration services	*	–	0.3	0.2	0.1	0.4	0.6
8	Decide for the realization of an idea	*	–	0.6	0.2	0.2	–	0.7

* The "Authentication" service (Service 1) is not requested by users within as specific action; instead, it is requested only once at the beginning of each user session.

Table 5
Services that are involved in the execution of each action, and their probability of being selected.

The same set of services is exercised also when using the communication and collaboration services (Action 7). In this case however, due to the "social" nature of such an action, the selection probability of the *Social Media* service is higher, while the usage probability of the J2EE and SO containers is reduced. Finally, deciding for the realization of an idea (Action 8), requires the same services as for the generation of the idea, with the exception of the social media plugin, which is not used in the decision process.

It should be noted that the introduced scenario allows to evaluate the system at varying some key parameters. The modeling and evaluation approach is independent of the actual values of system parameters (including available user profiles, actions, and services), and can be used to evaluate the performance behavior of the OPENNESS platform in a wide range of configurations and scenarios.

5.2 System Scalability

One of the most critical parameters in evaluating the scalability of a multi-service platform is the number of users that have access to the system. Figure 7a depicts the utilization percentage (U_s on the y-axis) of the different services provided by the platform at varying the total number of users accessing the system. In particular, the system has been evaluated with a total number of users multiple with respect to the reference configuration. The lower x-axis in the figure is labeled with the adopted multiplier (e.g., $2n$, $3n$), while the upper x-axis is labeled with the total number of users in the system. The reference configuration is highlighted with a vertical line in the figure.

As shown in the figure, the system reaches its maximum workload, due to a utilization factor of the proxy approaching 100%, when serving a population of around 1000 users. In such conditions, the services *SO Container* and *Social Media* reach a utilization factor near to 90% as well. Above a certain threshold (about 800 users) the utilization of some services appears to decrease. Such behavior is caused by the overload condition of the system. In fact, active users perform actions sequentially; the overload of some services (e.g., *SO Container*) causes some actions to experience a high execution time, with the consequence of having less actions to

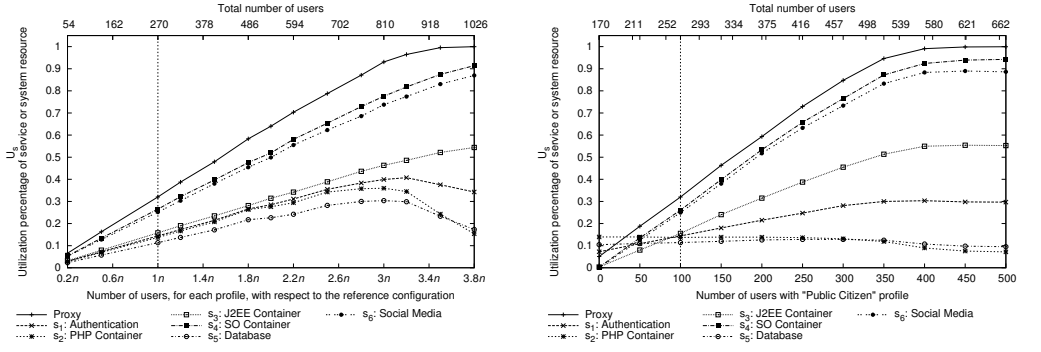
be initiated by users per unit of time.

Although the previous figure provides a good indication of system scalability with respect to the number of users, such results assume that the ratio between the different user profiles remain constant during the growth of the user-base. In general, this is however not the case. Actually, some kinds of users are subject to a higher increasing rate with respect to the others. In particular, the number of users with *Public Citizen* profile can experience a very rapid growth, depending on the popularity and spread of the platform.

Figure 7b depicts the utilization percentage of the different services (on the y-axis), at varying the number of users with profile *Public Citizen* (on the lower x-axis). Also in this case, the upper x-axis is marked with the total number of users in the system. When comparing this picture with the previous one it becomes evident that the ratio between the different user profiles has a great impact on the scalability of the system. In this case the saturation of the system resources is reached with just 600 users, i.e., 40% less then obtained when assuming a proportional increase of users with all the different profiles. Also in this case the bottleneck appears to be the proxy, followed by the *SO Container* and *Social Media* services, which reach a utilization factor of about 90%.

