Fault-Tolerant Techniques and Security Mechanisms for Model-based Performance Prediction of Critical Systems

Ricardo J. Rodríguez
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza,
María de Luna 1, 50018 - Zaragoza, Spain
rjrodriguez@unizar.es

Catia Trubiani
Dipartimento di Informatica
Università dell'Aquila,
Via Vetoio, Coppito 67010 - L'Aquila, Italy
catia.trubiani@univaq.it

José Merseguer
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza,
María de Luna 1, 50018 - Zaragoza, Spain
jmerse@unizar.es

ABSTRACT

Security attacks aim to system vulnerabilities that may lead to operational failures. In order to react to attacks software designers use to introduce Fault-Tolerant Techniques (FTTs), such as recovery procedures, and/or Security Mechanisms (SMs), such as encryption of data. FTTs and SMs inevitably consume system resources, hence they influence the system performance, even affecting its full operability.

The goal of this paper is to provide a model-based methodology able to quantitatively estimate the performance degradation due to the introduction of FTTs and/or SMs aimed at protecting critical systems. Such a methodology is able to inform software designers about the performance degradation the system may incur, thus supporting them to find appropriate security strategies while meeting performance requirements. This approach has been applied to a case study in the E-commerce domain, whose experimental results demonstrate its effectiveness.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*performance measures*; C.4 [**Performance of Systems**]: *fault tolerance, modeling techniques*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Design, Performance, Security

Keywords

Critical Systems, Fault-Tolerant Techniques, Security Mechanisms, Model-based Performance Prediction.

1. INTRODUCTION

Communication networks are globally used to perform many transactions: electronic purchases, bank transfers or even stock exchanges can be accomplished with a computer connected to a network. This new concept of electronic market allows to perform almost everything remotely, so saving a lot of time to the users.

The main drawback in this domain is that some bad human behaviours may occur: spam or junk mails, viruses, trojan horses or other attacks are commonly suffered. For example, with the Denial-of-Service (DoS) attack [12] multiple requests are sent to a server with the intention of consuming its resources and, in last term, bringing the server down. These harmful actions clearly have an impact on the functionality of servers that might not be able to attend all incoming requests, and finally might bring down their services for saturation.

Relevant efforts of software designers are devoted on devising the security strategies suitable to protect information and computational systems against not authorised accesses. In fact, when designing critical systems it is fundamental to study the attacks that may occur and plan how to react from them. The occurrence of attacks in software systems leads software designers to introduce different Fault-Tolerant Techniques (FTTs), such as recovery procedures, and/or Security Mechanisms (SMs), such as encryption of data, in order to react to intrusions.

Despite these efforts, it is necessary to consider the costs that have to be incurred to guarantee a certain security level in critical systems. In fact, the security costs can be very relevant and may span along different dimensions, such as budgeting, performance and reliability [18, 19]. In this paper we focus on the security costs related to the system performance.

FTTs and SMs inevitably consume system resources hence they influence the performance, even affecting its full operability. Therefore, the necessity of balancing security and performance in these systems becomes clear: security strategies must assure that the system guarantees a minimal level of functionality.

This paper works towards this goal. We define a model-based methodology able to quantitatively estimate the system performance while introducing some FTTs and/or SMs aimed at protecting critical systems. Such a methodology is able to inform software designers about the performance degradation the system may incur, thus supporting them to find appropriate security strategies while minimising performance penalties.

To this end, we make use of a library of models that represent a subset of FTTs and SMs ready to be composed. Once a system model is built, in order to conduct a joint analysis of security and performance with our approach it is necessary: (i) to specify the appropriate security annotations (e.g. the confidentiality of some data), and (ii) to annotate the model with performance related data (e.g. the system operational profile). Thereafter, such an annotated model can be automatically transformed into a performance model whose solution quantifies the prediction of performance properties for the system under design.

The starting point of this work can be found in [7, 8, 22], where we introduced a preliminary set of models aimed at representing the most common security strategies: models for FTTs have been introduced in [22], whereas models for SMs have been presented in [8]. This paper jointly considers FTTs and SMs (namely, we consider a subset of FTTs and SMs, as summarised in Section 2) with the aim to enlarge the set of alternatives in the hands of software designers while making critical systems more secure. The final goal is to allow the addition of security strategies to a given system model thus to enable a model-based performance analysis.

The setting where our approach works is Unified Modelling Language (UML) [20] for software modelling and Generalized Stochastic Petri Nets (GSPNs) [1] for performance analysis.

UML models are aimed at representing the architecture of critical software systems. Such models can be extended for specific purposes through a technique called profiling [17, 26]. A UML profile defines a set of stereotypes and tagged-values which are used to extend its semantic. In this paper we use two profiles: (i) the Modelling and Analysis of Real-Time and Embedded Systems (MARTE) profile [21] for the specification of performance properties that enable the performance analysis; (ii) and the Security Analysis and Modelling (SecAM) profile [23] for the specification of security properties.

UML annotated models are transformed into GSPN models, i.e., formal models representing the system for performance analysis purposes. This choice has been driven by two main factors: (i) GSPNs provide a formal notation which avoids any source of ambiguity while representing the stochastic behaviour of systems; (ii) GSPNs have a clear graphical notation and several tools have been developed for analysis. The transformation from UML to GSPN can been carried out using well-established tools, such as ArgoSPE [13], ArgoPN [10] or ArgoPerformance [11].

The remainder of the paper is organised as follows. Section 2 provides some background on the SMs and FTTs we consider while designing critical systems. Section 3 presents our approach and the types of analyses it can support. Section 4 reports the case study we used to validate the approach, and experimental results are discussed. Section 5 compares our approach with respect to the existing literature. Finally, Section 6 provides concluding remarks and future research directions.

2. BACKGROUND

This section provides some background on the security mechanisms (see Section 2.1) and the fault-tolerant techniques (see Section 2.2) we consider in this paper while designing critical systems.

2.1 Security Mechanisms

The Security Mechanisms we consider in this paper are: *Encryption*, which refers to the usage of mathematical algorithms to transform data into a form that is unreadable without knowledge of a secret (e.g. a key); *Decryption*, which is the inverse operation of Encryption and makes the encrypted information readable again; the *Digital Signature*, which is a mathematical scheme for demonstrating the authenticity of a digital message or document through its *Generation* and *Verification*.

Some preliminary operations, such as the generation of public and secret keys and the process of obtaining a certificate from a certification authority, are executed once by all software entities involved in the security annotations. The generation of public and private keys involves a software component that sets the key type and length thus to generate the public and the private keys. The process of obtaining a certificate from a certification authority involves a software component that sends its information and its public key; the certification authority checks the credentials and, if trusted, generates the certificate and sends it back to the software component.

Encryption. The sender of the message decides the type of algorithm to use and the key length. The encryption can be of two different types: (i) asymmetric encryption (i.e., by public key); (ii) symmetric encryption (i.e., by a shared secret key). For asymmetric encryption the sender sets the padding scheme it requires and verifies the receiver's certificate if it is not already known. Finally, the encryption algorithm is executed on the message with the public key of the receiver. For symmetric encryption the sender sets the algorithm mode, performs a key-exchange protocol if a shared key is not already exchanged, and requires the exchange of certificates. Finally, the encryption algorithm is executed on the message with a session key obtained combining the keys generated by the sender and the receiver.

Decryption. After receiving the encrypted message, the algorithm type and the key length are extracted, and the decryption algorithm is executed to obtain the plain text.

