TESIS DE LA UNIVERSIDAD DE ZARAGOZA

2013

52

Ricardo Julio Rodríguez Fernández

Perfomance Analysis and Resource Optimisation of Critical Systems Modelled by Petri Nets

Departamento

Informática e Ingeniería de Sistemas

Director/es

Merseguer Hernáiz, José Javier

Júlvez Bueno, Jorge Emilio

ISSN 2254-7606

Prensas de la Universidad Universidad Zaragoza

Universidad Zaragoza
1542

Tesis Doctoral

PERFOMANCE ANALYSIS AND RESOURCE OPTIMISATION OF CRITICAL SYSTEMS MODELLED BY PETRI NETS

Autor

Ricardo Julio Rodríguez Fernández

Director/es

Merseguer Hernáiz, José Javier
Júlvez Bueno, Jorge Emilio

UNIVERSIDAD DE ZARAGOZA

Informática e Ingeniería de Sistemas

2013

Performance Analysis and Resource Optimisation of Critical Systems Modelled by Petri Nets

Ricardo Julio Rodríguez Fernández

Ph.D. DISSERTATION

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza

Advisors: Dr. Jorge Emilio Júlvez Bueno
Dr. José Javier Merseguer Hernáiz

Junio de 2013

A María y su paciencia infinita. ※

Mi locura es sagrada. No me toquen.
(Iliana Godoy)

Quod quisque possit, nisi tentando nesciat.
(No se puede saber de lo que cada uno es capaz si no se pone a prueba)
(Publilius Syrus)

Agradecimientos

Mis primeras palabras de agradecimiento son para Jorge Júlvez y José Merseguer, grandes profesionales – muy grandes – y mejores personas, que me han sabido guiar a buen puerto en todo momento en esta carrera de fondo llena de desniveles. Me han dejado mi libertad de crítica, pensamiento y acción, siempre aderezada con unos toques de advertencia cuando me descarriaba demasiado del objetivo. Gran parte de este trabajo es gracias a ellos, así que chicos, una vez más, muchas gracias!

No me puedo olvidar de Javi, mi padrino científico, quien también metía en vereda a la cabra cuando tiraba demasiado para el monte, y con quien he disfrutado buenos momentos tanto profesionales como personales. Tampoco al resto de gente – y algunos antiguos compañeros de promoción – con quien he tenido el gusto de compartir laboratorio durante este período (Irina, Estíbaliz, Jorge, Roberto, Guillermo, Juan...), sobremesas y discusiones bizarras (Javier, Jorge, Diego, Nacho, Roberto), compartido horas de trabajo en común, tanto en España como en otros lugares (Simona, Rafa, Catia), y, compartido también, sobre todo, muchas horas de cantina y otros menesteres (Ritu, Jesús, Diego).

Tampoco puedo olvidarme de esas grandes personas y buenos amigos que he tenido el gusto de conocer y trabajar con ellos durante mis estancias en Cardiff, como Omer, Yaser, Ioan o Raquel, con quienes he compartido muy buenos momentos que han hecho que me sintiera como en casa.

Por último, a mi familia (gracias por apoyarme siempre, dejarme estudiar lo que quise, y pagarme una educación en una universidad pública), amigos y ex-compañeros y amigos de promoción (Fergus, Jacobo), quienes han tenido el gusto – o la desgracia, según el día... – de soportar mis idas y venidas, mis cambios de humor, mis frustaciones y alegrías, durante este largo período.

En resumen, a todos aquellos con los que en algún momento me he cruzado durante este periplo de cuatro años, que hoy llega a su final, y que me han tenido que sufrir de un modo u otro. Gracias.

Performance Analysis and Resource Optimisation of Critical Systems Modelled by Petri Nets

Resumen

Un sistema crítico debe cumplir con su misión a pesar de la presencia de problemas de seguridad. Este tipo de sistemas se suele desplegar en entornos heterogéneos, donde pueden ser objeto de intentos de intrusión, robo de información confidencial u otro tipo de ataques. Los sistemas, en general, tienen que ser rediseñados después de que ocurra un incidente de seguridad, lo que puede conducir a consecuencias graves, como el enorme costo de reimplementar o reprogramar todo el sistema, así como las posibles pérdidas económicas. Así, la seguridad ha de ser concebida como una parte integral del desarrollo de sistemas y como una necesidad singular de lo que el sistema debe realizar (es decir, un requisito no funcional del sistema). Así pues, al diseñar sistemas críticos es fundamental estudiar los ataques que se pueden producir y planificar cómo reaccionar frente a ellos, con el fin de mantener el cumplimiento de requerimientos funcionales y no funcionales del sistema.

A pesar de que los problemas de seguridad se consideren, también es necesario tener en cuenta los costes incurridos para garantizar un determinado nivel de seguridad en sistemas críticos. De hecho, los costes de seguridad puede ser un factor muy relevante ya que puede abarcar diferentes dimensiones, como el presupuesto, el rendimiento y la fiabilidad.

Muchos de estos sistemas críticos que incorporan técnicas de tolerancia a fallos (sistemas FT) para hacer frente a las cuestiones de seguridad son sistemas complejos, que utilizan recursos que pueden estar comprometidos (es decir, pueden fallar) por la activación de los fallos y/o errores provocados por posibles ataques. Estos sistemas pueden ser modelados como sistemas de eventos discretos donde los recursos son compartidos, también llamados sistemas de asignación de recursos. Esta tesis se centra en los sistemas FT con recursos compartidos modelados mediante redes de Petri (Petri nets, PN). Estos sistemas son generalmente tan grandes que el cálculo exacto de su rendimiento se convierte en una tarea de cálculo muy compleja, debido al problema de la explosión del espacio de estados. Como resultado de ello, una tarea que requiere una exploración exhaustiva en el espacio de estados es incomputable (en un plazo prudencial) para sistemas grandes.

Las principales aportaciones de esta tesis son tres. Primero, se ofrecen diferentes modelos, usando el Lenguaje Unificado de Modelado (Unified Modelling Language, UML) y las redes de Petri, que ayudan a incorporar las cuestiones de seguridad y tolerancia a fallos en primer plano durante la fase de diseño de los sistemas, permitiendo así, por ejemplo, el análisis del compromiso entre seguridad y rendimiento. En segundo lugar, se proporcionan varios algoritmos para calcular el rendimiento (también bajo condiciones de fallo) mediante el cálculo de cotas de rendimiento superiores, evitando así el problema de la explosión del espacio de estados. Por último, se proporcionan algoritmos para calcular cómo compensar la degradación de rendimiento que se produce ante una situación inesperada en un sistema con tolerancia a fallos.

Preface

A critical system must fulfil its mission despite the presence of security issues. These systems are usually deployed in heterogeneous environments, where they are subject to suffer security issues, such as intrusion attempts, confidential data theft or other type of attacks. Systems usually need to be redesigned after a *security disaster*, which can lead to severe consequences, such as the huge cost of reimplementing or redeploying all the system, as well as economic losses. Security has to be conceived as an integral part of the development process and as a singular need of what the system should perform (i.e., a non-functional requirement). Thus, when designing critical systems it is fundamental to study the attacks that may occur and plan how to react to them, in order to keep fulfilling the system functional and non-functional requirements.

Despite considering security issues, it is also necessary to consider the costs incurred to guarantee a certain security level in critical systems. In fact, security costs can be very relevant and may span along different dimensions, such as budgeting, performance and reliability.

Many of these critical systems that incorporate Fault-Tolerant (FT) techniques to deal with security issues are complex systems using resources that are compromised (i.e., they fail) by the activation of faults. These systems can be naturally modelled as Discrete Event Systems (DES) where resources are shared, also called Resource Allocation Systems (RAS). In this dissertation, we focus on FT systems using shared resources modelled as Petri nets (PNs) as formal model language. These systems are usually so large that make the exact computation of their performance a highly complex computational task, due to the well-known state explosion problem. As a result, a task that requires an exhaustive state space exploration becomes unachievable in reasonable time for large systems.

The main contribution of this dissertation is threefold. Firstly, we provide different models, expressed by means of the Unified Modelling Language (UML) and Petri nets (PNs), to bring security and FT issues into foreground while designing, then allowing the analysis of security-performance trade-off. Secondly, we provide several algorithms to compute the performance (also performability – i.e., performance under failure conditions) by means of upper throughput bounds, then avoiding the state space explosion problem. Lastly, we provide algorithms to compensate the throughput degradation produced by an unexpected situation in a FT system.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction and State of the Art

1.1 Motivation

Complex, large scale and distributed systems are required to fulfil their mission despite the presence of security issues. Today, one of the main challenges in the software engineering area is to devise methods for the development of such critical systems. The system requirements, also called properties, are expressed in the initial phases when developing a system. A system requirement defines how a system should be (functional) or should perform (non-functional).

Functional requirements are devoted to calculations, data processing or any other functionality that defines what the system is supposed to behave. On the contrary, non-functional requirements (or non-functional properties, NFPs) involve how the system is supposed to perform its activities (e.g., how many customers per unit of time can be attended by a service, how many bytes per second can be transferred by a network device, or how many bytes per unit of time can be ciphered with a cipher algorithm).

Examples of NFPs are performance, dependability or security of a system. Several works in the literature [Devanbu and Stubblebine, 2000, Wing, 2003, McGraw, 2004, Barnum and McGraw, 2005, Khan and Zulkernine, 2008, Mouratidis and Giorgini, 2008] remark that security has not been conceived as an integral part of the development process and claim for it. Nowadays, most of the systems are deployed without taking into account security. Thus, after a *security disaster* the systems usually need to be redesigned, accounting for security from the beginning. This "fix it later" approach can lead to severe consequences, such as the huge cost of reimplementing or redeploying all the system, as well as economic losses [Randimbivololona, 2001] due to the unavailability of services or the disclosure of personal customers data (as the case of Sony PlayStation Network or RSA company breach in 2011).

Thus, the new generation of development methods have to face challenges never intended before, where security plays a main role. In our opinion, these methods need to address at

least the following issues to start considering the menaces to security successfully:

1. They should provide an *integrated* approach. By integrated we mean the need of considering security as a *first-class citizen* in all the stages of the software and systems life-cycle: Analysis, Design, Implementation, Testing, Deployment and Maintenance. So security needs to be integrated in the life-cycle of systems as functional properties (FP) are today. As we mentioned before, many works in the literature [Devanbu and Stubblebine, 2000, Wing, 2003, McGraw, 2004, Khan and Zulkernine, 2008,Mouratidis and Giorgini, 2008] claim for the necessity for this integration.

2. The methods should promote the *analysis* of system security even before the deployment of the system. This challenge links with the previous one since the analysis of security is then just considered as a new stage inside the life-cycle. For instance, as it was pointed out in [Randimbivololona, 2001], the cost of verification in the avionics system domain is the 50% of overall costs when the system is already deployed. This trend should change when analysis is really coupled with the development, thus saving costs.

3. The methods should promote a *unified* view of the security issues. Nowadays, we realise that the different – and sometimes orthogonal – aspects of security (e.g., access control or cryptography) are dealt by specialised communities that unfortunately do not share goals nor even vocabulary. This fact impairs communication and the possibility to integrate advances from different communities.

4. These methods should not be oblivious to the current software engineering (SE) techniques. For example, the Model-Driven Development (MDD) paradigm can play an important role in the generation of models for the analysis of security in software systems.

Moreover, large scale and distributed systems are deployed in heterogeneous environments where they are subject to suffer security issues, such as intrusion attempts, confidential data theft or other type of attacks. For example, a Denial-of-Service (DoS) attack [Garber, 2000] sends multiple requests to a server with the intention of consuming its resources and, in last term, bringing it 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 cut their services off by saturation.

Relevant efforts of software designers are devoted to devise the security strategies suitable to protect information and computational systems against not authorised accesses. Indeed, when designing critical systems it is fundamental to study the attacks that may occur and plan how to react to them. The occurrence of attacks in software systems leads software designers to introduce different Fault-Tolerant Techniques (FTTs), such as recov-

ery procedures, and/or Security Mechanisms (SMs), such as encryption of data, in order to react to intrusions or other type of attacks.

Despite these efforts, it is necessary to consider the costs 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 [Menascé and Virgilio, 2000, Menascé, 2003]. Therefore, FTTs and SMs inevitably consume system resources hence they influence the performance, even affecting its full operability. The security and performance trade-off becomes clear: **the addition of security into a system must assure that the system guarantees a minimal level of functionality** (i.e., to maintain a certain system throughput).

This dissertation steps forward in such a trade-off. Firstly, we bring security into foreground while designing, and secondly, we provide several strategies for performance (and performability – i.e., performance under failure conditions) analysis as well as resource optimisation analysis.

About considering security during design phase, we propose a Unified Modelling Language (UML) [OMG, 2005] extension (through *profiling*), called `SecAM` (stands for Security Analysis and Modelling) which enhances UML modelling expressiveness by providing security-related concepts, and we also propose FTTs models that can be easily added to system design and make easier its analysis.

Many of these critical systems that incorporate FT techniques, then called FT systems, are complex systems using shared resources that are compromised or even fail by the activation of faults. These systems can be naturally modelled as Discrete Event Systems (DES) where resources are shared, also called Resource Allocation Systems (RAS) [Colom, 2003]. In this dissertation, we focus on FT systems using shared resources modelled as Petri nets (PNs) as formal model language – more precisely, as process Petri nets [Tricas, 2003].

Unfortunately, these systems are usually large what makes the exact computation of their performance a highly complex computational task. The main reason for this complexity is the well-known state explosion problem. As a result, a task that requires an exhaustive state space exploration becomes unachievable in reasonable time for large systems.

Therefore, our main contributions on performance and resource optimisation analysis are: firstly, an iterative strategy to compute upper throughput bounds closer to the real throughput[1] than the ones that can be achieved in previous works [Chiola et al., 1993, Campos et al., 1992]; secondly, an iterative algorithm to compute the number of resources that mitigate the impact of activation of faults in FT systems; and thirdly, an Integer Linear Programming Problem (ILPP) that minimises the cost of compensation needed for maintaining a given throughput in a FT system.

Such strategies make use of linear programming techniques for which polynomial com-

[1]The notion of real throughput refers to the throughput of the system modelled, which can be calculated by exact analysis or simulation.

plexity algorithms exist (but the ILPP), so they offer a good trade-off between accuracy and computational complexity. They can be applied to any real-life application whose Petri net model matches the net class considered in this work, i.e., process Petri net [Tricas, 2003]. This kind of real-life applications can be found in manufacturing, logistics or dissimilar systems such as web services. In general, such applications represent real-life problems where resources are shared.