By analyzing the quality of service perceived by users of the OPENNESS platform, in terms of the mean time to have a service request fulfilled, T_s, we observe that the practical limit to system scalability is way lower than 600 users (Figure 8a). How it is shown in the figure, even with a total number of users higher than 400, users experience a great increase of the mean waiting time for four of the six services. With a population of more than 500 users, average waiting times of 10 seconds and higher are experienced, which are clearly unacceptable from a user's point of view. Such a great increase is probably due to the saturation of the proxy, which has been shown to be the performance bottleneck in the considered scenario.

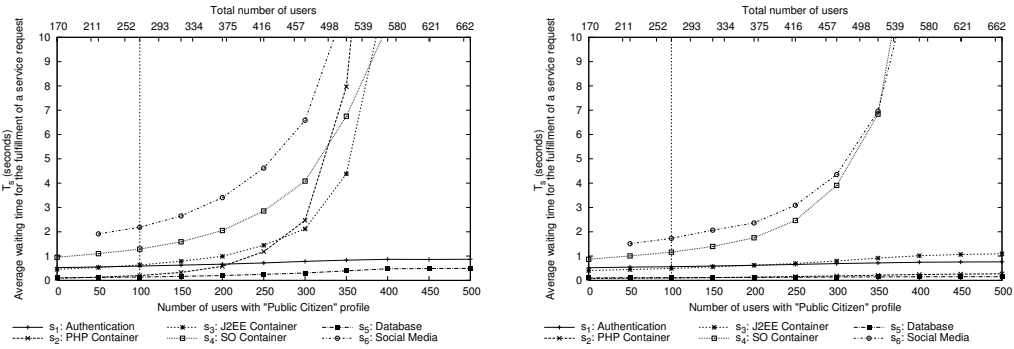
Evaluations reported in Figure 8b show the effect of an increase of proxy performances. In particular, we assume to double the proxy processing power, thus halving its processing delay. Such an increase in proxy performances is modeled by halving the γ_{proxy} parameter for each service. As shown by comparing Figure 8a



(a) With respect to the global number of users

(b) With respect to users with profile *Public Citizen*

Fig. 7. Impact on services utilization of an increasing number of users interacting with the platform.



(a) Default architecture

(b) Improved proxy

Fig. 8. Impact on quality of service perceived by users of the OPENNESS platform of the total number of users with profile *Public Citizen*.

and Figure 8b in the reference configuration (vertical line in both the figures), the resulting system-level improvement is minimum: the greater impact is on the *Social Media* service, for which we however have a reduction of the mean waiting time of about 25%. For the *Authentication* and *Database* services the improvement is practically negligible.

Still, such modification has a great impact on the system scalability with respect to the *Public Citizen* user profile (see Figure 8b). The practical limit to the number of users in the system is greatly improved in this case. Actually, it is possible to obtain acceptable waiting times for all the services until about 500 total users. With an improved proxy, the bottlenecks are now the *SO Container* and *Social Media* services, which provide unacceptable mean waiting times for a population of users greater than 500 (330 of which having *Public Citizen* profile). This is partially due to the higher service time with respect to the other services (see Table 4), but also to the fact that they are heavily used exactly by the *Public Citizen* profile.

5.3 Impact of Security Mechanisms

In this section the OPENNESS platform is evaluated considering the effect of introducing some security mechanisms. We assume that the introduction of such mechanisms produces an adverse effect on system performance, as largely documented by experimental work in the literature (see Section 3). The obtained results are then compared with the ones described in the previous section.

Based on a threat analysis previously carried out on the reference system, a number of countermeasures were identified to contrast the detected threats; we refer to them through the identifier C*n*. Among the 13 countermeasures that were identified [19], we focus on two of them that could have a significant impact on system performance:

C007 "Validate inputs provided by users with blacklist or whitelist approaches".

C008 "Implement an Intrusion Prevention System".