Digital Signature Generation. The hash function algorithm must be specified, and the digest is generated. The encryption algorithm is applied on the digest by using the software component private key.

Digital Signature Verification. A message and the digital signature are received as inputs. Two operations are performed: the first one is to calculate the digest; the second one is the actual execution of the encryption algorithm applied on the input digital signature producing a forecast of the real signature. The last computation involves the verification of the digital signature which compares the forecast digital signature with the received one, in order to confirm the verification.

For sake of space the models of the aforementioned security mechanisms are not reported, for further details please refer to [8].

2.2 Fault-Tolerant Techniques

Fault-Tolerant Techniques are added to mitigate the consequences of faults that when exploited may lead to failures, i.e. to assure that critical systems remain fully operative. FTTs are classified in: fault detection, fault recovery, fault handling and fault masking [3]. Depending on when FTTs are applied, they are: (i) either proactive techniques, when they mitigate the effect of the failures as a way of prevention, i.e., without any previous proof of having failures; (ii) either reactive techniques when they are applied once some fault is detected; (iii) or proactive-reactive techniques [27].

In [22], an initial FTT model library was introduced. Such a library contains a model of a proactive-reactive FTT inspired in the one given in [27]. In this paper, we extend such a FTT library with two new FTTs techniques. The FTTs we consider in this paper are: *Switch Over Failing* and *Ping And Restore*. Both techniques are fault detection and recovery reactive FTTs aimed at adding redundancy capacity to the system, but in a different way.

Switch Over Failing. It provides an Intrusion Detection System (IDS) which is in charge of analysing incoming requests, and filtering legal ones to be correctly processed by the system. Besides, the IDS defines a threshold that allows to establish an attack limit. When such a limit is exceeded, the IDS brings down the machine

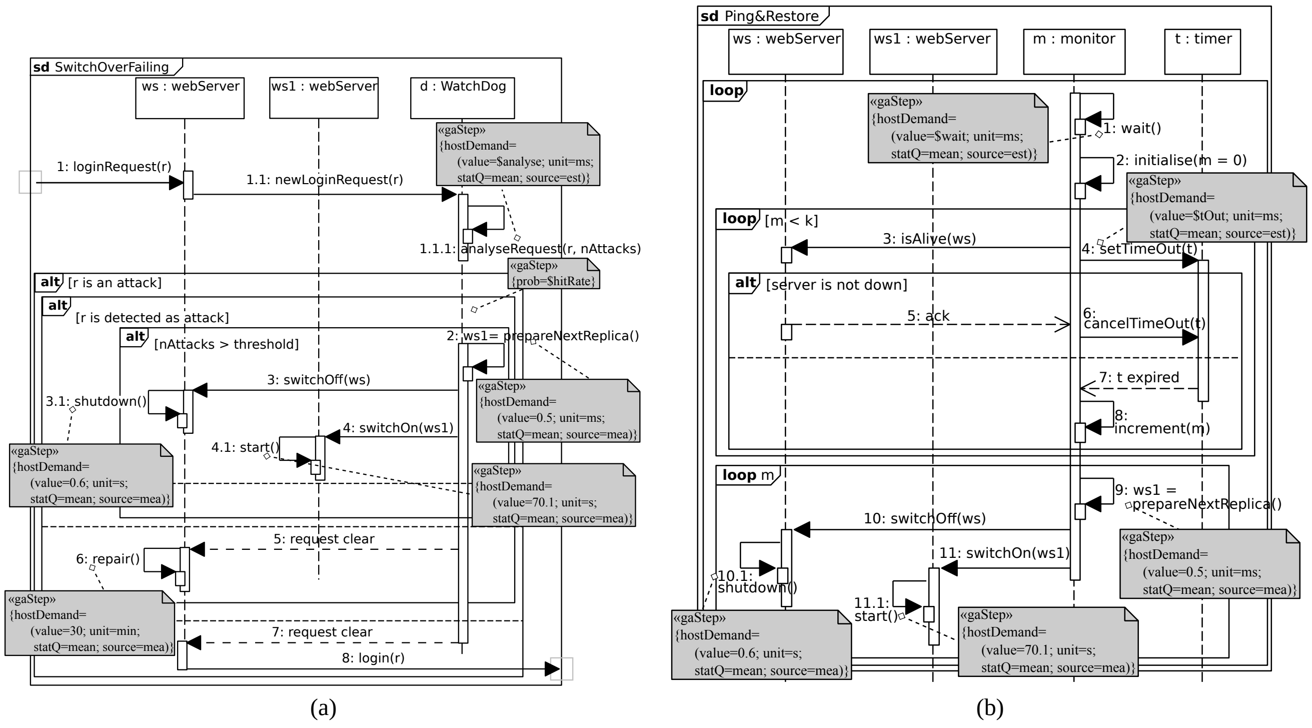

sd SwitchOverFailing
ws : webServer
ws1 : webServer
d : WatchDog
1: loginRequest(r)
1.1: newLoginRequest(r)
1.1.1: analyseRequest(r, nAttacks)
«gaStep» {hostDemand= (value=$analyse; unit=ms; statQ=mean; source=est)}
«gaStep» {prob=$hitRate}
alt [r is an attack]
alt [r is detected as attack]
alt [nAttacks > threshold]
2: ws1= prepareNextReplica()
3: switchOff(ws)
3.1: shutdown()
4: switchOn(ws1)
4.1: start()
«gaStep» {hostDemand= (value=0.5; unit=ms; statQ=mean; source=mea)}
«gaStep» {hostDemand= (value=0.6; unit=s; statQ=mean; source=mea)}
«gaStep» {hostDemand= (value=70.1; unit=s; statQ=mean; source=mea)}
5: request clear
6: repair()
«gaStep» {hostDemand= (value=30; unit=min; statQ=mean; source=mea)}
7: request clear
8: login(r)
(a)
sd Ping&Restore
ws : webServer
ws1 : webServer
m : monitor
t : timer
loop
«gaStep» {hostDemand= (value=$wait; unit=ms; statQ=mean; source=est)}
1: wait()
2: initialise(m = 0)
«gaStep» {hostDemand= (value=$tOut; unit=ms; statQ=mean; source=est)}
loop [m < k]
3: isAlive(ws)
4: setTimeOut(t)
alt [server is not down]
5: ack
6: cancelTimeOut(t)
7: t expired
8: increment(m)
loop m
9: ws1 = prepareNextReplica()
10: switchOff(ws)
11: switchOn(ws1)
10.1: shutdown()
11.1: start()
«gaStep» {hostDemand= (value=0.5; unit=ms; statQ=mean; source=mea)}
«gaStep» {hostDemand= (value=0.6; unit=s; statQ=mean; source=mea)}
«gaStep» {hostDemand= (value=70.1; unit=s; statQ=mean; source=mea)}
(b)

Figure 1: UML Sequence Diagram of the (a) *SwitchOverFailing* and (b) *Ping&Restore* Fault-Tolerant Techniques.

which is collecting the potentially harmful requests, and brings up a new (and clean) machine replica.

Ping And Restore. It provides a Monitor software component which observes the vulnerable system machines. When it finds some of these machines in an undefined state (i.e., affected by attacks), it brings down such a machine and brings up a new (and clean) replica.

Note that in both FTTs the machine replica may have a different operating system or software capabilities, as a way of mitigating incoming illegal requests.