1.2 State of the Art

This section discusses the state-of-the-art by introducing interesting works of the literature related to the topics covered by this thesis, namely: software engineering considering security as a requirement, Fault Tolerance design, security-performance trade-off analysis, performability/performance analysis, and resource optimisation analysis. These works are introduced by briefly describing and comparing them with our work.

Security as a requirement. Bringing security into the design stage of development has been pointed out by many works in the literature, and from very different perspectives. Some works try to bring security to requirements analysis, capturing so security in the initial stage [Haley et al., 2006,Yskout et al., 2008,Wolter and Meinel, 2010]. Haley et al. give in [Haley et al., 2006] a powerful framework to, first, capture security goals of the system, and then check that such goals are fulfilled. In [Yskout et al., 2008], Yskout et al. propose an automatic transformation from a security requirements model to an architectural model. They use UML as target model. Wolter and Meinel provide in [Wolter and Meinel, 2010] an extension for Business Processing Modelling (BPM) giving some graphical annotations to express authorisations.

Other works capture security requirements and preserve them into architectural designs [Schmidt and Wentzlaff, 2006, Abi-Antoun and Barnes, 2010, Heyman et al., 2011]. Schmidt and Wentzlaff propose in [Schmidt and Wentzlaff, 2006] a Jackson's problem frame approach to map usability and security characteristics into architectural design artifacts. In [Abi-Antoun and Barnes, 2010], Abi-Antoun and Barnes present SECORIA, a method to analyse an architecture to seek information flow vulnerabilities and the conformance level with regard to specification in execution time. Heyman et al. give in [Heyman et al., 2011] an operational framework to develop at the same time security and architectural design artifacts.

Aspect-oriented modelling approach is used in [Georg et al., 2010, Braga, 2011] as a way of expressing security requirements. Georg et al. propose in [Georg et al., 2010] an Aspect-Oriented Risk-Driven Development (AORDD) where UML-Sequence Diagrams are transformed to Alloy language, then it is able to verify by logic checkers, and to Bayesian Belief Networks (BBFs) [Heckerman, 1995] allowing different analysis trade-offs.

Other works propose new design framework methodologies to integrate security as a NFP

into the system's design [Mouratidis et al., 2003,Islam et al., 2011,Khan, 2011]. Mouratidis et al. introduce in [Mouratidis et al., 2003] a methodology based on Tropos, which brings security to system design through a hierarchical approximation. In [Islam et al., 2011], Islam et al. propose a framework to deal with security legal regulations within Tropos methodology while Khan claims in [Khan, 2011] for a prescriptive framework to add security from the beginning of the system/software design, so saving costs and without affecting to backward system/software compatibility. Microsoft also proposes a secure development life-cycle with Security Development Lifecycle (SDL) [Microsoft, 2010], assuring security practices in each one of the development phases (based on classic spiral model) by reducing software maintenance costs and increasing software reliability and security against bug-coding.

There exist some other works in the literature which propose security patterns, such as [Fernández, 2004, Halkidis et al., 2008]. Fernández proposes in [Fernández, 2004] a methodology for building secure software based on UML, patterns and constraints expressed by Object Constraint Language (OCL) [OMG, 2010]. In [Halkidis et al., 2008], Haldikis et al. estimate the resilience of a bunch of security patterns to STRIDE (Spoofing Identity, Tampering with Data, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) attacks.

Some other works make use of formal methods [Schneider, 2000, Horvath and Dörges, 2008, Patzina et al., 2010]. Schneider proposes in [Schneider, 2000] a security automata for expressing security policies. Horvath and Dörges model in [Horvath and Dörges, 2008] security patterns using High-Level Petri nets (HLPNs), while Patzina et al. introduce in [Patzina et al., 2010] a Petri nets model called Monitor Petri nets, which enable the modelling of use cases and misuse cases.

Other works propose light-weight UML extensions for considering security into UML designs [Houmb and Hansen, 2003, Hussein and Zulkernine, 2006, Cirit and Buzluca, 2009]. In [Houmb and Hansen, 2003] a UML profile (SecurityAssessmentUML) for model-based security assessments is introduced. Mainly focused on risk identification and risk analysis in a security assessment, it allows to analyse the frequency of attacks using fault trees obtained from UML activity diagrams. A UML profile (called UMLintr) for specifying intrusions is described in [Hussein and Zulkernine, 2006]. UMLintr aims at specifying intrusion scenarios in UML diagrams to make much easier the communication of customers with software designers. However, this profile does not specify properties of distributed attacks and it is very focused on intrusion domain. A recent work in [Cirit and Buzluca, 2009] proposes a UML profile for specification of Role-Based Access Control (RBAC). This UML profile allows to graphically model the access control specifications in the domain problem and uses Object Constraint Language (OCL) to validate the well-formedness and meaning of information models against the RBAC.

Other works are focused on UML profiles for business processes or grid computing, such as [Rodríguez et al., 2006, Trujillo et al., 2009, Rosado et al., 2010b] . In [Rodríguez et al., 2006], Rodríguez et al. propose a UML profile that increases the

expressive ability of activity diagrams by incorporating security requirements into the business process modelling. In [Trujillo et al., 2009], UML profile is proposed for defining security requirements for Data Warehouses (DW) at the business level, then taking security issues into account and enforcing them from early DW design phases. UML extensions have been also proposed in the mobile grid system domain. A UML profile for building use case diagrams in the mobile Grid context called GridUCSec-Profile is introduced in [Rosado et al., 2010b]. This profile allows to analyse the system's security requirements from the early stages of development and to make decisions in the design phase about which security mechanisms should be used. The GridUCSec-Profile is used also in [Rosado et al., 2010a], where a methodology is proposed to analyse, design and construct a Secure Mobile Grid System.

Among other works using UML for expressing security issues, and by more closeness to our approach, it is worth mentioning [Lodderstedt et al., 2002, Jürjens, 2002, Goudalo and Seret, 2008]. SecureUML [Lodderstedt et al., 2002] allows to build secure distributed systems;in particular, it enables the specification of RBAC-based access control requirements together with several authorisation constraints. SecureUML also provides a model transformation to standard UML/OCL, then obtaining a secure system model where OCL constraints can be checked. Besides, it supports code generation from the secure system model. UMLsec [Jürjens, 2002] allows to specify security relevant information during development of security-critical systems. It considers RBAC as access control policy, like SecureUML, and also provides tool-support for formal security verification. Goudalo and Seret claim in [Goudalo and Seret, 2008] a UML extension for security as a real solution, and propose a set of stereotypes to deal with confidentiality into information systems. A recent work [Thapa et al., 2010] combines UMLsec and MARTE profiles, allowing to address both security and timing properties together in a UML model. The combined metamodel is then transformed to a USE (UML-based Specification Environment) specification to be verified using USE tools.

This thesis proposes in Chapter 3 a UML extension through profiling. Although SecureUML provides a comprehensive framework for the specification and verification of access control policies, it does not cover other security aspects, like the ones considered in our approach (i.e., non-repudiation, authenticity, availability). Besides, `SecAM` strives for gaining formal models for analysis and verification. In contrast to the UML extension given in [Goudalo and Seret, 2008], our approach aims to be applicable in more domains than just information systems, and covers more security attributes than confidentiality issue. Unlike UMLsec [Jürjens, 2002], `SecAM` has been built by following well-known UML profile rules given by Lagarde [Lagarde et al., 2007] and Selic [Selic, 2007]. This UML-profile compliance should make easier the task of adding `SecAM` to existing UML profile-based case tools. In contrast to UMLsec, `SecAM` defines security attributes from a more abstract way and not proposes ad-hoc stereotype solutions. Besides, `SecAM` enables security properties to be expressed as parameters for further quantitative/qualitative analysis.

Fault Tolerance design. Regarding Fault-Tolerant (FT) techniques applied at software architectural level also several works can be found [Nguyen-Tuong and Grimshaw, 1999, Bondavalli et al., 2001, Majzik et al., 2003, Rugina et al., 2007, Harrison and Avgeriou, 2008].

In [Harrison and Avgeriou, 2008] Harrison and Avgeriou studied how several FT techniques (FTTs) can be carried out as best-known architectural patterns. By the use of architectural patterns they aim to directly create software architectures satisfying quality attributes. Nguyen-Tuong and Grimshaw presented in [Nguyen-Tuong and Grimshaw, 1999] a reflective model, called *Reflective Graph & Event* (RGE), which is applied for making failure-resistant applications. Using this reflective model they are able to express FT algorithms as reusable components allowing composition with user applications. Rugina et al. propose in [Rugina et al., 2007] an approach for system dependability modelling using AADL (Architecture Analysis and Design Language), being the design model transformed into Generalised Stochastic Petri nets (GSPNs). This approach was applied to an Air Traffic Control System. Bondavalli et al. [Bondavalli et al., 2001, Majzik et al., 2003] have a vast work in the area of translating UML diagrams into dependability models, having also used Petri nets as a target model in some of these works. Their proposal of translation needs an intermediate model as a first step.

This thesis introduces a FT model library (see Chapter 4), where we have explored the idea of combining models that represent FT techniques and software behavioural designs. The combined model is useful for dependability assessment, as it is shown through an example. In our honest opinion, the key point that we are proposing is to gain a "library" of UML models representing FT techniques ready to use in critical designs.

Security-performance trade-off analysis. 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 [Blaze et al., 2002, Gupta et al., 2002, Kant et al., 2000].

Security properties are often considered in trade-off with other features, for instance in [Menascé, 2003] 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 [Menascé, 2003] 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 [Juric et al., 2006], 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 [Juric et al., 2006] 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 [Cilardo et al., 2007] where security applications are planned and implemented with embedded security strategies, and subsequently monitored. Our approach differs from [Cilardo et al., 2007] 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 [Woodside et al., 2009]. 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 [Woodside et al., 2009] is only performed on the SSL protocol.

The lack of a model-based approach to this problem is the major motivation behind our work, as it shown by the proposals given in Chapter 4 and 5. We aim 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, the models that we propose aim at informing software designers about the performance of different design solutions for critical systems.

Performability analysis. Other interesting parameter while designing critical system is *performability*. Performability [Meyer, 1982] evaluates the performance (throughput) and the reliability of systems whose provided services may suffer some degradation due to errors and failures. Many studies evaluate the performability of a FT system through analytical models, usually represented as Markov processes [Goŝeva-Popstojanova and Trivedi, 2001, Gokhale et al., 2004]. These studies consider the FT systems modelled ad-hoc, and they do not provide any solution to mitigate the impact of activation of faults into the FT system. An evaluation of performability using Petri net-based models is presented in [Sanders and Meyer, 1991,Bobbio, 1989]. Stochastic Activity Networks (SANs) are used in [Sanders and Meyer, 1991], associating reward rates directly with the markings of designated places and reward impulses with the completion of activities. Such an idea is extended for GSPNs by Bobbio in [Bobbio, 1989].

A more recent approach is given by Reussner et al. in [Reussner et al., 2003], where a compositional approach is presented using Markov chains as modelling formalism. Other works [Abdelmoez et al., 2004,Cortellessa and Grassi, 2007] in the literature study the impact of error propagation on reliability, also focused on component-based systems.

This thesis proposes in Chapter 4 a compositional PN models for FT techniques. These compositional PN models allow us to make performability (i.e., performance under failure conditions) analysis easier when FT parameters change. Thus, these FT models can be useful for evaluating different FT approaches in the same system model. Finally, Chapter 5 introduces a model-based methodology for performance prediction of critical systems where

FT techniques and other security mechanisms are incorporated.

Recall that FT systems (i.e., systems that incorporate FT techniques) can be naturally modelled as Discrete Event Systems (DES) where resources are shared, also called Resource Allocation Systems (RAS) [Colom, 2003]. In this dissertation, we focus on FT systems using shared resources modelled as Petri nets (PNs) – more precisely, as process Petri nets [Tricas, 2003].

Performance analysis. Performance estimation using PNs is a topic which has been broadly studied. Some works are concerned to the exact computation of analytical measures of the performance [Ajmone Marsan et al., 1995], while others overcome the state explosion problem providing performance bounds [Ramchandani, 1974, Chiola et al., 1993, Campos et al., 1992, Liu, 1995]. The use of performance bounds, on which our approach is based, avoids the necessity of calculating the whole state space. The advantage of using performance bound computation is the reduced computing time, but its drawback is the difficulty to assess how accurate the computed bound is with respect to the real system performance.

One of the first works on performance bounds computation is [Ramchandani, 1974], where strongly connected Marked Graphs (MGs) with deterministic timing are considered, and the reachability of the computation bound is proved. Some other works that compute performance bounds use linear programming techniques [Chiola et al., 1993, Campos et al., 1992], in the same way that our approach. These bounds are frequently calculated by using the first order moment (i.e., the mean) of the distributions associated to the firing delay. In [Liu, 1995], the second order moment is used to obtain a sharper (i.e., more accurate) performance bound.

Other works provide bounds for queueing systems instead of PN models like our approach does, e.g., [Haddad et al., 2005, Casale et al., 2008, Osogami and Raymond, 2010]. Haddad et al. give in [Haddad et al., 2005] space complexity upper and lower bounds for Stochastic Petri nets with product-form solution. In [Casale et al., 2008], Casale et al. propose performance upper and lower bounds for closed queueing networks with general independent and non-renewal services. They use linear programming techniques on the queue activity probabilities. Osogami and Raymond provide in [Osogami and Raymond, 2010] upper and lower bounds on the tail distribution of the transient waiting time for a general independent services queue. They use the two first moments of the service time and interarrival time, and solve it through semidefinite programming (SDP), a convex optimisation technique used for optimisation of complex systems. On the contrary, our approach introduced in Chapter 6 uses first order moment and linear programming techniques.

This thesis proposes in Chapter 6 an improvement of upper bound computation for the particular case of MGs and of process Petri nets by using regrowing techniques (that is, by adding more components to the initial bottleneck of the net).

We have also applied our approach for getting improved performance bound to the domain of scientific workflow, as it is summarised in Chapter 10. Petri nets

and their extensions have been widely used for the specification, analysis and implementation of scientific workflows [van der Aalst and van Hee, 2004] (e.g., GWorkflowDL [Pellegrini et al., 2008, Vossberg et al., 2008], Grid-Flow [Guan et al., 2006] or FlowManager [Aversano et al., 2002]). In Chapter 10, we propose the use of ordinary PNs for deriving performance models of pure graph-based workflows.