The introduction of validation mechanisms for data provided by users (C007) requires additional checks in the application, with a subsequent increase, although

Countermeasure ID	Service ID 1	2	3	4	5	6
C007	–	5%	5%	5%	–	–
C008	10%	10%	10%	10%	10%	10%

Table 6
Increase in mean service times due to the introduction of security mechanisms C007 and C008.

limited, of the time required to satisfy a service request. Such countermeasure does not impact on the whole OPENNESS platform, but only on components that run application code, i.e., the *PHP Container*, *J2EE Container*, and *SO Container*. Concerning the introduction of intrusion prevention mechanisms (C008) we assume that they are introduced in several points of the architecture, thus having an impact on all the service provided by the platform. In numerical terms, we assume an increase of mean service times of 5% with the introduction of C007, and 10% with the introduction of C008 (Table 6).

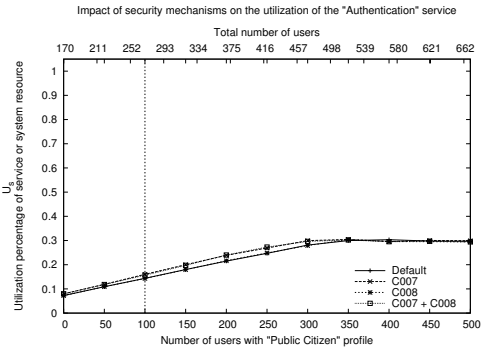
The following evaluation focuses on highlighting the differences between four different system configurations:

(i) Default configuration;
(ii) Implementation of C007;
(iii) Implementation of C008;
(iv) Implementation of C007 and C008.

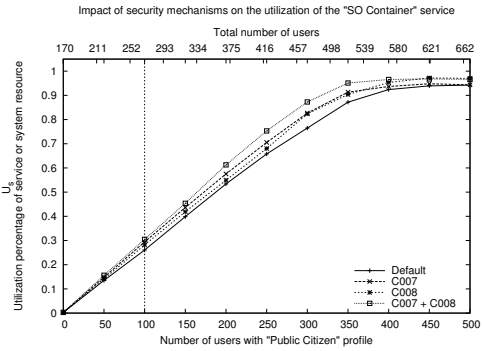
For both simplicity and space constraints, we focus on a subset of the available services: the *Authentication* and the *SO Container*. This choice is due to the following considerations on results obtained in Section 5.2: i) excluding the proxy, the *SO Container* service resulted the less scalable service, together with the *Social Media* service; ii) the *Authentication* service, together with the *Database*, resulted instead the one less suffering from scalability problems. Focusing on these two services provides then a good understanding, for the purpose of this paper, of the behavior of the overall platform. In this case we consider an increase in users with the *Public Citizen* profile only, as this configuration was deemed to be more critical with respect to system scalability.

Concerning the utilization of the *Authentication* service (Figure 9a), the impact of security countermeasures results to be minimum. Even for a large number of users, the utilization of the service does not go beyond 30%, without a noticeable increase with respect to the default configuration.

The introduced security mechanisms have instead a considerable impact on the utilization of the *SO Container* service (Figure 9b). In this case, even the introduction of the sole C007 mechanism produces a noticeable increase of the load on the service. In particular, for a configuration with 300 users with profile *Public Citizen* (470 users in total), the increase in service time is around 10% just with the introduction of C007 *or* C008, and around 20% if introducing both. In particular, the greatest increase occurs with a mid-large number of users (between 400 and 500 users in total), while the increase is slightly lower with a larger user population. We also note that the countermeasure C007 is the one causing the greater part of the load increase, causing an increase higher than 10%, while C008 is nearly irrelevant.

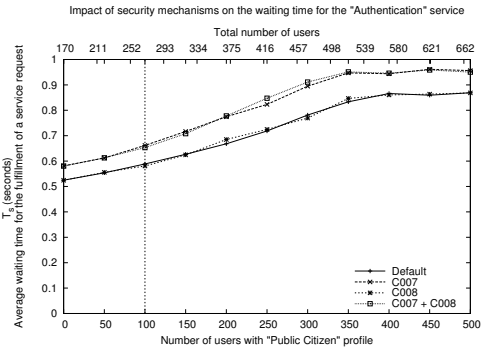


(a) Authentication

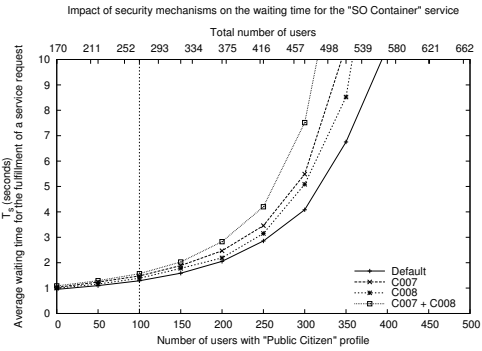


(b) SO Container

Fig. 9. Impact of security mechanisms C007 and C008 on the utilization factor.