Figure 1(a) illustrates the UML Sequence Diagram (UML-SD) of the *SwitchOverFailing* FTT. Grey notes indicate the system performance properties and they are specified as annotations by means of the MARTE profile. For example, the `gaStep` stereotype represents a part of the scenario (defined in sequence with other actions) for which it is possible to indicate the demands of such a part on the system resources, such as its execution on the host processor (called its `hostDemand` attribute).

Each external incoming request is sent to the `WatchDog` that analyses it; such an analysis requires processing resources (`hostDemand`) of *$analyse* milliseconds (ms), which is a mean value to be estimated.

Note that the request may be an attack or not, and it can be either detected or not detected. When the request is not an attack, the web server redirects the request to other services inside the system. Successful attack detection occurs with a probability of *$hitRate*. When the attack is detected and the threshold is reached, the `WatchDog` prepares a redundancy replica to be switched on, starts it (which has a duration of 70.1 seconds [27]), and then it switches off the server receiving the attacks, which has a cost of 0.6 seconds [27]. When the request is an attack but it is not detected, then the server collapses and it needs to be repaired, and we assumed it lasts for 30 minutes. For such a duration the server is inoperative, i.e., it is not attending new requests, because it represents the maintenance time, i.e., someone locally or remotely must fix the error or restart the server.

It is worth to notice that input parameters of the model, such as *$analyse*, *$hitRate*, are values set by the IDS which implements this FTT. That is, this model will be useful for performing sensitive analysis of different IDS solutions.

Figure 1(b) depicts the UML-SD of the *Ping&Restore* FTT. The cyclic behaviour of the monitor is as follows. It waits a certain amount of time (*$wait* ms) and then initialise a variable *m*, which counts the number of replicas in a non-functional state. The monitor sets a timeout having a duration of *$tOut* ms and iterates up to *k* machines to be recovered, asking for each machine if it is alive. When the machine answers, then the monitor cancels the timeout. Otherwise, the timeout is expired and the current machine is marked for recovering. When the number of machines to be recovered are reached or all machines have been inspected, then the monitor iterates preparing a new replica, switches off the faulty machine and switches on the new replica.

As in the previous case, it is worthy of mention that input parameters of the model, such as *$wait*, *$tOut*, are values useful for performing sensitive analysis of different monitor solutions.

3. OUR APPROACH

In this section we present a model-based methodology that allows to quantify the trade-off between the security strategies introduced to cope with the security attacks and the consequent performance degradation.

In Figure 2 the process that we propose is reported. The process has been partitioned in two sides: on the top-hand side all models that can be represented with a software modelling notation (e.g. UML) appear; on the bottom-hand side all models represented with a performance modelling notation (e.g. GSPN) appear.

The starting point of the process is a *Performance-Annotated*

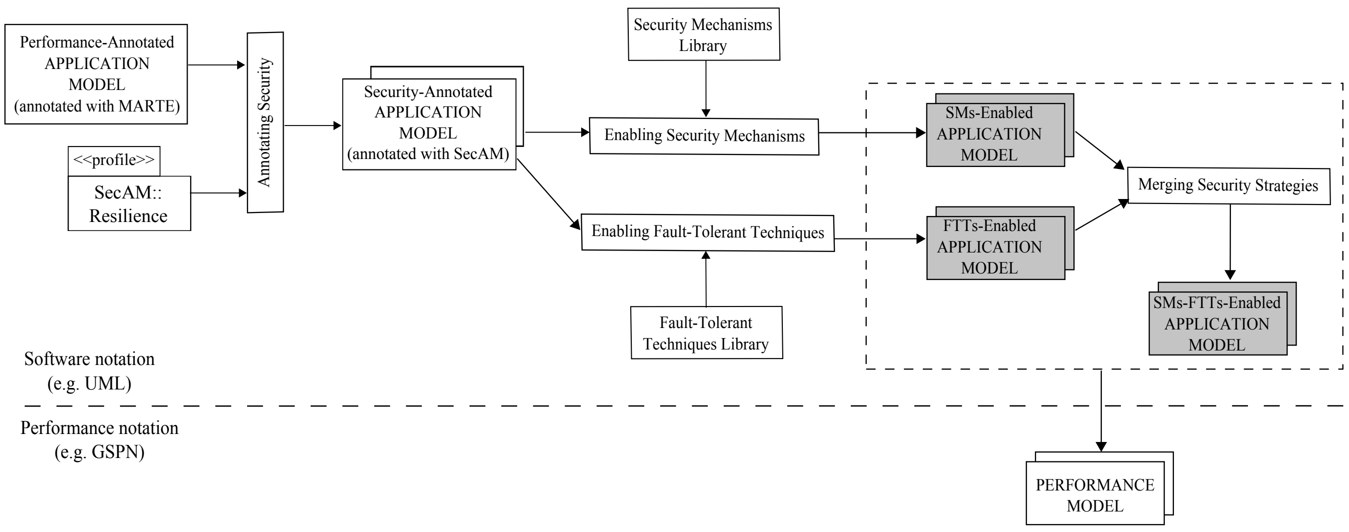

Performance-Annotated APPLICATION MODEL (annotated with MARTE)
<<profile>>
SecAM:: Resilience
Annotating Security
Security-Annotated APPLICATION MODEL (annotated with SecAM)
Security Mechanisms Library
Enabling Security Mechanisms
Enabling Fault-Tolerant Techniques
Fault-Tolerant Techniques Library
SMs-Enabled APPLICATION MODEL
FTTs-Enabled APPLICATION MODEL
Merging Security Strategies
SMs-FTTs-Enabled APPLICATION MODEL
Software notation (e.g. UML)
Performance notation (e.g. GSPN)
PERFORMANCE MODEL

Figure 2: A process to estimate the system performance while adding Security Mechanisms and Fault-Tolerant Techniques.

Application Model that is a static and dynamic representation of a software system. For sake of simplification, we assume that such a model is annotated with performance parameters related to the application such as the expected workload and system operational profile. The standard MARTE profile [21] has been adopted to specify performance parameters in our UML models.

A *Security-Annotated Application Model* is obtained by introducing security annotations in the former. Such annotations specify *where* security strategies have to be inserted, namely which software services have to be protected and how (e.g. some data must be encrypted). Security annotations have been incorporated by the *Resilience* package of the Security Analysis and Modelling (SecAM) profile [23, 24] that enables the specification of attacks, vulnerabilities and intrusions in UML models. Security attacks are characterised with their kind (i.e., flooding, spoofing or brute force), type (i.e., active or passive), objective (i.e. DoS, run arbitrary code or privilege escalation), class (i.e. virus, worm or buffer overflow) and occurrence rate (i.e. the probability of success).

The task of *Enabling Security Mechanisms* has been already presented in [8]. This step is driven by the security annotations specified in the application model, and a *SMs-Enabled Application Model* is finally obtained. As an example, if a security annotation specifies that data must be kept secret, an additional pattern with the steps needed for the encryption mechanism must be introduced in the system model. Such a pattern is one of the mechanisms modelled in our *Security Mechanisms Library* (see Section 2.1).

The task of *Enabling Fault-Tolerant Techniques* consists in embedding the appropriate fault-tolerant techniques in the system model, and a *FTTs-Enabled Application Model* is finally obtained. As an example, if an attack annotation specifies that spoofing can be performed for a certain service, an additional pattern with the steps needed for the FTT acting against such an attack must be introduced in the system model wherever the service is invoked. Such a pattern is one of the techniques modelled in our *Fault-Tolerant Techniques Library* (see Section 2.2).