An analysis of the overhead for scientific workflows in Grid environments was given by Nerieri et al. in [Nerieri et al., 2006]. The analysis includes both load imbalance and data movement, with these being identified as the most significant sources of overhead. As discussed in this paper, Park & Humphrey [Park and Humphrey, 2008] already analysed the problem of load imbalance and data throttling for scientific workflows. They proposed a process envelope based framework for throttling data transfers. Nonetheless, they do not provide any analysis method in order to automatically obtain such data-throttling values. The proposal that we describe in Chapter 10 gives a method that can automatically derive (sub-optimal) values for them.

Lastly, it is worth mentioning the work in [Aalst et al., 2002], where a Petri net structural analysis is undertaken for business workflows. A specific class of Petri nets, WF-nets, is used and tailored towards workflow analysis. WF-nets can model workflows with different kind of control operations such as sequence, choice, synchroniser, fork or merge. The types of structural analysis that can be undertaken includes correctness, deadlock analysis or liveness.

Resource optimisation analysis. Finally, resource optimisation and its usage have been already studied for workflow Petri nets (WF-nets) [Li et al., 2004] or some variants [Wang and Zeng, 2008, Hee et al., 2001, Chen et al., 2008]. The underlying PN model of WF-nets are free choice nets (FCNs). However, the kind of systems we are considering cannot be modelled through FCNs: in the systems we consider, it may exist conflicts in the resources acquirement synchronisation, which is not allowed in FCNs. Li et al. propose in [Li et al., 2004] an approach to estimate the resource availability by using Continuous Time Markov Chains (CTMCs) and compute the turnaround time (i.e., the shortest response time) by performing reduction operations on the original WF-net. This performance analysis has an exponential complexity in the worst case, whilst our approach has a polynomial complexity due to the use of linear programming (LP) techniques. Resource usage could be computed in our approach by calculating the average marking of resource places in the PN system. Wang and Zeng provide in [Wang and Zeng, 2008] a method for computing the best implementation case for a workflow represented by a PN model, based on the reachability graph. Such a method, however, can suffer scalability problems if the workflow size is large. Van Hee et al. give in [Hee et al., 2001] an algorithm to compute optimal resource allocation in stochastic WF-nets. Such an algorithm suffers from scalability problems because its complexity depends on the number of resources. On the contrary, our approach only depends on the net structure, no matters the number of resources in the system. Therefore, for large systems with great number of resources our approach is

more tractable than the one in [Hee et al., 2001]. Chen et al. propose in [Chen et al., 2008] a new PN model, called Resource Assignment Petri Net (RAPN), to define how resources are shared and assigned among different and concurrent project activities. The computation of the execution project time considers deterministic timing and, unlike our approach, such a new PN model is not able to model activities that utilise and release the same resource intermittently.

This thesis proposes, in Chapter 7, several approaches to minimise the cost of compensation needed for maintaining a given throughput in a FT system.

Another important issue related to resource sharing is deadlock prevention. The common use of system resources in concurrent systems may lead to deadlock problems, i.e., a process waits for the evolution of other process/es, while the latter is/are also waiting for the former to evolve. In order to deal with such problems, there exist deadlock prevention or avoidance policies which may be applied for assuring the liveness property and therefore to avoid deadlocks [Colom et al., 1990,Tricas et al., 2000,Ezpeleta and Valk, 2006,Wu et al., 2008, Lopez-Grao and Colom, 2011, Hu et al., 2012, Li et al., 2012].

1.3 Outline

The balance of this dissertation is as follows. Chapter 2 introduces the preliminary concepts needed to follow the rest of the dissertation, such as Petri nets (PNs), UML diagrams and Fault Tolerance. The rest of this dissertation has been divided in five main parts: *Design of Critical Systems*, *Performance Analysis*, *Applications*, *Tool Support* and *Conclusions*.

The first part of this dissertation is composed of Chapters 3 to 5 and it is mainly devoted to the contributions related to the specification of security and the design of critical systems. Namely, Chapter 3 introduces a UML profile focused on security, called `SecAM`, which has served for expressing security properties into UML designs in several publications. Then, Chapter 4 introduces a set of Fault-Tolerant (FT) techniques, expressed in UML models and directly into PN models, which allows to make easier the addition of these FT techniques into software designs. Lastly, Chapter 5 is devoted to performance prediction of critical systems by means of a model-based methodology which combine Fault-Tolerant Techniques (FTTs), such as recovery procedures, and/or Security Mechanisms (SMs).

The second part of this dissertation is related to the contributions on performance analysis theory. Chapter 6 introduces a bunch of strategies for the upper throughput bound computation on Petri nets. Recall that we are dealing with critical systems that incorporate FT techniques to deal with any unexpected situation, and these additions may have an impact on system performance. Thus, Chapter 7 introduces a set of algorithms to compensate the throughput degradation in critical systems.

The third part of this dissertation is devoted to applications of the theory introduced in previous chapters. Namely, Chapter 8 considers the design of a Secure Database System (SDBS) where the approaches presented in Chapters 6 and 7 are tested. Lastly, Chapter 10

addresses the application of approaches presented in Chapter 6 to other scientific domain, more precisely, scientific workflows. Chapter 10 also introduces a quantitative metric for workflows, and a data-throttling strategy for improving the use of bandwidth and input buffers of workflow tasks.

The forth part of this dissertation is related to tool support. Chapter 11 introduces `PeabraiN`, a tool developed as a side product during this dissertation that implements some of the approaches presented in Chapters 6 and 7.

Finally, Chapter 12 in the fifth part summarises the major contributions of this dissertation and establishes the current open problems.

Chapter 2

Preliminary Concepts

This chapter introduces some basic concepts that are needed to follow the rest of this dissertation. We start defining Petri nets (PNs) in the untimed and timed framework, and introducing a special class of PNs – more precisely, Process Petri net (PPN), which is a basis for our approach – and related concepts, such as upper throughput bounds. Secondly, the Unified Modelling Language (UML) is addressed by describing the semantics of the diagrams that we use in this dissertation. Lastly, the concepts related to Fault Tolerance are introduced.

2.1 Petri Nets

This section introduces some basic concepts regarding the class of Petri nets (PNs) we are considering in this dissertation. Firstly, we define process Petri nets in the untimed framework. Then, timed Petri net systems are defined. In the following, the reader is assumed to be familiar with Petri nets (see [Murata, 1989] for a gentle introduction).

2.1.1 Untimed Petri Nets

Definition 1 *A Petri net [Murata, 1989] (PN) is a 4–tuple* $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$*, where:*

- P *and* T *are disjoint non-empty sets of* places *and* transitions *(*$|P| = n$*,* $|T| = m$*) and*
- **Pre** *(***Post***) are the pre–(post–)incidence non-negative integer matrices of size* $|P| \times |T|$*.*

The *pre-* and *post-set* of a node $v \in P \cup T$ are respectively defined as ${}^\bullet v = \{u \in P \cup T | (u, v) \in F\}$ and $v^\bullet = \{u \in P \cup T | (v, u) \in F\}$, where $F \subseteq (P \times T) \cup (T \times P)$ is the set of directed arcs. A Petri net is said to be *self-loop free* if $\forall p \in P, t \in T\ t \in {}^\bullet p$ implies

$t \notin p^\bullet$. *Ordinary* nets are Petri nets whose arcs have weight 1. The *incidence matrix* of a Petri net is defined as $\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$.

A vector $\mathbf{m} \in \mathbb{Z}_{\geq 0}^{|P|}$ which assigns a non-negative integer to each place is called *marking vector* or *marking*.

Definition 2 *A* Petri net system*, or* marked Petri net $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$*, is a Petri net* $\mathcal{N}$ *with an* initial marking $\mathbf{m_0}$*.*

A transition $t \in T$ is *enabled* at marking $\mathbf{m}$ if $\mathbf{m} \geq \mathbf{Pre}(\cdot, t)$, where $\mathbf{Pre}(\cdot, t)$ is the column of $\mathbf{Pre}$ corresponding to transition t. A transition t enabled at $\mathbf{m}$ can *fire* yielding a new marking $\mathbf{m'} = \mathbf{m} + \mathbf{C}(\cdot, t)$ (*reached* marking). This is denoted by $\mathbf{m} \xrightarrow{t} \mathbf{m'}$. A sequence of transitions $\sigma = \{t_i\}_{i=1}^{n}$ is a *firing sequence* in $\mathcal{S}$ if there exists a sequence of markings such that $\mathbf{m_0} \xrightarrow{t_1} \mathbf{m}_1 \xrightarrow{t_2} \mathbf{m}_2 \ldots \xrightarrow{t_n} \mathbf{m}_n$. In this case, marking $\mathbf{m}_n$ is said to be *reachable* from $\mathbf{m_0}$ by firing σ, and this is denoted by $\mathbf{m}_0 \xrightarrow{\sigma} \mathbf{m}_n$. The *firing count vector* $\boldsymbol{\sigma} \in \mathbb{Z}_{\geq 0}^{|T|}$ of the firable sequence σ is a vector such that $\boldsymbol{\sigma}(t)$ represents the number of occurrences of $t \in T$ in σ. If $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$, then we can write in vector form $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}$, which is referred to as the *linear* (or *fundamental*) *state equation* of the net.

The set of markings *reachable* from $\mathbf{m_0}$ in $\mathcal{N}$ is denoted as $RS(\mathcal{N}, \mathbf{m_0})$ and is called the *reachability set*.

A place $p \in P$ is $k-$*bounded* if $\forall \mathbf{m} \in RS(\mathcal{N}, \mathbf{m_0}), \mathbf{m}(p) \leq k$. A net system $\mathcal{S}$ is *k-bounded* if each place is k-bounded. A net system is *bounded* if there exists some k for which it is k-bounded. A net $\mathcal{N}$ is *structurally bounded* if it is bounded no matter which $\mathbf{m_0}$ is the initial marking.

Two transitions t, t' are said to be in *structural conflict* if they share, at least, one input place, i.e., ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$. Two transitions t, t' are in *equal conflict* if $\mathbf{Pre}(\cdot, t) = \mathbf{Pre}(\cdot, t') \neq \mathbf{0}$, where $\mathbf{0}$ is a vector with all entries equal to zero.

A transition t is *live* if, for each marking $\mathbf{m} \in RS(\mathcal{N}, \mathbf{m_0})$ there exists a marking $\mathbf{m'}$ reachable from $\mathbf{m}$ where transition t is enabled. A marked Petri net $\mathcal{S}$ is *live* when every transition is live. Hereafter, we assume that $\mathcal{S}s$ we work with are live.

A *p-semiflow* is a non-negative integer vector $\mathbf{y} \geq \mathbf{0}$ such that it is a left anuller of the net's incidence matrix, $\mathbf{y}^\top \cdot \mathbf{C} = 0$. In the sequel, we omit the transpose symbol in the matrices and vectors for clarity. A p-semiflow implies a token conservation law independent from any firing of transitions. A *t-semiflow* is a non-negative integer vector $\mathbf{x} \geq \mathbf{0}$ such that is a right anuller of the net's incidence matrix, $\mathbf{C} \cdot \mathbf{x} = 0$. A p- (or t-)semiflow $\mathbf{v}$ is *minimal* when its support, $\|\mathbf{v}\| = \{i | \mathbf{v}(i) \neq 0\}$, is not a proper superset of the support of any other p- (or t-)semiflow, and the greatest common divisor of its elements is one. A Petri net is said to be *conservative* (*consistent*) if there exists a p-semiflow (t-semiflow) which contains all places (transitions) in its support.

A Petri net is said to be *strongly connected* if there is a directed path joining any pair of nodes of the net structure. A *state machine* is a particular type of ordinary Petri nets where each transition has exactly one input arc and exactly one output arc. More formally:

Definition 3 [Murata, 1989] *A state machine is a subclass of Petri nets such that* $\forall t \in T, |t^\bullet| = |^\bullet t| = 1$.

Marked graphs (MGs) are a subclass of ordinary Petri nets that are characterised by the fact that each place has exactly one input and exactly one output arc. More formally:

Definition 4 [Murata, 1989] *A marked graph (MG) is an ordinary Petri net such that* $\forall p \in P, |^\bullet p| = |p^\bullet| = 1$.

In this dissertation, we deal with Petri nets that model systems where resources are shared. Examples of this kind of systems can be found in manufacturing, logistics or web services systems. In general, these systems represent real-life problems where some items are processed and require the use of different resources (which are shared) during its processing. These systems can be naturally modelled in terms of process Petri nets, a subclass of Petri net whose inner structure is a strongly connected state machine. More formally:

Definition 5 [Tricas, 2003] *A process Petri net (PPN) is a strongly connected self–loop free Petri net* $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$ *where:*

1. $P = P_0 \cup P_S \cup P_R$ *is a partition such that* $P_0 = \{p_0\}$ *is the* process-idle place, $P_S \neq \emptyset, P_S \cap P_0 = \emptyset, P_S \cap P_R = \emptyset$, P_S *is the set of* process-activity places *and* $P_R = \{r_1, \ldots, r_n\},\ n > 0, P_R \cap P_0 = \emptyset$ *is the set of* resources places*;*
2. *The subnet* $\mathcal{N}' = \langle P \setminus P_R, T, \mathbf{Pre}, \mathbf{Post} \rangle$ *is a strongly connected state machine, such that every cycle contains* p_0.
3. *For each* $r \in P_R$*, there exist a unique minimal p-semiflow associated to* r, $\mathbf{y}_r \in \mathbb{N}^{|P|}$, *fulfilling:* $\|\mathbf{y}_r\| \cap P_R = \{r\}, \|\mathbf{y}_r\| \cap P_S \neq \emptyset, \|\mathbf{y}_r\| \cap P_0 = \emptyset$ *and* $\mathbf{y}_r(r) = 1$. *This establishes how each resource is reused, that is, they cannot be created nor destroyed.*
4. $P_S = \bigcup_{r \in P_R} (\|\mathbf{y}_r\| \setminus \{r\})$.

Definition 5 implies that $PPNs$ are conservative and consistent. Intuitively, Definition 5 establishes a kind of nets where:

a) there is a process using different shared resources;

b) every place in the net is covered by some p-semiflow and uses at least one resource;

c) the number of instances of each resource remains constant; and

d) resources cannot change its type.