(a) Authentication



(b) SO Container

Fig. 10. Impact of security mechanisms C007 and C008 on the mean waiting time.

The impact of the introduction of the two security countermeasures is greatly noticeable in the mean waiting times of users. As shown in Figure 10a, if considering the mean waiting times experienced by users, the two countermeasures have a considerable impact even on the *Authentication* service. It should be noted that such drawback was not noticeable if analyzing just infrastructure-oriented metrics like in Figure 9a. However, the increase in the waiting time seems to be constant, i.e., it does not get worse with an even higher number of users, and it is always around 10%. This is probably due to the limited utilization of the service, which prevent the stacking of excessive delays in the fulfillment of user requests.

Finally, Figure 10b depicts the impact of the considered security mechanisms on the mean waiting time for the *SO Container* service. The impact of both mechanisms is in this case huge, especially when the number of users in the system increase. With a number of users barely double with respect to the default configuration (200 *Public Citizen* users, 370 in total) introducing both the countermeasures increases the mean waiting time of about 30%.

If the number of users continues to increase the deterioration is sharp: with 300 *Public Citizen* users (470 in total) the introduction of both the countermeasures produces a mean waiting time which is nearly double with respect to the one ob-

tained in the default configuration. Moreover, even the introduction of only one of the two mechanisms increases the waiting time for the *SO Container* service of nearly 30%. This behavior is mainly due to two factors: i) the high mean serving time of the *SO Container* service, but also ii) the high utilization of this service by users with *Public Citizen* profile. Finally, it is interesting to note that in this case – despite their different impact on service times (see Table 6) – the two countermeasures have a similar impact on the overall system performance figures, with a slight higher impact due to C007.

6 Concluding Remarks

In this paper we applied stochastic modeling to the evaluation of performance and scalability metrics of the multi-service web-based OPENNESS platform. The approach presented in this paper highlights one emerging application of stochastic modeling, i.e., the evaluation of the impact of security countermeasures on the performance of a service-based architecture. The framework defined in this paper also highlights how achieving modularity can improve the reusability and maintainability of models based on Stochastic Petri Nets.

In the last ten years, security and model-based evaluation have been more and more coupled together [13], and formalism for *quantitative* analysis of security properties have started to emerge (e.g., the ADVISE formalism [10,14]). In this paper we addressed a complementary aspect in "quantifying" security, i.e., quantifying its impact on system performance. While quantifying security and quantifying its performance impact are usually addressed in isolation, strong relationship exist between these two aspects. As a first step towards a unified performance/security evaluation approach, future work aims at extending the framework presented in this paper with attack models, and apply quantitative security analysis in order to be able to identify the most convenient architectural solutions, *quantitatively* balancing security, performance, and costs requirements.

Acknowledgement

This work has been partially supported by the TENACE PRIN Project (n. 20103P34XC) funded by the Italian Ministry of Education, University and Research, and the PON Ricerca e Competitività 2007–2013 VINCENTE [18] project.

References

[1] Apostolopoulos, G., V. Peris, P. Pradhan and D. Saha, *Securing electronic commerce: reducing the SSL overhead*, IEEE Network **14** (2000), pp. 8–16.

[2] Avižienis, A., J.-C. Laprie, B. Randel and C. Landwehr, *Basic concepts and taxonomy of dependable and secure computing*, IEEE Transactions on Dependable and Secure Computing **1** (2004), pp. 11–33.

[3] Bondavalli, A., A. Ceccarelli and P. Lollini, *Architecting and validating dependable systems: Experiences and visions*, in: *Architecting Dependable Systems VII*, Lecture Notes in Computer Science **6420**, Springer, 2010 pp. 297–321.