Note that both the security strategies we consider (i.e., SMs and FTTs) can be analysed in isolation or can be jointly analysed while merging the previous models (i.e., the *SMs-Enabled* and *FTTs-Enabled* models) and a *SMs-FTTs-Enabled Application Model* is finally obtained.

A key aspect of our approach is the composability of models, and this is achieved through two features: (i) entry points for FTTs and SMs are unambiguously defined by security annotations, and (ii) models in the SMs and FTTs libraries have been designed to be easily composable with application models.

Shaded boxes of Figure 2 represent the models that can be finally transformed into GSPN-based *Performance Model*(s). This step involves not only a transformation between modelling notations[1], but an additional task is necessary to appropriately instrument the target performance model, because security strategies inevitably introduce additional performance parameters to be set in the model. The definition of such parameters is embedded in the security libraries where they are defined in an application-independent way. For example, the encryption mechanism introduces additional parameters affecting system performance, such as the complexity and resource requirements of the encryption algorithm, its mode of operation (e.g. CBC), the lengths of the keys, etc. Hence, the GSPN performance model finally generated has to be carefully parameterised with proper performance data.

The GSPN performance models can be solved by means of any available formal model analysis tools, such as the PeabraiN [25] simulator, and the model evaluation provides performance indices that jointly take into account the security strategies as well as the performance features of critical systems. Note that such a trade-off analysis can be conducted on multiple security settings by only modifying the security annotations and re-running the steps of our approach. In fact, in Figure 2 we can define a certain multiplicity in the security annotations to emphasise that different strategies can be adopted for the same system design according to different settings.

Finally we observe that several types of analysis can be conducted on the models built with this approach: (i) a performance model with a set of security requirements can be compared with one without security to simply study the performance degradation introduced from certain security strategies; (ii) the performance estimates from different performance models can be compared to each other to study the trade-off between security and performance across different design configurations.

[1]Well consolidated techniques have been exploited to transform software models (e.g. UML models) into performance models (e.g. GSPN), see [4] for an extensive survey on this topic.

4. CASE STUDY

Our approach has been applied to an E-Commerce System (ECS). It is a web-based system that manages business data: customers browse catalogues and make selections of items that need to be purchased; at the same time, suppliers can upload their catalogues, change the prices and the availability of products etc.

An overview of the ECS *Performance-Annotated Application Model* (see Figure 2) is depicted in Figure 3.

Figure 3(a) reports the UML Use Case Diagram representing the services we consider in our analysis. In particular, the *makePurchase* service is executed only if a customer has been properly logged into the system, i.e., by invoking the *login* service. A logged user can either make a purchase, with a probability of 0.7 (as indicated by the tagged value of `gaStep` stereotype of the MARTE profile), or asking for other services with a probability of 0.3. The `gaScenario` annotations in *makePurchase* remark the existence of two different scenarios, one which occurs with a probability of 0.25 and has a duration of 2.5ms, and the other one with a probability of 0.75 and an average duration of 7.5ms.

Figure 3(b) reports the UML Deployment Diagram. ECS has a number of web servers nodes which attend the incoming requests. Such servers are connected through a Wide Area Network (WAN) to a dispatcher node which forwards requests to a control node and a database node, through a Local Area Network (LAN). A monitor node is intentionally added to implement the *Ping&Restore* FTT, and it will be used when designing security strategies.

Figure 3(c) reports the UML Sequence Diagram of the *login* service. When a new login request arrives to the system, the web server redirects it to the dispatcher, which diverts it to the user controller. The latter component finally communicates with the database to get the actual user credentials (i.e., user name and password). Once the user controller receives the user credentials from the database, it verifies them against the ones provided by the user. If verification is successful (which happens 85% of times), then the customer is logged into the system, and the corresponding acknowledge is sent back to the web server.

Note that MARTE [21] annotations indicate, for instance, some performance features of the system. For example, the incoming requests to ECS are characterised by an incoming rate of *$cusRate*, in terms of milliseconds (`gaStep` stereotype), see Figure 3(c). The verification of user credentials (*verifyUserCredentials* method) consumes, on average, 12.4ms, as specified in the `hostDemand` tagged value, see Figure 3(c).

The rest of this Section is organized as follows. We firstly describe the experimental setting (see Section 4.1) of our case study: the step-wise application of the approach presented in Section 3 is discussed as well as the input parameters used for the experimentation. Then we collect the experimental results (see Section 4.2) of our case study: performance models are simulated and a performance index (i.e., the throughput of the system) is studied across different design configurations.

4.1 Experimental setting

The first step of our approach (see Figure 2) is to annotate the performance-annotated application model by means of the SecAM profile [23,24] in order to specify attacks, vulnerabilities and intrusions in UML models.

We assume that the *login* service can be the objective of external attackers that have the objective of bringing down the system while consuming its resources. The incoming requests are annotated through `secaAttackGenerator` stereotype describing an attack occurrence probability of rate *$attRate*. Hence, we obtain a *Security-Annotated Application Model* (see Figure 2).

Security annotations indicate to add several SMs and FTTs. In particular, for our experimentation we consider: (i) SMs, i.e., encryption and decryption, digital signature generation and verification; (ii) FTTs, i.e., switch over failing and ping&restore. An overview of the ECS *SMs-FTTs-Annotated Application Model* (see Figure 2) is depicted in Figure 4.

The activity of enabling Security Mechanisms (see Figure 2) leads to introduce SMs in the communication between the web server and the dispatcher. Figure 4 shows that before sending data the web server generates a digital signature (box *DSGeneration*) and encrypts the credentials inserted by the users (box *Encryption*). Both the digital signature and encrypted data are forwarded from the dispatcher to the user controller and finally to the database. This latter component needs to decrypt (box *Decrypt*) the received data and verifies the digital signature (box *DSVerification*). Hence, we obtained a *SMs-Enabled Application Model* (see Figure 2). We do not execute the performance analysis of such model because such experimentation has been already performed in [8], hence we decided to only analyse such model in conjunction with application models equipped with FTTs.

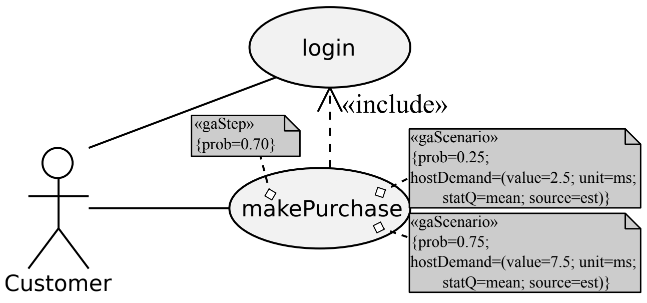
The activity of enabling Fault-Tolerant Techniques (see Figure 2) leads to introduce FTTs in order to protect the web server. Both the FTTs we presented in Section 2.2 are considered in our experimentation. In particular, the addition of FTTs gives rise to two different system models: (i) the *FTTs-Enabled Application Model (SoF)*, where only the *Switch Over Failing* FTT has been considered; (ii) the *FTTs-Enabled Application Model (P&R)*, where only the *Ping And Restore* FTT has been introduced. The SoF technique is depicted in Figure 4, whereas a monitor node is intentionally added to implement the P&R technique, as reported in Figure 3(b). Hence, we obtained two *FTTs-Enabled Application Model*s (see Figure 2).