Let $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$ be a PPN. A vector $\mathbf{m_0} \in \mathbb{Z}_{\geq 0}^{|P|}$ is called *acceptable initial marking* [Tricas, 2003] of $\mathcal{N}$ if:

1) $\mathbf{m_0}(p) \geq 1,\ p \in P_0$;

2) $\mathbf{m_0}(p) = 0,\ \forall p \in P_S$; and

3) $\mathbf{m_0}(r) \geq \mathbf{y}_r(r),\ \forall r \in P_R$, where $\mathbf{m_0}(r)$ is the *capacity*, i.e., number of items, of the resource r and $\mathbf{y}_r$ is the unique minimal p-semiflow associated to r.

Definition 6 *A* process Petri net system*, or* marked process Petri net $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$*, is a process Petri net* $\mathcal{N}$ *with an* acceptable initial marking $\mathbf{m_0}$.

2.1.2 Timed Petri Nets

In order to be able to use Petri nets for systems performance evaluation, the inclusion of the notion of time must be considered. There are two ways of introducing the notion of time in Petri nets, either in places or transitions. Since transitions are representing the actions of a system, which have associated some duration, we associate such a duration to the firing delay of transitions [Ramchandani, 1974]. Besides, we consider that the firing delays of transitions follow an exponential distribution functions.

A Petri net model where a set of exponential rates is considered (one for each transition in the model) is called a *Stochastic Petri net* (SPN) model [Florin and Natkin, 1985, Ajmone Marsan et al., 1995]. These rates characterise the probability distribution function of the transition delay, which follow an exponential distribution function and are obtained as the inverse of the mean. These rates are considered to be marking-independent, i.e., its values are constant.

In this dissertation, we consider that the average service time of a transition t can be zero, i.e., it fires in zero units of time. These transitions are called *immediate transitions*. Otherwise, transition t is a *timed transition*. The exponential transitions are graphically represented by a white box, whilst immediate transitions are black boxes. It will be assumed that all transitions in conflict are immediate. An immediate transition t in conflict will fire with probability $\frac{\mathbf{r}(t)}{\sum_{t' \in A} \mathbf{r}(t')}$, where A is the set of enabled immediate transitions in conflict and $\mathbf{r}(t) \in \mathbb{N}_{>0}$ is the routing rate associated to transition t. The firing of immediate transitions consumes no time. When a timed transition becomes enabled, it fires following an exponential distribution with mean $\delta(t)$. More formally, we will consider the following timed Petri net classes:

Definition 7 *A* Stochastic Petri Net *(SPN)* [Florin and Natkin, 1985] *system is a pair* $\langle \mathcal{S}, \delta, \mathbf{r} \rangle$ *where* $\mathcal{S} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0} \rangle$ *is a Petri net system,* $\delta \in \mathbb{R}_{\geq 0}^{|T|}$ *is a positive real function such that* $\delta(t)$ *is the mean of the exponential firing time distribution associated to transition* $t \in T$ *and* $\mathbf{r} \in \mathbb{N}_{>0}^{|T|}$ *is the vector of routing rates associated to transitions.*

Definition 8 *A* Stochastic Marked Graph *(SMG) is a Stochastic Petri net whose underlying Petri net is a Marked Graph.*

Definition 9 *A Stochastic Process Petri net (SPPN) system is a Stochastic Petri net system whose underlying Petri net is a Process Petri net.*

There exist different semantics for the firing of transitions, being *infinite* and *finite* server semantics the most frequently used. Given that infinite server semantics is more general (finite server semantics can be simulated by adding self-loop places), we will assume that the timed transitions work under infinite server semantics.

The average marking vector, $\overline{\mathbf{m}}$, in an ergodic [Ross, 1983] Petri net system is defined as [Florin and Natkin, 1989]:

$$\overline{\mathbf{m}}(p) \underset{AS}{=} \lim_{\tau\to\infty} \frac{1}{\tau} \int_0^\tau \mathbf{m}(p)_u du \tag{2.1}$$

where $\mathbf{m}(p)_u$ is the marking of place p at time u and the notation $\underset{AS}{=}$ means *equal almost surely*.

Similarly, the steady-state throughput, χ, in an ergodic Petri net is defined as [Florin and Natkin, 1989]:

$$\chi(t) \underset{AS}{=} \lim_{\tau\to\infty} \frac{\boldsymbol{\sigma}(t)_\tau}{\tau} \tag{2.2}$$

where $\boldsymbol{\sigma}(t)_\tau$ is the firing count of transition t at time τ.

By definition, all the places of a $SPPN$ are covered by p-semiflows, and therefore it is structurally bounded. In this work, we will assume that the $SPPN$ under study is a live and structurally bounded net with Freely Related T-semiflows (i.e., a FRT-net) [Campos and Silva, 1992]. It is known that the Markov process that describes the time evolution [Ajmone Marsan et al., 1995] of these nets is ergodic [Campos and Silva, 1992], i.e., when the observation period tends to infinite, the estimated values of average marking and steady-state throughput tend to a certain value, what implies the existence of the above limits.

The vector of visit ratios expresses the relative throughput of transitions in the steady state. The visit ratio $\mathbf{v}(t)$ of each transition $t \in T$ normalised for transition t_i , $\mathbf{v}^{t_i}(t)$, is expressed as follows:

$$\mathbf{v}^{t_i}(t) = \frac{\chi(t)}{\chi(t_i)} = \boldsymbol{\Gamma}(t_i) \cdot \chi(t),\ \forall t \in T \tag{2.3}$$

where $\boldsymbol{\Gamma}(t_i) = \dfrac{1}{\chi(t_i)}$ represents the *average inter-firing time* of transition t_i.

The visit ratios of two different transitions t, t' in equal conflict must be proportional to the corresponding routing rate $\mathbf{r}(t), \mathbf{r}(t')$ defining the conflict resolution condition $\mathbf{r}(t) \cdot \mathbf{v}^{t_i}(t') = \mathbf{r}(t') \cdot \mathbf{v}^{t_i}(t)$. This condition can be also written in vector form as:

$$\mathbf{R} \cdot \mathbf{v}^{t_i} = 0 \tag{2.4}$$

where $\mathbf{R}$ is a matrix containing as many rows as pairs of transitions in equal conflict.

In FRT-nets, the vector of visit ratios $\mathbf{v}$ exclusively depends on the structure of the net and on the routing rates [Campos and Silva, 1992]. The vector of visit ratios $\mathbf{v}$ normalised for transition t_i, $\mathbf{v}^{t_i}$, can be calculated by solving the following linear system of equations [Campos and Silva, 1992]:

$$\begin{aligned} \begin{pmatrix} \mathbf{C} \\ \mathbf{R} \end{pmatrix} \cdot \mathbf{v}^{t_i} &= 0 \\ \mathbf{v}^{t_i}(t_i) &= 1 \end{aligned} \tag{2.5}$$

2.2 The Unified Modelling Language

This section introduces briefly the main Unified Modelling Language (UML) diagrams that we use in this dissertation. Mainly, they are: UML Use Case (UML-UC) diagrams, UML Deployment Diagrams (UML-DD), UML State Machine (UML-SM) diagrams and UML Sequence Diagrams (UML-SD). In the following, the reader is assumed to be familiar with UML (see [OMG, 2005] for a gentle introduction).

UML, standard *de facto* as modelling language, is a powerful language which allows to represent from architectural to behavioural aspects of the systems. The focus in UML in this dissertation is motivated by the fact that UML is well-known by the system designers and they are very familiar with its use for designing.

The UML is a semi-formal language developed by the Object Management Group (OMG) to specify, visualise and document models of software and non-software systems. UML has gained widespread acceptance in the software development process for the specification of software systems based on the object-oriented paradigm.

UML provides several types of diagrams which allow to capture different aspects and views of the system. A UML model of a system consists of several diagrams which represent the functionality of the system, its static structure, the dynamic behaviour of each system component and the interactions among the system components.

UML defines twelve types of diagrams, divided into three main categories:

- **Static diagrams**, which are intended to model the structure (logical and architectural) of the system. They are: *class diagram*, *object diagram*, *component diagram* and *deployment diagram*.
- **Behavioural diagrams**, which are intended to describe system dynamics, and they are subdivided in: *sequence diagram*, *collaboration diagram*, *use case diagram*, *state-machine diagram* and *activity diagram*.
- **Diagrams to organise modules**, allowing to reduce complexity of the system. There exist *packages*, *subsystems* and *models*.

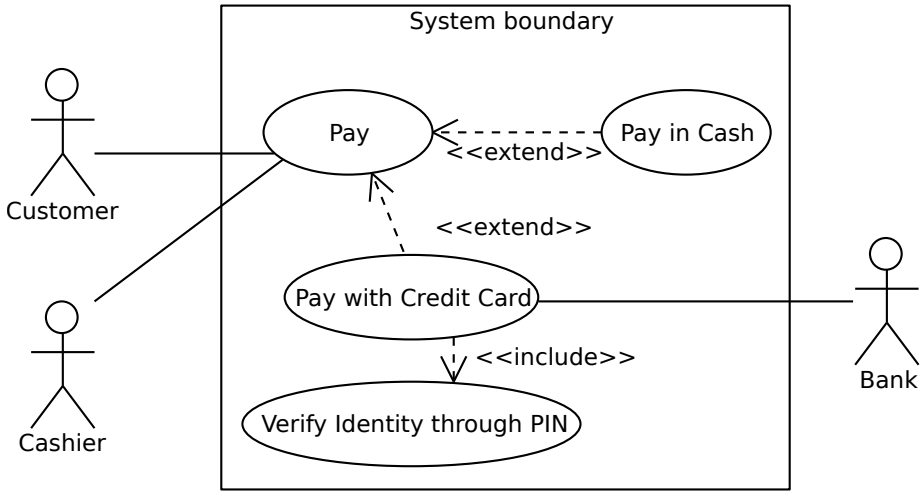


Figure 2.1: A UML Use Case (UML-UC) diagram of a payment system.

2.2.1 UML Use Case Diagrams

A UML Use Case (UML-UC) diagram is a UML behavioural diagram that describes the functionality of a system in a horizontal way. That is, rather than merely representing the details of individual features of a system, UML-UCs can be used to show all of its available functionality. The description of a UML-UC shows a list of steps, typically defining interactions between a role (known in UML as an *actor*, that can be a human or an external system) and a system, to achieve a goal. A UML-UC is represented by an oval with a label that describes it. A use case can extend another use case when the former is a special case behaviour of the latter. A use case can also include other use case when the former needs the latter to complete.

For instance, Figure 2.1 depicts a UML-UC diagram representing a paying system. There are three actors who interacts with the system, namely the `Customer`, the `Cashier` and the `Bank`. A customer pays for a service, while a cashier has to accept the customer's payment. The use case `Pay` is extended by two uses cases, `Pay in Cash`, which represents the case when the customer decides to pay for the services in cash, and by `Pay with Credit Card` when the customer decides to pay for with the credit card.

The latter use case, `Pay with Credit Card`, includes other use case, called `Verify Identity through PIN`, which needs to be completed in order to complete the payment through the credit card. Note that the `Pay with Credit Card` use case interacts as well with the `Bank` actor – the credit card issuer.

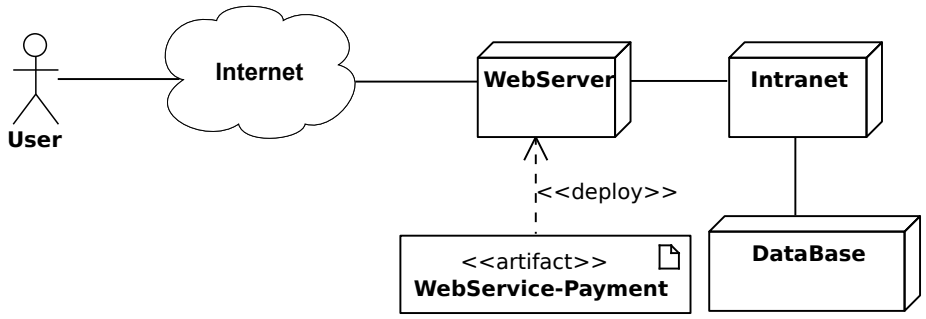


Figure 2.2: A UML Deployment Diagram (UML-DD) of a secure database system.

2.2.2 UML Deployment Diagrams

A UML Deployment Diagram (UML-DD) is a UML static diagram that is used to visualise the topology of the physical components of a system where the software components are deployed. They describe the static deployment view of a system and consist of nodes (represented as cubes), software components (represented as a rectangle and associated to a node through an arrow labelled `deploy`) and their relationships (lines). In other words, a UML-DD shows the hardware for a system (nodes), the software that is installed on that hardware (components, usually called `artifacts`), and the middleware used to connect the disparate machines to one another.

Figure 2.2 depicts a system representing an online payment service. There exits a `User`, represented as an actor, that interacts with a `WebServer` through the Internet. The web server deploys a software artifact called `WebService-Payment` which is in charge of interacting with the users and complete the payments. The web server is connected through an `Intranet` to a `DataBase` server that stores information about user's payments, accounts, etc.

2.2.3 UML State Machine Diagrams

A UML State-Machine Diagram (UML-SM) is a UML behavioural diagram that can be used for modelling discrete behaviour through finite state-transitions systems. A UML-SM is a directed graph in which nodes denote states and connectors denote state transition. A state is represented by a rounded rectangle labelled with a state name, while transitions are represented by arrows labelled with the triggered events followed (optionally) by executed actions. The initial transition originates from a solid circle and sets the default state where system begins. The final transition ends in a solid circle surrounded by an empty circle.