[4] Bondavalli, A., P. Lollini and L. Montecchi, *Qos perceived by users of ubiquitous umts: Compositional models and thorough analysis*, Journal of Software **4** (2009), pp. 675–685.

[5] Ciardo, G., R. German and C. Lindemann, *A characterization of the stochastic process underlying a stochastic petri net*, Software Engineering, IEEE Transactions on **20** (1994), pp. 506–515.

[6] García, D. F., R. García, J. Entrialgo, J. García and M. García, *Evaluation of the effect of SSL overhead in the performance of e-business servers operating in B2B scenarios*, Computer Communications **30** (2007), pp. 3063–3074.

[7] Hill, M. D., *What is scalability?*, ACM SIGARCH Computer Architecture News **18** (1990), pp. 18–21.

[8] IEEE Standards Boards, *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12-1990 (1990).

[9] Juric, M. B., I. Rozman, B. Brumen, M. Colnaric and M. Hericko, *Comparison of performance of Web services, WS-Security, RMI, and RMISSL*, Journal of Systems and Software **79** (2006), pp. 689–700.

[10] LeMay, E., M. Ford, K. Keefe, W. Sanders and C. Muehrcke, *Model-based security metrics using adversary view security evaluation (advise)*, in: *Eighth International Conference on Quantitative Evaluation of Systems (QEST 2011)*, 2011, pp. 191 –200.

[11] Luke, E., *Defining and measuring scalability*, in: *Proceedings of the Scalable Parallel Libraries Conference*, 1993, pp. 183–186.

[12] Montecchi, L., P. Lollini and A. Bondavalli, *A DSL-Supported Workflow for the Automated Assembly of Large Performability Models*, in: *Proceedings of the 10th European Dependable Computing Conference (EDCC'14)*, Newcastle upon Tyne, UK, 2014.

[13] Nicol, D. M., W. H. Sanders and K. S. Trivedi, *Model-based evaluation: from dependability to security*, IEEE Transactions on Dependable and Secure Computing **1** (2004), pp. 48–65.

[14] Nostro, N., A. Ceccarelli, A. Bondavalli and F. Brancati, *A methodology and supporting techniques for the quantitative assessment of insider threats*, in: *Proceedings of the 2nd International Workshop on Dependability Issues in Cloud Computing*, ACM, 2013, p. 3.

[15] Sanders, W. and J. Meyer, *Reduced base model construction methods for stochastic activity networks*, IEEE Journal on Selected Areas in Communications **9** (1991), pp. 25–36.

[16] Sanders, W. H. and J. F. Meyer, *Stochastic activity networks: formal definitions and concepts*, in: *Lectures on formal methods and performance analysis*, Springer-Verlag New York, Inc., New York, NY, USA, 2002 pp. 315–343.

[17] Sosnoski, D., *Java web services: The high cost of (WS-)Security*, IBM developerWorks (July 2009).

[18] *VINCENTE: A Virtual collective INtelligenCe ENvironment to develop sustainable Technology Entrepreneurship ecosystems*, PON Ricerca e Competitività 2007–2013, Decreto Direttoriale prot. N. 647 del 08/10/2012.

[19] VINCENTE, *R4.1 "Valutazione di scalabilità in OPENNESS e relativo impatto su sicurezza"*, Technical report (2014).

[20] Wolter, K. and P. Reinecke, *Performance and security tradeoff*, in: A. Aldini, M. Bernardo, A. Pierro and H. Wiklicky, editors, *Formal Methods for Quantitative Aspects of Programming Languages*, Lecture Notes in Computer Science **6154**, Springer Berlin Heidelberg, 2010 pp. 135–167.

[21] Zhao, Y. and N. Thomas, *Efficient solutions of a PEPA model of a key distribution centre*, Performance Evaluation **67** (2010), pp. 740–756, special Issue on Software and Performance.

[22] Zia, T., A. Zomaya and N. Ababneh, *Evaluation of overheads in security mechanisms in wireless sensor networks*, in: *International Conference on Sensor Technologies and Applications (SensorComm 2007)*, 2007, pp. 181–185.