The input parameters used for our experimentation have been reported in Table 1 (system resources and number of their instances) and Table 2 (timing of system actions).

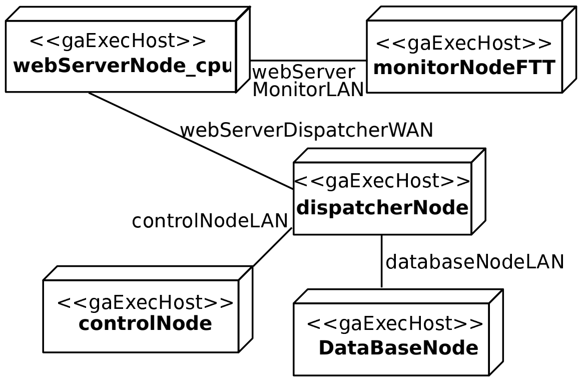
As already mentioned in Section 3, the definition of security parameters is embedded in the Security-Annotated Application Model (see Figure 2) where they are defined in an application-independent way. However, the task of enabling security implies the usage of such strategies at the application level, thus they can be influenced by further application-dependent characteristics. For example, the encryption mechanism efficiency is influenced by the key length of the encryption algorithm, the speed of the CPU executing the encryption algorithm, the length of the message to be encrypted, etc. In particular, we refer to [2, 18, 27] in order to determine reliable numerical values, whereas application-dependent parameters come from the experimentation we conducted in [9].

In the sequel of this section the input parameters are defined as follows. IDS parameters (*$analyse*, *$hitRate*) have been chosen following values of an IDS given in [2] (a mean value for analysing of 32.04ms, and a generic-attack detection rate of 71.4%). The input parameters for the monitor (*$wait*, *$tOut*) have been set to 5 minutes and 1ms, respectively. Timing of recovering replicas have been taken from [27]. The timing values of the referenced UML-SD SMs have been chosen from [9] and [18] while considering the MD5 hash algorithm with a public key length of 1024 bits.

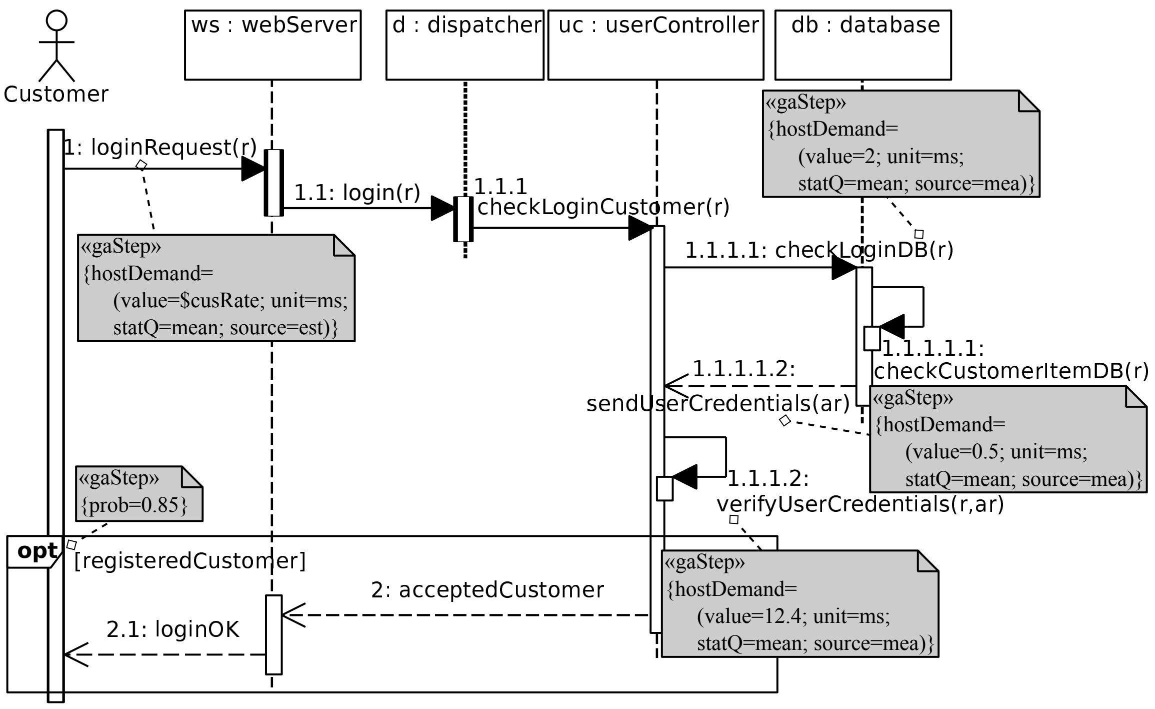
For incoming requests, we have set a rate equal to 37 visitors per second as happens in the Amazon site. We have estimated a think time of registered customers of 2 minutes, and after such a time, then the customer may decide either logout or make a purchase, having a probability of 0.3 and 0.7, respectively. As it is shown in Figure 3(a), a new purchase may have two different scenarios, each one with different duration.


login
«include»
«gaStep» {prob=0.70}
«gaScenario» {prob=0.25; hostDemand=(value=2.5; unit=ms; statQ=mean; source=est)}
makePurchase
«gaScenario» {prob=0.75; hostDemand=(value=7.5; unit=ms; statQ=mean; source=est)}
Customer

(a) UML Use Case Diagram.

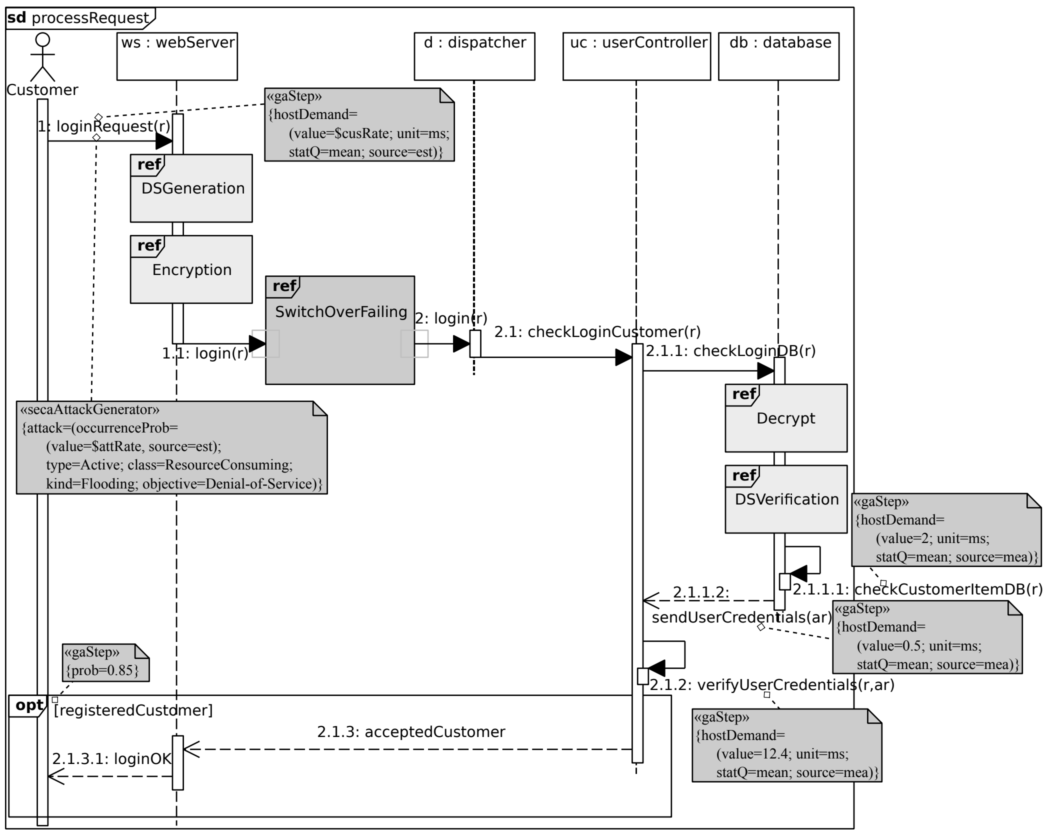

<<gaExecHost>> webServerNode_cpu
webServer MonitorLAN
<<gaExecHost>> monitorNodeFTT
webServerDispatcherWAN
<<gaExecHost>> dispatcherNode
controlNodeLAN
databaseNodeLAN
<<gaExecHost>> controlNode
<<gaExecHost>> DataBaseNode

(b) UML Deployment Diagram.