Figure 2.3 shows a UML-SM corresponding to the behaviour of a wireless computer keyboard. The initial state is the state from the initial transition, that is, `Initialise` state. A state can have a set of optional actions, such as *entry actions* that are executed upon

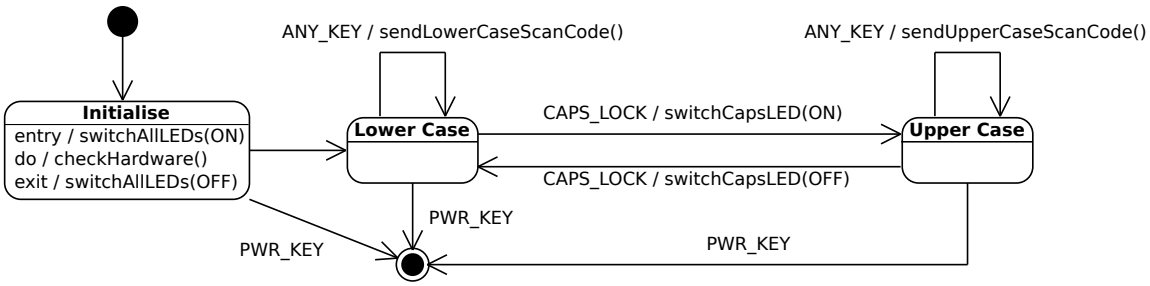


Figure 2.3: A UML State-Machine Diagram (UML-SM) of a computer keyboard.

entry to a state, *exit actions* that are executed upon exit from a state and *do actions* that represent activities to execute during staying into a state. For instance, the `Initialise` state has an entry action to switch on the keyboard LEDs called `switchAllLEDs(ON)`, a do activity that performs a self-checking on hardware called `checkHardware()` and as exit activity it switches off the keyboard LEDs by means of the activity `switchAllLEDs(OFF)`. Once the do and exit activities have been performed, the state of the system is led to the final state when event of pressing the power button of the keyboard occurs (event `PWR_KEY`), or to `Lower Case` state otherwise.

The system remains in `Lower Case` state until `CAPS_LOCK` key is pressed. When this happens, a triggered action is invoked (`switchCapsLED(ON)`) and the new state is `Upper Case`. In the same way, the system remains in the latter state until `CAPS_LOCK` key is pressed again, that triggers `switchCapsLED(OFF)` and the return to `Lower Case` state. When any other key is pressed, the appropriate scan code is sent to the operative system. As in the `Initialise` state, when `PWR_KEY` is pressed the system is led to the final state from both states `Lower Case` and `Upper Case`.

2.2.4 UML Sequence Diagrams

A UML Sequence Diagram (UML-SD) is a UML behavioural diagram that shows how processes into a system cooperate with one another and in what order. A UML-SD shows different processes or objects that live simultaneously as parallel vertical lines (called *lifelines*), and the messages exchanged between them, as solid horizontal arrows, in the order in which they occur. A message response is represented as a dashed horizontal arrow. All messages have a label to identify the invoked method between the processes or objects. This allows the specification of simple run-time scenarios in a graphical manner. A run-time scenario is called *interaction*, and it is enclosed on a a solid-outline rectangle. The left upper corner of an interaction contains the diagram's label beginning with letters "sd", which stands for Sequence Diagram.

Figure 2.4 shows a UML-SD corresponding to an interaction called *Consult Available Reports* between a financial analyst (user) with a financial reporting system. The user,

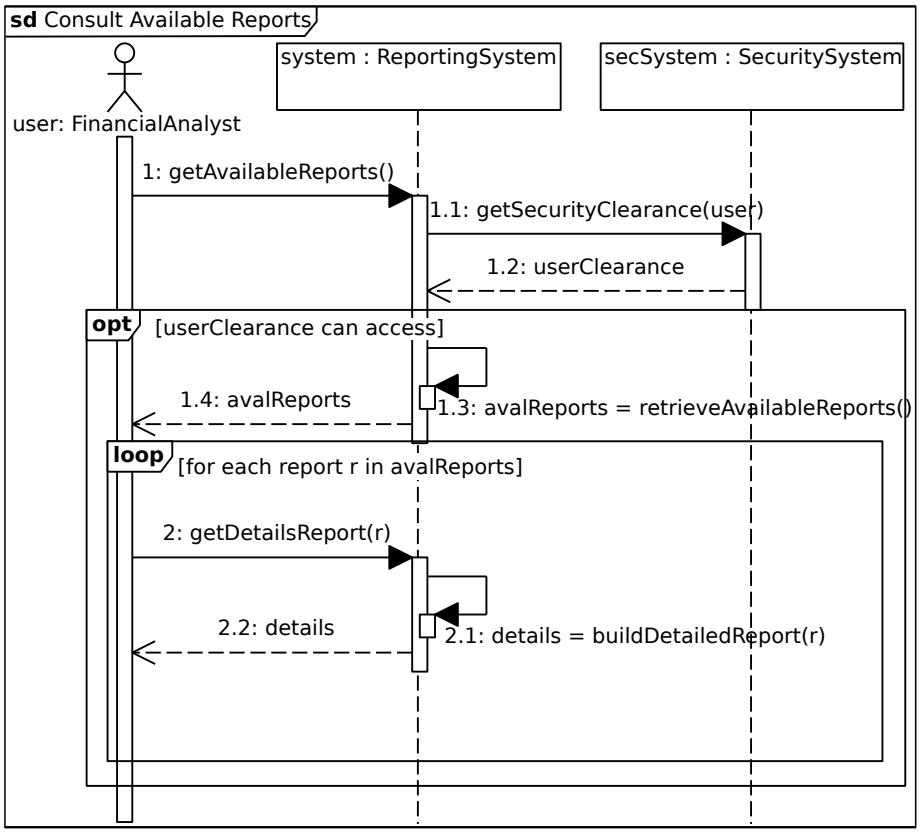


Figure 2.4: A UML Sequence Diagram (UML-SD) of a financial reporting system.

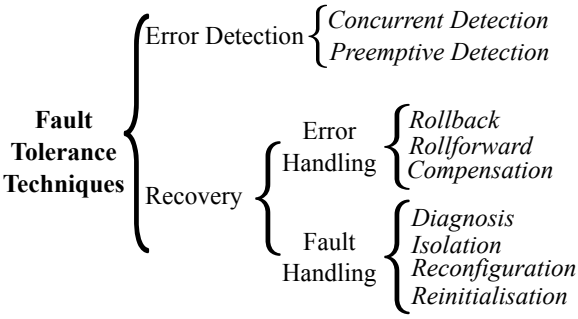


Figure 2.5: Phases involved on a Fault-Tolerant technique (adapted from [Avizienis et al., 2004]).

represented as an actor in the Figure 2.4 asks for the available reports to the reporting system. Then, the system interacts with the security system in order to verify the security clearance of the user. The security system returns (represented by a dashed arrow) the user clearance.

A UML-SD can incorporate combined fragments that model a sequence that, given a certain condition, will occur; otherwise, the sequence does not occur. For instance, Figure 2.4 has two combined fragments. The first one is an optional combined fragment element (keyword `opt`) that takes place when the user has enough clearance to access to available reports. When this occurs, the available reports are retrieved (self-message `avalReports = retrieveAvailableReports()`) are returned to the user. Finally, the user iterates for each one of the available reports by asking for detailed information. This iteration has been represented by means of a loop combined fragment element (keyword `loop`). Note that an optional combined fragment element is modelling a typical *if-then* case. A *if-then-else* case can also be modelled with an alternative combined fragment element, using keyword `alt` rather than `opt` and with a dashed horizontal line to separate each one of the cases that occur once the condition has been evaluated.

2.3 Fault Tolerance

Fault Tolerance (FT) aims at fault avoidance carrying out error detection and system recovery [Avizienis et al., 2004]. Figure 2.5 depicts the phases involved in a FT technique:

Error detection tries to identify the presence of an error in the system. It takes places either while the system is providing its services (concurrent), or when services are not being provided (preemptive). For instance, a hardware checking when the system boots up is a preemptive error detection technique.

Recovery techniques are aimed at handling possible errors and/or faults in the system

and leading it to a state without detected errors. Recovery techniques may have two steps: an *error handling* (optional step), which tries to eliminate the presence of an error in the system; and *fault handling* (mandatory step), which tries to avoid the reactivation of the detected fault.

There are three common techniques when dealing with a detected error:

- *rollback*, when the system is conducted to a previous saved state (i.e., prior to error occurrence) without detected errors;
- *rollforward*, when the system is conducted to a new state without detected errors (in this case, later to error occurrence); and
- *compensation*, when there is enough redundancy to mask the error in the erroneous state.

Unlike rollback or rollforward that happen on demand, compensation may happen on demand or systematically, independently of the presence (or absence) of an error. For instance, an example of a compensation handling technique triggered on demand is an exception handler mechanism. In this paper, we consider that error handling takes place on demand.

The fault handling techniques that can be carried out to prevent faults from reacting are:

- *diagnosis*, which records the origin (cause) of the error, locating where it happened and the type of error raised;
- *isolation*, which excludes (in a logical or physical way) faulty components from normal service delivery, so avoiding its participation in service delivery;
- *reconfiguration*, which reschedules service requests between non-failed components; and
- *reinitialisation*, which reconfigures the faulty system services by changing its configuration, stores this new configuration and reinitialises such affected services.

Part I

Design of Critical Systems

Chapter 3

A UML Profile for Security

In this chapter we summarise the main contributions of this dissertation related to the design of a UML extension focused on security [Rodríguez et al., 2010, Rodríguez and Merseguer, 2010, Rodríguez et al., 2012d]. This extension is performed through profiling. Briefly, a UML profile defines a set of stereotypes and tagged values that allow the expression of non-functional properties (such as performance, dependability or security), which are eventually attached to UML model elements extending its semantics.

3.1 Motivation

As we claimed in Section 1.1, there is a need to express security as a Non-Functional Property (NFP) into the design of systems. This need is even more important when the system is deployed in a harmful environment, where the system may be the victim of persistent and targeted attacks.

The approach we present in this chapter encompasses all the challenges that we identify to be addressed by new generation of development methods (see Section 1.1) relying on the Unified Modelling Language (UML) [OMG, 2005]. UML is the current standard modelling language, both for the industry and the software engineering research community. We propose a domain specific language, built as a UML profile, called `SecAM` (which stands for Security Analysis and Modelling) that integrates with the UML for the modelling and analysis of security.

Integration `SecAM` is used to annotated security issues in the requirements, design and deployment models of the UML. Therefore the security specification is *integrated* with the system functional specification, i.e., with the system models. The rest of the stages will use these models for development, then providing the necessary *integrated* view of security in all the stages of the life-cycle.

Analysis `SecAM` is conceived so that it allows to leverage the system models for security

analysis purposes. Sometimes the very same models can be directly used for analysis, sometimes they are transformed into formal models that allow analysis (e.g., Fault trees or Petri nets).

Unified view and vocabulary `SecAM` uses the standard Value Specification Language (VSL) [OMG, 2009]. VSL aims at the specification of NFPs, say performance, dependability and security. Through this language the different communities researching security gain a common vocabulary for expressing the security properties they manage.

Current SE techniques `SecAM` resorts to current software engineering trends. For example, the model-driven paradigm (MDD) to transform system models into formal models of analysis or the profiling mechanism, later explained.

In the following, we introduce the `SecAM` profile that integrates with the UML for the modelling and analysis of security. The `SecAM` profile was originally published in [Rodríguez et al., 2010] and used in several manuscripts such as [Rodríguez and Merseguer, 2010, Rodríguez et al., 2012d].

3.2 `SecAM` UML profile

The basis that support the `SecAM` profile are well-known, rely on standards and mean the current mainstream in software engineering with UML.

At this regard, SecAM relies on the "UML profile for Modelling and Analysis of Real-Time and Embedded systems" (MARTE) [OMG, 2009]. MARTE is an Object Management Group (OMG) standard defined using the "profiling" mechanism, a current innovative software engineering technique, as proposed by the fourth principle in the previous section.

Profiling was introduced by UML to indeed add new capabilities to the language. A UML profile is just an extension of the UML defined in terms of:

Stereotypes They are concepts in the target domain that will be added to the UML. For example, in `SecAM` we will add stereotypes for the security concepts, e.g., *attack* or *intrusion*.

Tags The attributes of the stereotypes. For example, for the *attack* stereotype, as we will show later, we will define attributes such as its `type`, `objective` or `location`.

Constraints They are formulae that apply to stereotypes and UML elements to extend their semantics.

Another important feature of MARTE is that it provides an analysis framework called Quantitative Analysis Model (GQAM). `SecAM` inherits GQAM, what confers it the analysis

capabilities that we defended in the previous section, second principle. The analysis in MARTE addresses the schedulability and performance NFPs, while in `SecAM` the security.

MARTE has been specialised for dependability modelling and analysis, leading to the definition of a Dependability Analysis and Modelling (DAM) profile [Bernardi et al., 2011]. MARTE and DAM together provide the basic bricks on which we build on our profile proposal in the security context.

Aside of MARTE, `SecAM` is not the first attempt to enlarge UML for the analysis of NFPs. The Dependability Analysis and Modelling (DAM) profile [Bernardi et al., 2011] also follows this technique, in particular to introduce the dependability[1] NFP in UML.

The relations between MARTE, `SecAM` and DAM are described in Figure 3.1(a). The VSL referred before is the language all they share and then what justifies the third principle in the Section 3.1.

`SecAM` relies on MARTE and DAM, as shown in Figure 3.1(a). The result is a common and powerful UML framework that can be used for the joint specification of different NFPs, concretely performance and schedulability (from MARTE), dependability (from DAM) and security (from `SecAM`). Like MARTE and DAM, the `SecAM` profile is organized in two main packages, namely the *SecAM_UML_Extensions*, that includes the set of stereotypes, and the *SecAM_Library*. The latter contains basic (typically enumeration types) and complex types, used to define the stereotype tags and composite security NFPs.

The `SecAM` stereotypes are divided in sub-packages: *Cryptographic*, *SecurityMechanisms*, *Resilience* and *AccessControl* as in Figure 3.1(b). In the following, we introduce each package first depicting the stereotypes and tagged-values, and giving some explanation over them. Then, a small example for putting on each package in practice is introduced.