Customer
ws : webServer
d : dispatcher
uc : userController
db : database
«gaStep» {hostDemand= (value=2; unit=ms; statQ=mean; source=mea)}
1: loginRequest(r)
1.1: login(r)
1.1.1 checkLoginCustomer(r)
«gaStep» {hostDemand= (value=$cusRate; unit=ms; statQ=mean; source=est)}
1.1.1.1: checkLoginDB(r)
1.1.1.1.1: checkCustomerItemDB(r)
1.1.1.1.2: sendUserCredentials(ar)
«gaStep» {hostDemand= (value=0.5; unit=ms; statQ=mean; source=mea)}
«gaStep» {prob=0.85}
1.1.1.2: verifyUserCredentials(r,ar)
opt
[registeredCustomer]
2: acceptedCustomer
2.1: loginOK
«gaStep» {hostDemand= (value=12.4; unit=ms; statQ=mean; source=mea)}

(c) UML Sequence Diagram.

Figure 3: ECS- Performance-Annotated Application Model.



sd processRequest
Customer
ws : webServer
d : dispatcher
uc : userController
db : database
1: loginRequest(r)
«gaStep» {hostDemand= (value=$cusRate; unit=ms; statQ=mean; source=est)}
ref DSGeneration
ref Encryption
ref SwitchOverFailing
1.1: login(r)
2: login(r)
2.1: checkLoginCustomer(r)
2.1.1: checkLoginDB(r)
ref Decrypt
ref DSVerification
«secaAttackGenerator» {attack=(occurrenceProb= (value=$attRate, source=est); type=Active; class=ResourceConsuming; kind=Flooding; objective=Denial-of-Service)}
«gaStep» {hostDemand= (value=2; unit=ms; statQ=mean; source=mea)}
2.1.1.1: checkCustomerItemDB(r)
2.1.1.2: sendUserCredentials(ar)
«gaStep» {hostDemand= (value=0.5; unit=ms; statQ=mean; source=mea)}
«gaStep» {prob=0.85}
2.1.2: verifyUserCredentials(r,ar)
opt [registeredCustomer]
2.1.3: acceptedCustomer
2.1.3.1: loginOK
«gaStep» {hostDemand= (value=12.4; unit=ms; statQ=mean; source=mea)}

Figure 4: ECS - SMs-FTTs-Enabled Application Model.

Resource	No. instances
`webServer`	50
`dispatcher`	40
`userController`	30
`database`	20
`webServer` (replicas)	5
`watchDog`	5

Table 1: Input parameters: system resources and number of instances.

Method name/UML-SD	Duration (ms)
`login`	0.5
`checkLoginCustomer`	0.5
`checkLoginDB`	0.5
`checkCustomerItemDB`	2
`sendUserCredentials`	0.5
`verifyCustomerCredentials`	12.4
`acceptedCustomers`	0.5
`loginOK`	0.5
UML-SD DSGeneration	107
UML-SD DSVerification	68
UML-SD Encryption	117
UML-SD Decryption	117

Table 2: Input parameters: execution times of system actions.

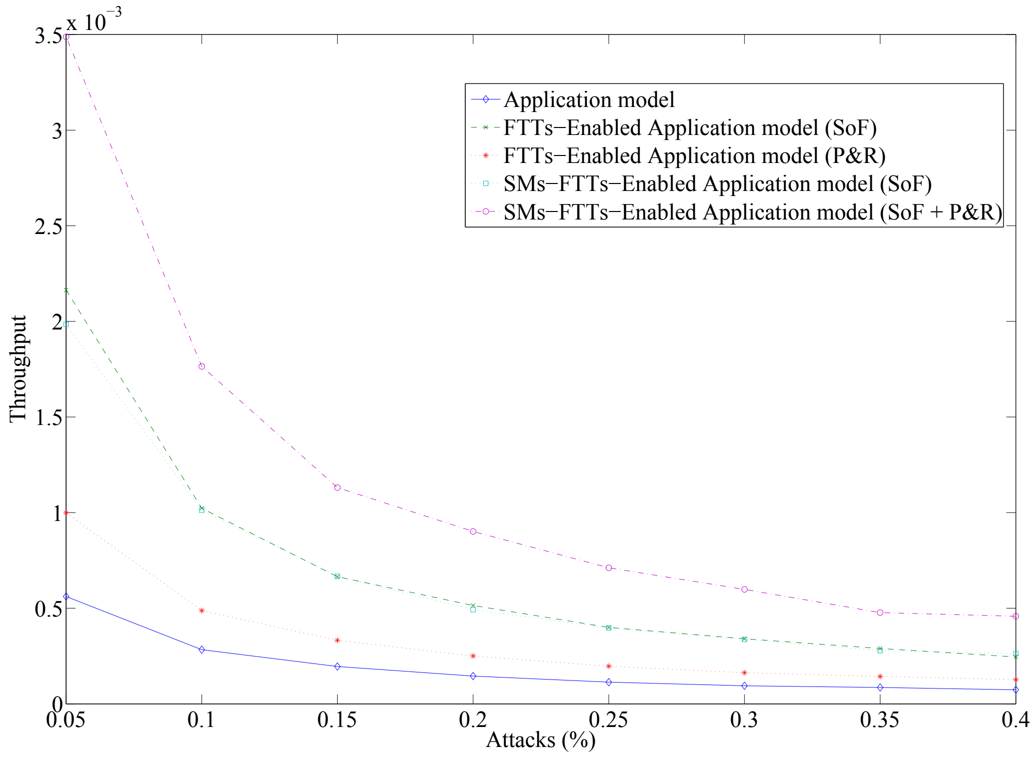
4.2 Experimental results

The experimentation has been conducted while considering the following scenarios: (i) the Performance-Annotated Application Model (see Figure 3); (ii) the FTTs-Enabled Application Model (SoF), i.e., without SMs but with *SwitchOverFailing* FTT only; (iii) the FTTs-Enabled Application Model (P&R), i.e., without SMs but with *Ping&Restore* FTT only; (iv) the SMs-FTTs-Enabled Application Model (SoF), i.e., with SMs and the *SwitchOverFailing* FTT only (see Figure 4); (v) the SMs-FTTs-Enabled Application Model (SoF + P&R), i.e., with SMs and both FTTs.