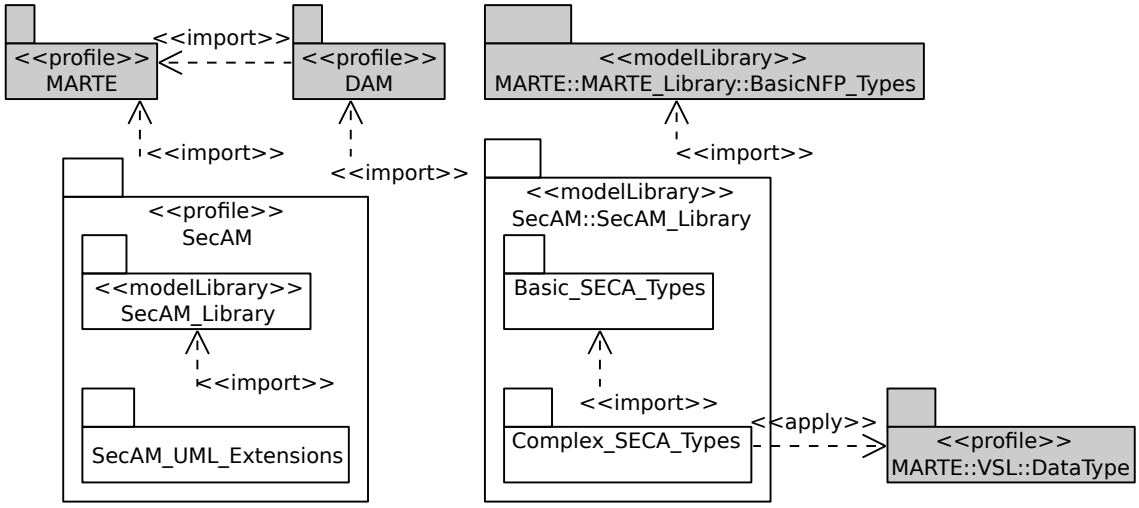
The rationale of this organisation has been to address different security issues, typically dealt by independent research communities. We have used a breadth-first approach to provide common basis for the specification of security in UML. Obviously, the profile is an open proposal, that can be refined to add new modelling capabilities of security in application domains.

The stereotypes in the first three sub-packages (cryptography, mechanisms and resilience) can be applied to behavioural diagrams, while the latter to structural diagrams. Each sub-package deals with a subset of well-known security attributes [Pfleeger and Pfleeger, 2006] (integrity, availability, confidentiality, authorisation, non-repudiation, authenticity) and, as shown in Table 3.1, the sub-packages overlap with respect to attribute coverage. These packages are largely explained in the sequel.

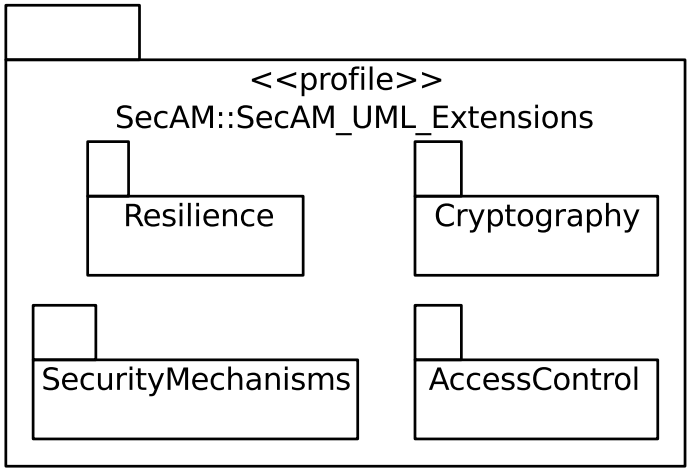
3.2.1 `SecAM::Resilience` package

The *Resilience* package, depicted in Figure 3.2, was initially proposed in [Rodríguez et al., 2010] to enable the specification in UML behavioural diagrams

[1] By dependability here we understand: availability, reliability, safety and maintainability.



(a)



(b)

Figure 3.1: (a) `SecAM` profile and library, (b) `SecAM` UML extensions (subpackages).

Security attributes	SecAM packages (P1)	(P2)	(P3)	(P4)
Integrity	√	√		√
Availability		√	√	
Confidentiality	√	√		√
Authorisation				√
Non-repudiation	√			
Authenticity	√			

(P1): `Cryptographic`; (P2): `SecurityMechanisms`
(P3): `Resilience`; (P4): `AccessControl`

Table 3.1: Security attributes and SecAM packages in which they are covered.

of attacks, vulnerabilities and intrusion concepts, and their causal relationships (i.e., the AVI chain) [Avizienis et al., 2004], as well as to support vulnerability stochastic analysis.

It contains two stereotypes, `SecaAttackGenerator` and `SecaStep`. They specialise the DAM stereotypes `DaFaultGenerator` and `DaStep`, respectively. Hence, by inheritance, they can be applied to all those UML behavioural model elements that can be stereotyped with the latters. For example, `SecaStep` can stereotype actions, activities, trigger events, transitions and states in UML State Machines diagrams, messages and fragments in UML Sequence Diagrams.

These two stereotypes have the attributes depicted in Figure 3.2 (left side), i.e., `attack` of `SecaAttackGenerator` and `vulnerability` and `intrusion` of `SecaStep`. The definition of the types of these attributes appears in Figure 3.2 (right side).

Herein, we add a new complex type to represent the concept of *coordinated attacks* (`SecaCoordAttack`). A coordinated attack allows attackers to avoid an intrusion detection by splitting a malicious attack pattern in several sub-patterns (`attacks` attribute). It can be classified (`type` attribute) as [Braynov, 2003]: a *cumulative* attack, where simultaneous attacks are initiated to overcome computer limitations; a *replicated attack*, where several attacks to replicated services occur to bring down the entire service structure; or a *mixed attack*, i.e., a combination of the previous ones. For probabilistic analysis purposes, we have characterized a coordinated attack by its occurrence probability (`occurrenceProb`), that is the joint probability of the occurrences of single attacks it coordinates.

Considering several sources [Barnum, 2008, Hansman and Hunt, 2005, Hussain et al., 2003] new attributes have been also added to the `Attack` class; i.e., `class`, `kind`, `objective` and `location`. The different classes of attacks (`ClassOfAttack`) are compliant to the taxonomy defined by Hansman and Hunt in [Hansman and Hunt, 2005], e.g., virus or worm, which define how an attack works.

On the other hand, an attack can be of different kind (`KindOfAttack`) [Barnum, 2008], depending on the method adopted by the attacker to succeed in the intent, e.g., injection or

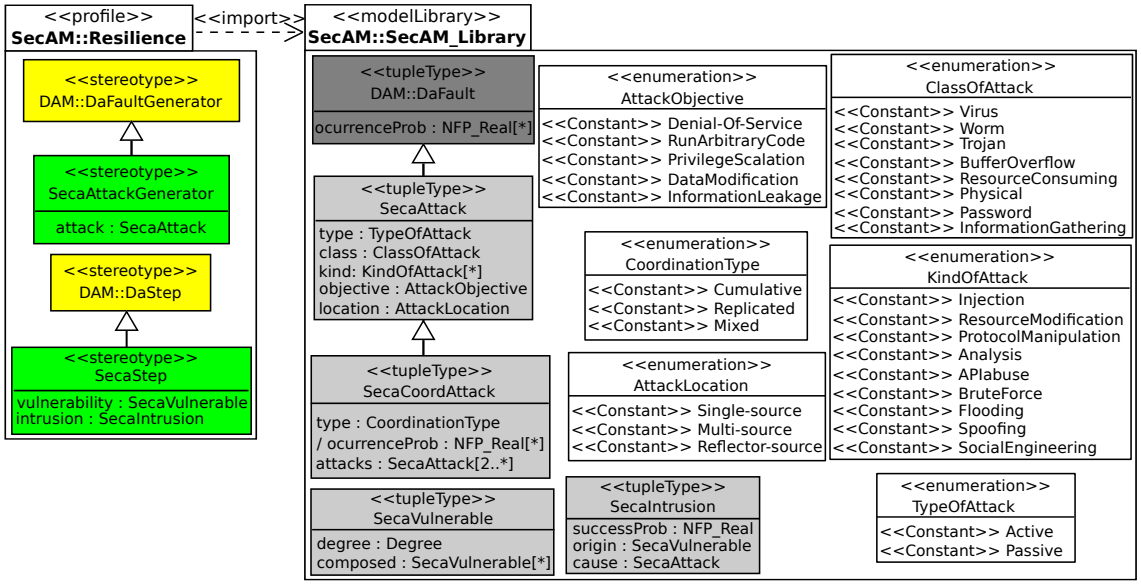


Figure 3.2: The `SecAM::Resilience` package.

resource modification. If we focus on the objective of the attack (`AttackObjective`), that is, what the attack is able to provoke in the system, we can distinguish: denial of service, run arbitrary code, privilege escalation, data modification and information leakage. Finally, considering from where the attack is actuating, three different locations [Hussain et al., 2003] can be identified (`AttackLocation`): single-source (originated at only one host), multi-source (replicated over multiple hosts), and reflector-source (the attacker uses legitimate hosts to attack the victim, hiding so his identity or amplifying his attack [Paxson, 2001]).

In Figure 3.3 we put on practice the *Resilience* extensions, as UML notes, with a two-fold purpose: 1) to characterize from a qualitative point of view the possible attacks to a server host, the activities of the server that are vulnerable to such attacks as well as the consequences of an intrusion; 2) to provide quantitative input parameters for carrying out vulnerability analysis. Observe that the annotations could appear clumsy even for a simple model then affecting the readability and/or scalability of profiling approaches. However, they are introduced for illustration purposes. Indeed, most of the current UML-CASE tools provide support to profiling techniques through proper visualization features.

The values assigned to tags are expressed using the VSL syntax. In particular, for complex NFP different values can be set: a value or variable name prefixed by the dollar symbol (*value* property); the origin of the NFP (*source*), e.g., an estimated or measured value; the type of statistical measure (*statQ*), e.g., a mean or a variance.

Figure 3.3 depicts a simple UML State Machine diagram representing a server attending customer requests. Such requests come from a WAN connection which can be exploited

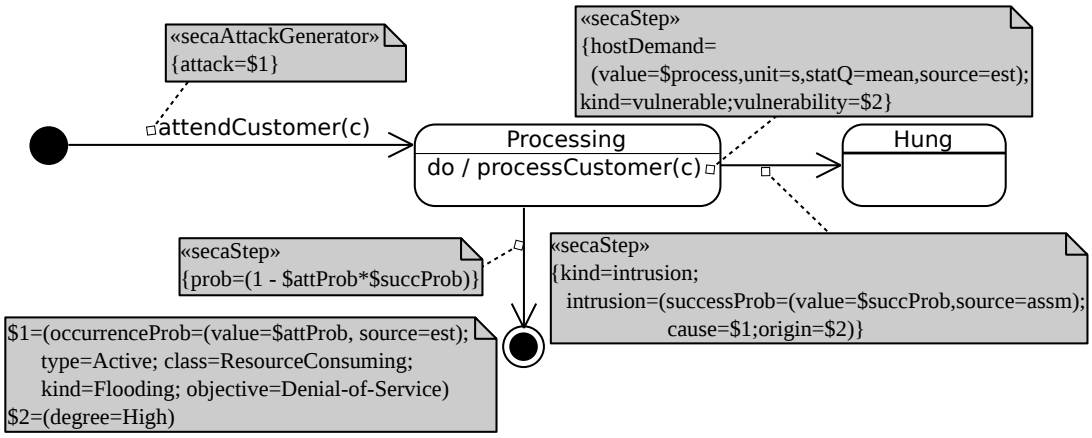


Figure 3.3: A UML-State Machine diagram with `SecAM::Resilience` annotations.

by external attackers (transition stereotyped `secaAttackGenerator`). The expected attack (`attack` tagged-value) is classified as active, resource-consuming, denial-of-service and flooding. The probability of an attack (`occurrenceProb`) is specified as an input parameter (*attProb*). The do-activity in the *Processing* state (stereotyped `secaStep`) is highly vulnerable to attacks (`vulnerability` tagged-value). We have used `hostDemand` to specify the estimated mean time duration (in seconds) of the do-activity as a parameter (*process*). The state *Processing* owns two immediate outgoing transitions, the annotations attached to the latter are meant to resolve the conflict in a probabilistic manner. When the incoming request is an attack, then an intrusion arises that leads to a process crash. Otherwise, the process terminates correctly. The transition from *Processing* to *Hung* is an intrusion step; it may occur with a probability $attProb \cdot succProb$, where *attProb* is the probability of an attack and *succProb* is the probability that, given an attack, it finally succeeds.

3.2.2 `SecAM::Cryptographic` package

Cryptography [Menezes et al., 1996] is primarily used to gain confidentiality over the communications between pairs; however, it also supports data integrity and authentication. Confidentiality assures that information is not disclosed to those unauthorised to own it, while data integrity guarantees the information keeps unaltered. Finally, authentication happens at two levels: entity authentication, that ensures two (or more) participants on a communication are identified, and data origin authentication that guarantees the information has been delivered from the origin.

The *Cryptographic* package has been devised to support mainly the specification of cryptographic design rather than its analysis as it happened with the resilience package. So, by adding contextual information to the UML models, such as when an encryption/decryption takes place and the main characteristics of these processes.

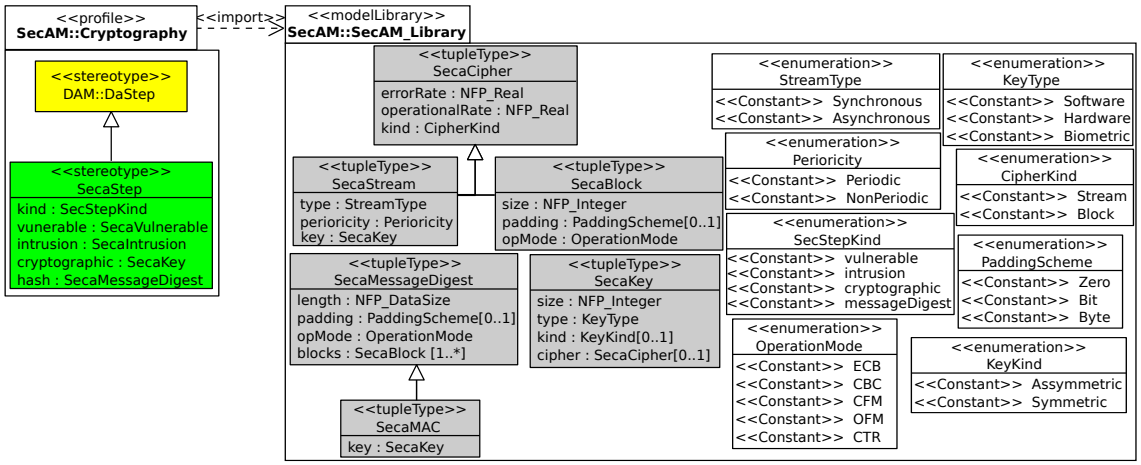


Figure 3.4: The `SecAM::Cryptographic` package.

Figure 3.4 depicts the set of extensions for specifying cryptography into UML behavioural models. The `SecaStep` stereotype, already considered in the *Resilience* subpackage, is now used to specify a cryptographic step through the new tags: `kind`, `cryptographic` and `hash`.

The `cryptographic` tag is a complex type (`SecaKey`) that enables to characterize the key, either asymmetric or symmetric (`KeyKind`). The latter can be of different types, depending on how/where it is deployed (`KeyType`): software, hardware (i.e., cryptographic devices) or biometric (e.g., fingerprint, facial recognition or retinal scanning). A cipher (`SecaCipher`) can be either a block or stream cipher, depending on the algorithm, and it uses a key. It is characterized by an error rate, i.e., the ratio of errors that the cipher can suffer during the process of encryption/decryption, and an operational rate, i.e., the number of encrypted/decrypted bits per time unit.

A stream cipher (`SecaStream`) uses a key-stream to cipher/decipher plain text, which generates a stream of secret bits given an initial key. It can be either self-synchronous (i.e., *ciphertext-auto-key*, CTAK) or synchronous (i.e., *key-auto-key*, KAK), depending whether the used key-stream is influenced or not by the ciphered/deciphered text. Besides, a stream cipher can be either periodic or non-periodic (e.g., Vernam cipher, running-key), depending on the self-repetition of the key-stream.

A block cipher (`SecaBlock`) has a block size, that is the number of characters (or bits) of the plain text message which can be ciphered at a time. Normally, the partition of the message into blocks is not exact and, therefore, a padding scheme is needed to "fill the gaps" then existing several padding schemes.

Besides, a block cipher uses an operation mode (`OperationMode`) which determines its encryption/decryption scheme. Several operation modes have been proposed; without loss of generality, we rely on the ones approved by NIST [Dworkin, 2001].

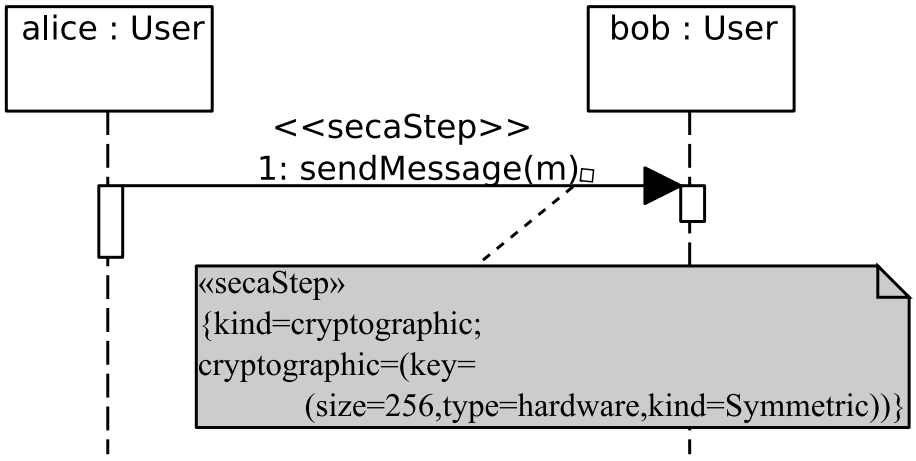


Figure 3.5: An encrypted communication (symmetric, hardware, and 256 bits).

A message digest (`SecaMessageDigest`), also called *hash*, is a value calculated through a cryptographic hash function. Such a value is used, for instance, to determine if a message has been altered. When the cryptographic hash function uses a key (then called keyed hash function), the obtained value is called message authentication code (MAC, `SecaMAC`). MAC values assure data integrity and data authenticity.

Figure 3.5 exemplifies the usage of the cryptographic extensions in a sequence diagram, where *Alice* sends a message to *Bob*. In this naïf example we have used the `secaStep` stereotype for indicating this communication step occurs in a cryptographic way, that is, an encryption is taking place. For instance, the encryption uses a key of 256 bits, the encryption process is done by hardware (e.g., a hardware dongle attached to *Alice*'s terminal) and using symmetric scheme.

3.2.3 `SecAM::SecurityMechanisms` package

The *Security Mechanisms* package, shown in Figure 3.6, aims to represent, in architectural system views, the different kinds of software or hardware devices used to attain security. For instance, from basic cryptographic devices [Menezes et al., 1996] to more sophisticated ones [Cheswick et al., 2003, Scarfone and Mell, 2007, Sousa et al., 2010b], the use of security communication links (e.g., Virtual Private Networks), honeypots [HoneyNet Project, 2004] or any kind of security software (e.g., antivirus, firewall software, etc.).

A security mechanism can be located either in a host machine or in a network. In the first case, it can be either a software, e.g., the Windows firewall or UNIX packet-filtering rules (`SecaHostFirewall`). In the second case, it can be a hardware device (e.g., a wormhole [Sousa et al., 2010b] or a cryptographic token/accelerator [Menezes et al., 1996]). The

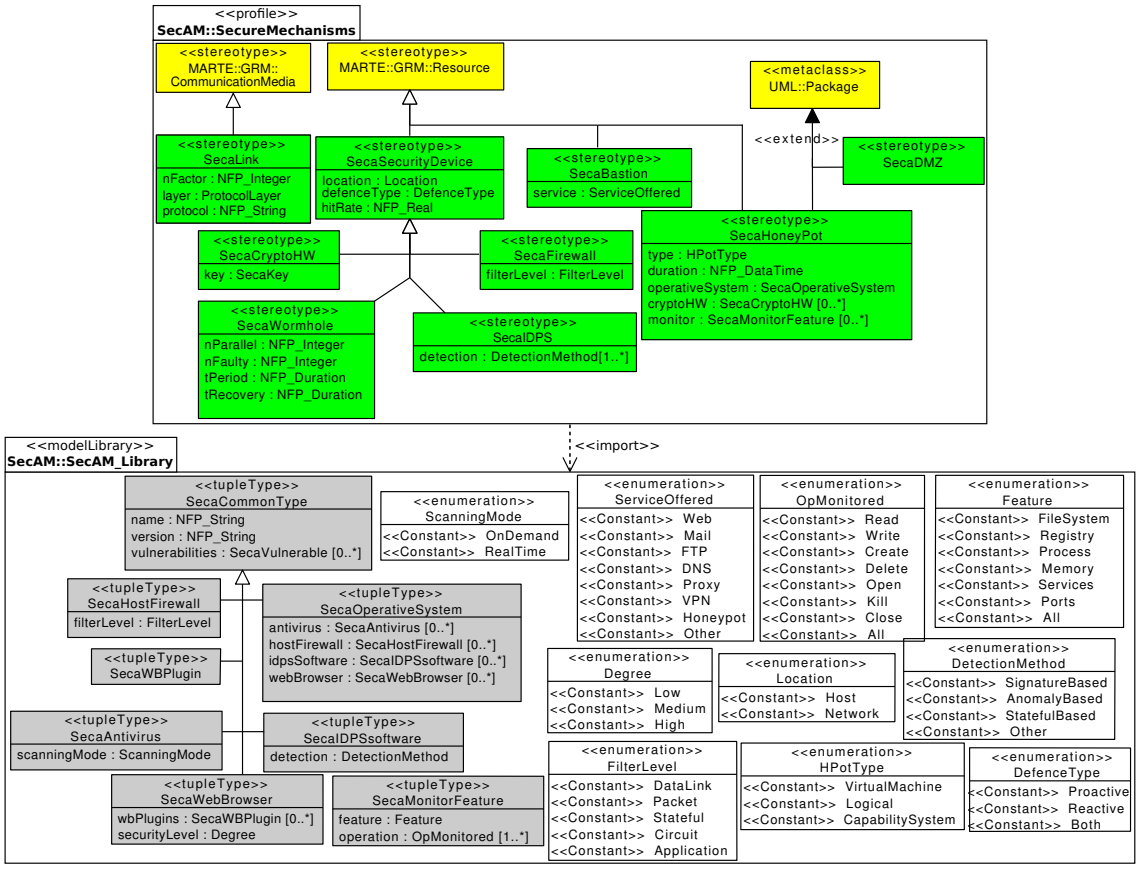


Figure 3.6: The `SecAM::SecurityMechanisms` package.

type of defense (`DefenceType`) of the security mechanism can be: *proactive*, if it prevents from possible security incidents, *reactive*, if it increments security once detected a security breach, or *both* [Sousa et al., 2010b].

Several types of specialised security hardware mechanisms can be distinguished, i.e., crypto hardware (`SecaCryptoHW`), wormholes (`SecaWormhole`), firewalls (`SecaFirewall`) and Intrusion Detection and Prevention System (IDPS) (`SecaIDPS`). Cryptographic tokens (e.g., USB tokens, smart cards or retinal scanning) and cryptographic accelerators which free the host CPU from performing cryptographic actions (e.g., the Sun Crypto Accelerator 6000 PCIe Card) are examples of crypto hardware [Menezes et al., 1996]. These devices may have a key which is used to perform cryptographic actions (the `SecaKey` complex type is shown in Figure 3.4).

A wormhole [Sousa et al., 2010b] enables both proactive and reactive defense on the system and is characterized by several parameters: the number of tolerated faulty hosts (`nFaulty` tag), the number of host replicas which can be recovered in parallel (`nParallel`) and timing parameters (the recovery time, `tRecovery`, and the period, `tPeriod`). A firewall [Cheswick et al., 2003], usually a network-location device, filters packets from the network at different layers of the OSI architecture (`FilterLevel`), e.g., data-link or application-based filtering. An IDPS monitors the network (or a single host) to discover security breaches. Several detection methods exist (`DetectionMethod`), our profile relies on the ones described by NIST [Scarfone and Mell, 2007].

An aggregation of security hardware devices can conform to a demilitarised zone (DMZ) [Cheswick et al., 2003]. There exists many configurations for a DMZ, from a single router to a more complex architecture. The hosts deployed on a DMZ are called *bastions* (`SecaBastion` stereotype), they offer secure services (e.g., web, mail, FTP, DNS, honeypot) and they can be protected by wormholes to preserve their functionality.

A honeypot (`SecaHoneyPot` stereotype) can be a virtual machine, a sandbox (logical) environment or a real system with capability features. Usually, a honeypot is used to test security metrics into systems, or to attract the attacker to a controlled and confined environment. Through `operativeSystem` tagged-value, we can specify the software which is running in such confined environment knowing the security breaches (e.g., vulnerabilities) within it. Besides, security monitoring tools can enhance the security of confined environment (via `monitor` tagged-value, specifying what operation and what feature are being monitored).

A secure communication link (`SecaLink` stereotype) is characterised by the number of authentication factor (e.g., one level if only a password is needed, two levels if a correct combination of user/password is needed, and so on), the layer where the secure communication is taking place and the specification of the secure protocol (e.g., IPSec).

The `SecaSecurityDevice` and the rest of stereotypes specialise the `MARTE::GRM:Resource` stereotype, then they can be applied to classes, lifelines or connectable elements. `SecaLink` stereotype specialises `CommunicationMedia` from MARTE, so it can be applied to any element representing a communication link (e.g.,

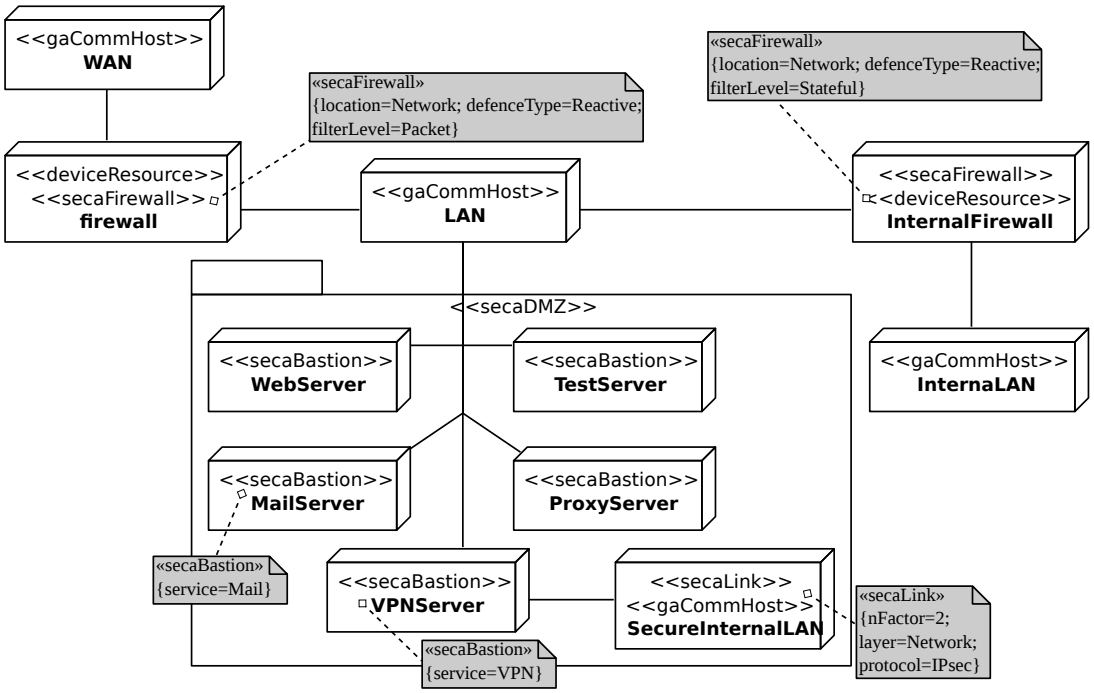


Figure 3.7: A deployment scenario composed by a DMZ and different bastions.

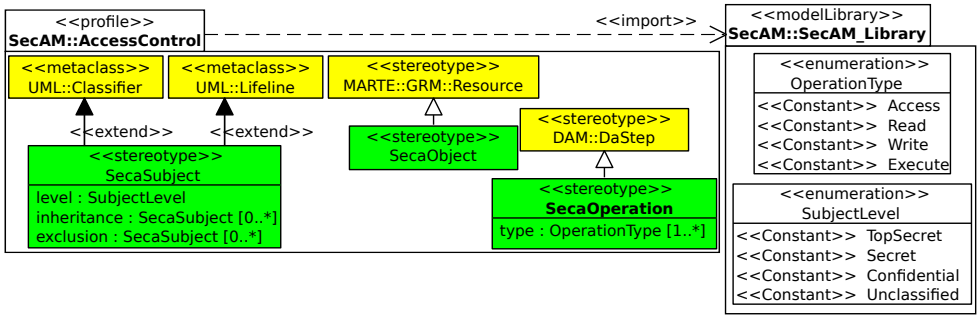


Figure 3.8: The `SecAM::AccessControl` package.

nodes or connectable elements). Finally, the `SecaDMZ` and `SecaHoneyPot` stereotypes, which extend the `Package` UML meta-class, can be applied to a UML package including the bastion hosts (if any) and the security hardware devices integrating the DMZ or any security software which integrate a host machine.