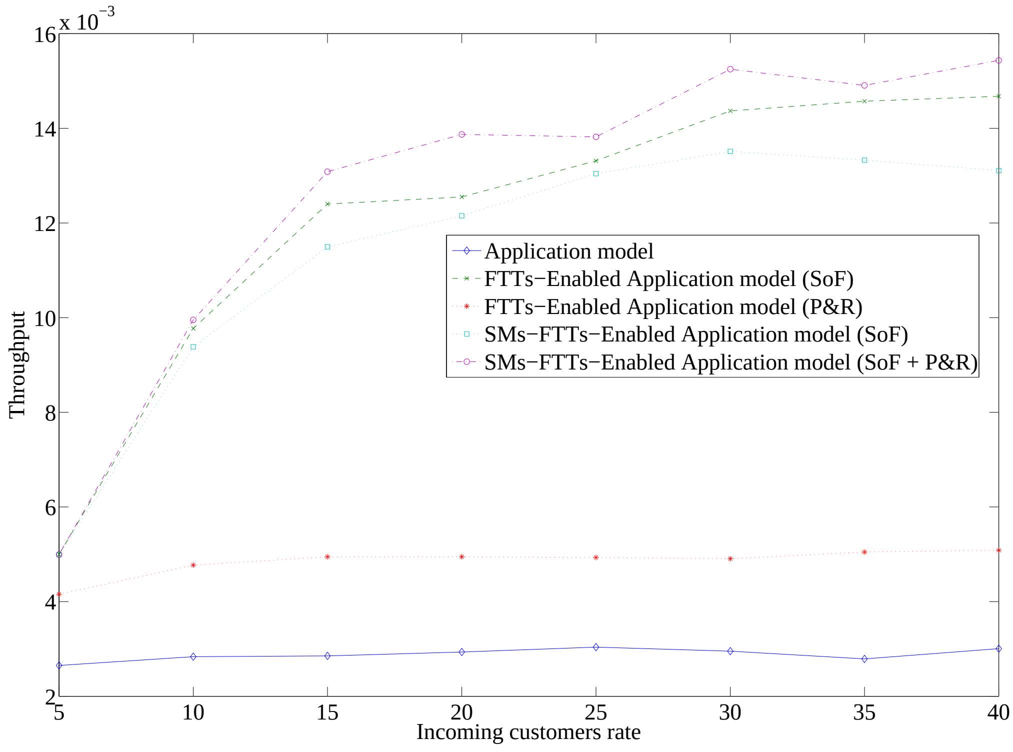
The transformation from UML software models to GSPN performance models has been carried out by ArgoSPE [13] tool. We have used the PeabraiN simulator [25], which is a PNML-compliant tool and allows to simulate GSPNs in transient mode. We have simulated an execution of the system of 2 hours with the input parameters reported in Tables 1 and 2.

Figure 5 shows the experimental results. Figure 5(a) reports the system throughput (transactions completed per unit of time) while varying attack rates from 0.05 to 0.4. When we consider attacks, the system throughput of the performance-annotated application model quickly drops down, reaching values lower than 10^{-4}. In fact, when the request is an attack but it is not detected, then the server collapses and it needs to be repaired, and such procedure lasts for 30 minutes.

On the contrary, when FTTs are enabled, the system is able to mitigate the effects of attacks, maintaining a certain level of server availability. However, the throughput of the FTTs-Enabled Application Model (SoF) is greater than the throughput of the FTTs-Enabled Application Model (P&R). Finally, we can observe that



(a) Throughput of the system while varying attacks rate.



(b) Throughput of the system while varying incoming customer rate.

Figure 5: ECS - Performance Analysis Results.

when we consider a scenario with SMs and both FTTs then the throughput outperforms any other combination. Ultimately, if the system is subjected to an increasing probability of attacks, then a better throughput is achieved while considering SMs and both FTTs, rather than considering FTTs in an isolated way.

Figure 5(b) reports the system throughput while varying the incoming customers rate from 5 to 40, and with a fixed attack rate of 1%, in all the considered scenarios. As it is shown, the throughput in the performance-annotated application model and with the P&R FTT only remains quite constant despite the increasing of the incoming customers rate. The throughput in the latter scenario, however, outperforms the former. In the rest of scenarios, the more incoming customers, the more throughput is achieved. The highest throughput is obtained in the scenario where SMs and both FTTs have been added. These results show that such scenario, i.e., SMs-FTTs-Enabled Application Model (SoF + P&R), is able to successfully support the increasing rate of incoming customers.

We can conclude that the conjunction of both FTTs techniques is beneficial for the application model. As future work we plan to investigate the system throughput while varying the probability of detecting attack conditions, i.e., by increasing the detection rate of the IDS algorithm.

We recall that the goal of this paper is to validate a methodology that applies FTTs and SMs at the architectural level by enabling the possibility of computing performance impact before deployment. More in general, several security capabilities can be tested to find the most suitable options.

From a performance analysis viewpoint, our experimentation follows standard practices: a performance model is built, instrumented with input parameters and finally evaluated through simulation. Further experimentation can be conducted by instrumenting the model with different numerical values for the input parameters. As future work, we plan to apply our approach to other real world examples in order to assess the scalability of the framework.

5. RELATED WORK

The problem of analysing the performance of security technologies has been widely addressed in literature, in particular most of the studies focus attention on the performance of existing standards such as IPsec and SSL. Examples of research investigation in this direction can be found in [5, 14, 16].

Security properties are often considered in trade-off with other features, for example in [18] the security is considered while minimising performance penalties. Our aim is similar to this one because we also target an analysis of how security strategies impact on system performance. However, in [18] the analysis is conducted using a specific security protocol (i.e., SSL) and a limited set of cryptographic algorithms, whereas our methodology is intended to enlarge the set of design options while modelling and analysing more general solutions.

Estimating the performance of a system with different security properties is a difficult task, as demonstrated in [15], where different measurements on different platforms have been performed to compare secure and non-secure Web services, RMI and RMI with SSL. Our work differs from [15] because we estimate the system performance before the deployment with the possibility of targeting different platforms.

An experimental approach with regard to the performance evaluation of security services is presented in [6] where security applications are planned and implemented with embedded security strategies, and subsequently monitored. Our approach differs from [6] because we adopt a model-based approach to predict the system performance, hence no implementation of the system is required.

Some works use the aspect-oriented modeling (AOM) paradigm to specify and integrate security risks and strategies into a system model, such as the one in [28]. It models security solutions as aspects in UML, and the annotated model is transformed into a performance model. This work uses an approach to the problem that is similar to ours, in that they are both based on model annotations and transformations. However, our work targets the problem of representing security strategies while guaranteeing certain security properties, whereas the analysis in [28] is only performed on the SSL protocol.

The lack of a model-based approach to this problem is the major motivation behind our work. This paper aims at overcoming the limitations of ad-hoc solutions (i.e., the well assessed security protocols like IPsec and SSL) that estimate the performance of specific security technologies. To achieve this goal, we propose a methodology that makes use of platform-independent models of security strategies (i.e., Fault-Tolerant techniques, security mechanisms) with the aim to inform software designers about the performance of different design solutions for critical systems.

6. CONCLUSION AND FUTURE WORK

Security attacks aim to system vulnerabilities that, when achieve success, may lead to system failures. As an attempt to mitigate these effects, software designers use to introduce Fault-Tolerant Techniques (FTTs) and/or Security Mechanisms (SMs). However, FTTs and SMs inevitably consume system resources, hence they influence system performance, in the worst case affecting its full operability.

In this paper we provided a model-based methodology able to quantitatively estimate the system performance while introducing FTTs and/or SMs aimed at protecting critical systems. The main goal of this methodology is to introduce different security models and compose them with software architectural models, thus to support software designers to find appropriate security strategies while meeting performance requirements.

We validated our proposal by applying it to a case study. The experiments put on evidence that our approach enables the estimation of system performance when adding security protection strategies, and sensitive analysis (testing various security alternatives) can be carried out as support while designing critical systems.

There exist many security techniques that may affect system performance, such as the use of firewalls, security protocols, remote logging, etc. For this paper, we only considered a subset of FTTs and SMs, however, as future work, the subset of techniques may be enlarged to enable the verification of (possibly future) techniques.

We consider this work as a starting point for investigating even more sophisticated tradeoffs, for example it would be relevant to study the tradeoff between security and other non-functional attributes, such as availability. In particular, addressing the problem of quantifying and locating data replicas for availability purposes without heavily affecting the security of the system may be crucial in certain domains.

Finally, we plan to automate the steps of our approach by means of a tool, thus to provide guidelines to software designers about the best choices of Fault-Tolerant techniques and security mechanisms for the attacks systems may suffer.

7. ACKNOWLEDGMENTS

The authors would like to thank Vittorio Cortellessa who deeply reviewed and commented this paper, so considerably improving its quality. This work has been partially supported by VISION

ERC project (ERC-240555), by Fundación Aragón I+D, DGA (CONAID), and by Spanish project DPI2010-20413.

8. REFERENCES

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing. John Wiley and Sons, 1995.

[2] D. Ariu, R. Tronci, and G. Giacinto. HMMPayl: An intrusion detection system based on Hidden Markov Models. *Computers & Security*, 30(4):221–241, 2011.

[3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, jan.-march 2004.

[4] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Trans. Software Eng.*, 30(5):295–310, 2004.

[5] M. Blaze, J. Ioannidis, and A. D. Keromytis. Trust Management for IPsec. *ACM Trans. Inf. Syst. Secur.*, 5(2):95–118, 2002.

[6] A. Cilardo, L. Coppolino, A. Mazzeo, and L. Romano. Performance Evaluation of Security Services: An Experimental Approach. In *Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, PDP ’07, pages 387–394, Washington, DC, USA, 2007. IEEE Computer Society.

[7] V. Cortellessa and C. Trubiani. Towards a library of composable models to estimate the performance of security solutions. In *Workshop on Software and Performance (WOSP)*, pages 145–156, 2008.

[8] V. Cortellessa, C. Trubiani, L. Mostarda, and N. Dulay. An Architectural Framework for Analyzing Tradeoffs between Software Security and Performance. In H. Giese, editor, *ISARCS’10: Proceedings of the 1st International Symposium on Architecting Critical Systems*, volume 6150 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2010.

[9] V. Cortellessa, C. Trubiani, L. Mostarda, and N. Dulay. An Architectural Framework for Analyzing Tradeoffs between Software Security and Performance - Extended results . Technical report, Università degli Studi dell’Aquila, 2010. TRCS 001/2010.

[10] J. Delatour and F. de Lamotte. ArgoPN: a CASE Tool Merging UML and Petri Nets. In P. T. Isaías, F. Sedes, J. C. Augusto, and U. Ultes-Nitsche, editors, *NDDL/VVEIS*, pages 94–102. ICEIS Press, 2003.

[11] S. Distefano, M. Scarpa, and A. Puliafito. From UML to Petri Nets: The PCM-Based Methodology. *IEEE Transactions on Software Engineering*, 37(1):65–79, jan.-feb. 2011.

[12] L. Garber. Denial-of-Service Attacks Rip the Internet. *IEEE Computer*, 33(4):12–17, 2000.

[13] E. Gómez-Martínez and J. Merseguer. ArgoSPE: Model-Based Software Performance Engineering. In *International Conference of Application and Theory of Petri Nets*, pages 401–410, 2006.

[14] V. Gupta, S. Gupta, S. C. Shantz, and D. Stebila. Performance Analysis of Elliptic Curve Cryptography for SSL. In *Proceedings of the 1st ACM workshop on Wireless security*, WiSE ’02, pages 87–94, 2002.

[15] M. B. Juric, I. Rozman, B. Brumen, M. Colnaric, and M. Hericko. Comparison of performance of Web services, WS-Security, RMI, and RMI-SSL. *J. Syst. Softw.*, 79:689–700, May 2006.

[16] K. Kant, R. Iyer, and P. Mohapatra. Architectural Impact of Secure Socket Layer on Internet Servers. In *Proceedings of the 2000 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, ICCD ’00, pages 7–14, Washington, DC, USA, 2000. IEEE Computer Society.

[17] F. Lagarde, H. Espinoza, F. Terrier, and S. Gérard. Improving UML Profile Design Practices by Leveraging Conceptual Domain Models. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, ASE’07, pages 445–448, New York, NY, USA, November 2007. ACM.

[18] D. Menascé. Security Performance. *IEEE Internet Computing*, 7(3):84–87, 2003.

[19] D. A. Menascé and A. F. A. Virgilio. *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. Prentice Hall, Upper Saddle River, NJ, USA, 1st edition, May 2000.

[20] OMG. *Unified Modelling Language: Superstructure*. Object Management Group, July 2005. Version 2.0, formal/05-07-04.

[21] OMG. *A UML profile for Modeling and Analysis of Real Time Embedded Systems (MARTE)*. Object Management Group, 2009. Document ptc/09-11-02.

[22] R. J. Rodríguez and J. Merseguer. Integrating Fault-Tolerant Techniques into the Design of Critical Systems. In H. Giese, editor, *Proceedings of the 1st International Symposium on Architecting Critical Systems (ISARCS)*, volume 6150 of *Lecture Notes in Computer Science*, pages 33–51, Prague, Czech Republic, June 2010. Springer.

[23] R. J. Rodríguez, J. Merseguer, and S. Bernardi. Modelling and Analysing Resilience as a Security Issue within UML. In *Proceedings of the 2nd International Workshop on Software Engineering for Resilient Systems (SERENE)*, London, United Kingdom, April 2010. ACM.

[24] R. J. Rodríguez, J. Merseguer, and S. Bernardi. A Security Analysis and Modelling profile: an Overview. Technical Report RR-01-11, Dpto. de Ingeniería e Informática de Sistemas, Universidad de Zaragoza, 2011.

[25] R. J. Rodríguez, J. Júlvez, and J. Merseguer. PeabraiN: A PIPE Extension for Performance Estimation and Resource Optimisation. In *Proceedings of the 12th International Conference on Application of Concurrency to System Designs*, 2012. Accepted for publication.

[26] B. Selic. A Systematic Approach to Domain-Specific Language Design Using UML. In *10th IEEE Int. Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 2–9, Santorini Island, Greece, May 2007. IEEE Computer Society.

[27] P. Sousa, A. Bessani, M. Correia, N. Neves, and P. Verissimo. Highly Available Intrusion-Tolerant Services with Proactive-Reactive Recovery. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):452–465, april 2010.

[28] M. Woodside, D. C. Petriu, D. B. Petriu, J. Xu, T. Israr, G. Georg, R. France, J. M. Bieman, S. H. Houmb, and J. Jürjens. Performance analysis of security aspects by weaving scenarios extracted from UML models. *J. Syst. Softw.*, 82:56–74, January 2009.