Figure 3.7 depicts a UML deployment diagram annotated with `SecAM::SecurityMechanisms` profile. The firewall connected to the WAN is said to use a packet-based filter, while the internal firewall uses a stateful-based filter. In the DMZ, each node has been annotated as `secaBastion` specifying the deployed service. For illustration purposes, we have only annotated two bastions. Finally, the VPN server has a secure communication link (`secaLink` stereotype) which uses a username/password authorisation, works on the network layer and uses the IPsec protocol.

3.2.4 `SecAM::AccessControl` package

The *Access Control* package addresses confidentiality, integrity and authorisation issues. Access control can be classified depending on the policy used in three basic groups [Bertino and Crampton, 2007]: Mandatory Access Control (MAC)[2], Discretionary Access Control (DAC) and Role-Based Access Control (RBAC) [Sandhu et al., 1996].

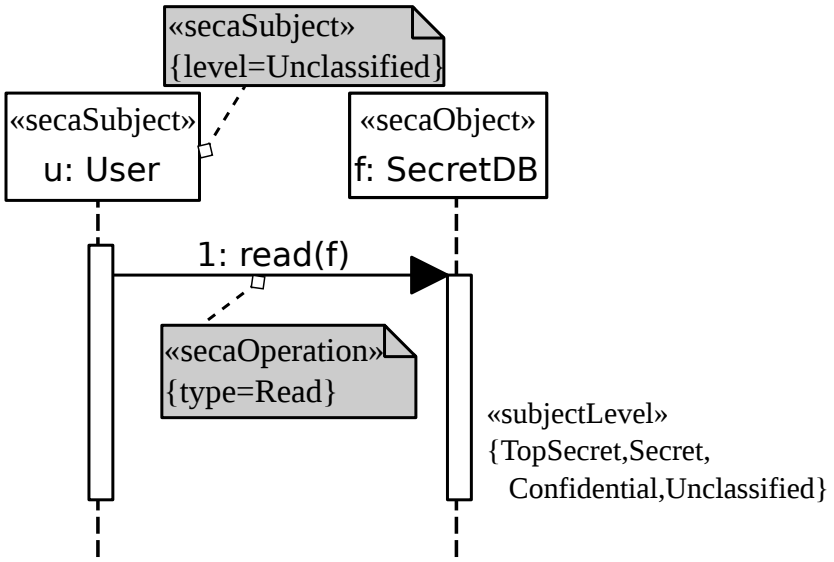
The small set of stereotypes defined in Figure 3.8 include the necessary concepts to specify all the aforementioned access control and integrity policies, and also any other access control policy that a user can devise. Control policies rely on:

1. Who is accessing to the system (`SecaSubject` stereotype);

2. Which objects should be protected for un-authorised access or modification (`SecaObject` stereotype);

3. Which operations a user wants to perform on the available objects (`SecaOperation`).

[2]MAC is also called Lattice-Based Access Control (LBAC).



context SecretDB **inv:**
read.sendEvent.constraint.stereotype.name='SecaUser'
and read.sendEvent.constraint.constrainedElement -> size() = 1
and (read.sendEvent.constraint.constrainedElement -> any(true)).level >= Secret

Figure 3.9: A UML-SD with access control policy.

In `SecaSubject` the attributes `inheritance` and `exclusion` allow to specify delegation of authority and separation of duties, respectively, two major concerns in access control policies. The other attribute, `level`, allows to specify the level of security of the subject. Such level aims to be defined by the engineer depending on the problem context. For instance, `High`, `Medium`, `Low` levels when designing a military security policy, or `Manager`, `Developer`, `Administrative` when dealing with a company's hierarchy.

The `type` attribute (`SecaOperation` stereotype) indicates the type of the operation to be performed. The set of operations (`OperationType` values) can be redefined in order to customise them to the target domain problem. For instance, if the engineer wants to consider a payment as an operation, s/he can define a permission constant such as `payment` which will be valid in his/her domain.

Permissions of an object (i.e., which operations are permitted on the protected objects) are reflected through the definition of constraints in Object Constraint Language (OCL) [OMG, 2010]. Moreover, OCLs can be defined to specify the different access control policies.

Figure 3.9 depicts a UML sequence diagram where there is a user (`SecaSubject`) and a database with sensitive data (`SecaObject`). The user tries to read into the protected object. The OCL in the Figure specifies the MAC policy, indicating that the security level of user

must be greater or equal than `Secret`. However, relying on the final user for the definition of the OCL constraints that define the access control policies may result in a non-trivial issue. For this reason, we are working on an automatic methodology to make it easier. For instance, providing check lists for collecting security attributes and then deriving the OCL constraints automatically. This an interesting step which deserves further study.

3.3 Concluding Remarks

There exists a real need to devise new methods for the development of complex, large scale and distributed systems which are exposed to malicious security issues. The profile we present here is an asset, inside UML, for these methods to take advantage of the four principles that we devised: integration, analysis, unified view and leveraging of current trends in software engineering.

A solution which tackles these security issues should consider them as an integrated approach, that is, considering security in the early stages of the software and systems life-cycle. Besides, such solution should also promote the analysis of system security before the deployment stage, a unified view of security issues and, at the same time, fit properly into the current software engineering techniques.

`SecAM` presents a powerful UML framework for the specification and analysis of security. `SecAM` is made of different sub-packages, each one targeting a subset of well-known security attributes: integrity, availability, confidentiality, authorisation, non-repudiation, authenticity.

A plug-in for Eclipse tool (built on MARTE-DAM profile plug-in) has been developed. However, this plug-in is in a (very) early development phase thus is not fully operative and not yet released. As future work, we plan to develop plug-ins for applying `SecAM` into UML tools, e.g., Eclipse (through a fully operative plug-in) or ArgoUML. Moreover, `SecAM` needs to be applied in complex case studies, and further extensions have to be developed for addressing concrete application domains.

Besides, we aim at extending the transformation to other interesting dependability/security analysis models, such as Fault Trees or Bayesian Networks. We also plan to define model-to-model (M2M) transformations to automatically compute `SecAM` derived tagged-values, to verify `SecAM` UML-profiled model for consistency (i.e., using OCL constraints) as well as to compute vulnerability related metrics.

Chapter 4

Fault-Tolerant Techniques for Critical Systems

This chapter summarises the main contributions of this dissertation related to the modelling of Fault-Tolerant techniques (FTTs). The proposed models have resulted in several publications [Rodríguez and Merseguer, 2010, Rodríguez et al., 2012d, Rodríguez et al., 2013b].

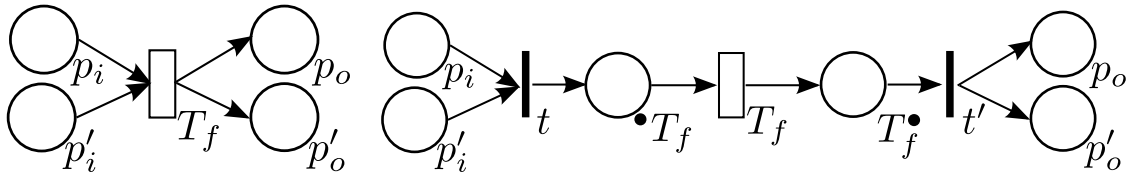
4.1 Motivation

As we claim in Section 1.1, large scale and distributed systems deployed in heterogeneous environments are subject to be a target of harmful attacks, with the intent of performing an intrusion, confidential data theft or other type of attacks. These systems, whose provided services may suffer some degradation due to errors and failures triggered by attacks, are commonly called *degradable systems*.

Normally, degradable systems include Fault-Tolerant (FT) techniques [Avizienis, 1997, Avizienis et al., 2004] that provide mechanisms to deal with failures inside the system and mitigate the consequences of faults. Some examples of FT techniques are: switching system requests between non-faulty components, adding watch-dogs for checking liveness of system components, or software exception handlers. A degradable system equipped with a FT technique is called a *FT system*.

Therefore, 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 the aforementioned FT Techniques (FTTs), such as recovery procedures, and/or Security Mechanisms (SMs), such as encryption of data, in order to react to intrusions.

This chapter addresses the issue of integrating already developed fault-tolerant (FT) techniques into software designs for their analysis through automatically obtained formal models (as it is shown in Chapter 5). More precisely, this chapter is subdivided in two



(a) Original model (b) Transformed model

Figure 4.1: Transformation rule $\mathcal{TR}$ of a transition t_f subject to fail (faulty transition).

main sections, one of them devoted to a review of FT concepts [Avizienis et al., 2004, Avizienis, 1997] and our proposal of compositional Petri net (PN) models for FT techniques, and a second one where we propose a UML model library containing FT techniques that are ready to use in UML designs of critical systems.

Firstly, we introduce a compositional PN model for FTTs based on the basic concepts of FT given in Section 2.3. These compositional PN models allow us to make sensitive performability analysis easier when some FT parameters change (e.g., expected failure rate or activity timing related to recovery actions). Thus, these FT models can be useful for evaluating different FT approaches in the same system model.

Lastly, we introduce a UML model library which models different FTTs, namely: a *Proactive-Reactive Recovery* technique (inspired in the one given in [Sousa et al., 2010a]), *Switch Over Failing* technique and *Ping And Restore* technique. These UML models can be transformed to PN models using well-known techniques [Distefano et al., 2011, Gómez-Martínez and Merseguer, 2006] (see [Balsamo et al., 2004] for an extensive survey on this topic), and they may allow to test different techniques for the same design to find the ones fitting better. However, such PN models are more complex than the Process PNs that we introduced in Section 2.1 – indeed, they belong to the class of Generalised Stochastic Petri Nets (GSPNs), and therefore these models cannot be analysed with the methods presented in the Part II of this dissertation but with simulation.

4.2 Compositional PN Models for Fault Tolerance

In this section, we provide compositional PN-based models for the Fault-Tolerant (FT) techniques based on the basic concepts of FT given in Section 2.3. Recall that a FT technique may involve both error detection – concurrent or preemptive – and recovery phases – divided in error handling (rollback, rollforward or compensation) and fault handling (diagnosis, isolation, reconfiguration or reinitialisation).

Consider we have a system modelled with a PN in which there is an activity (represented by a timed transition T_f) which is subject to fail. We called it *faulty transition*, as it may lead to a fault. Before adding any FT technique to the system, we apply a transformation rule $\mathcal{TR}$ in the PN. This transformation rule allows us to apply our approach in the general

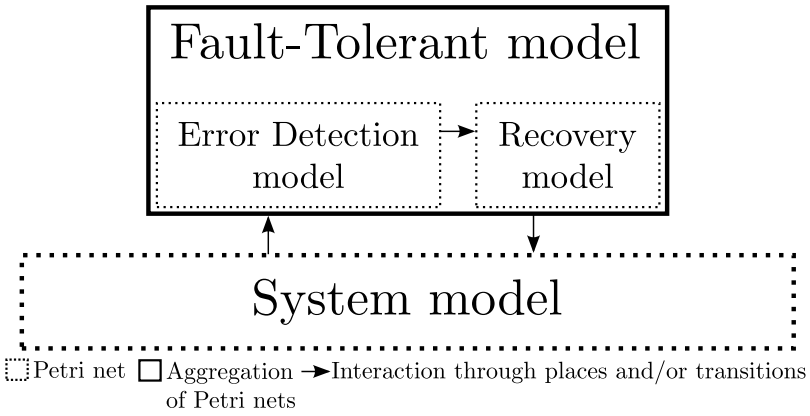


Figure 4.2: Integration between a PN-based system model and a PN-based FT technique.

case, and it is not modifying the behaviour of the original PN model anyhow.

Figure 4.1 shows how this transformation rule $\mathcal{TR}$ works: two immediate transitions t and t' and two places ${}^\bullet T_f$ and $T_f^\bullet$ are added just from(to) transition T_f, and all input(output) places of transition T_f are accordingly connected to transition t and t'.

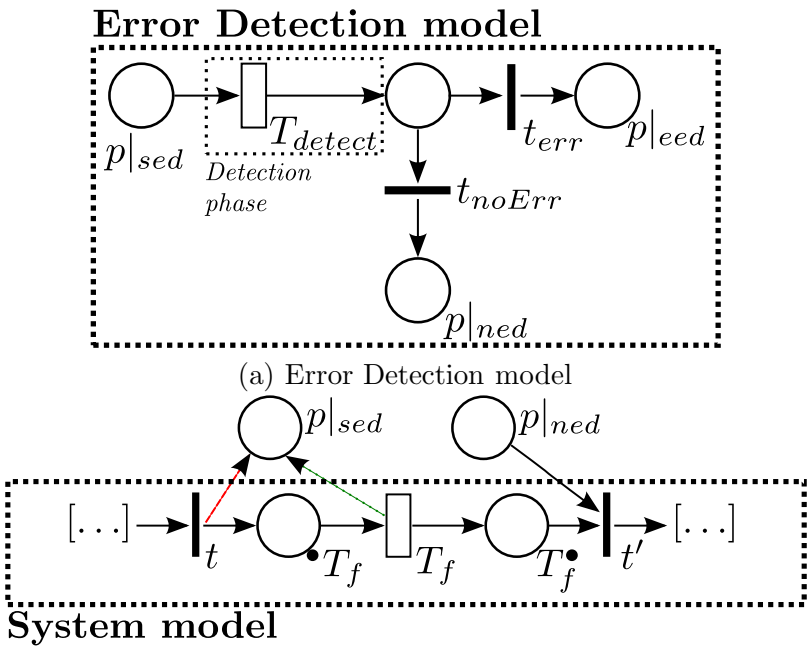
Figure 4.2 depicts the interaction between a PN that models the behaviour of a given system and a PN that models a FT technique. A PN-based FT model is subdivided in *Error Detection* and *Recovery* sub-models. Each sub-model represents respectively the phases involved in a FT technique. In the sequel, we explain each model and its interactions in detail.

4.2.1 PN Error Detection Model

Figure 4.3(a) depicts the PN model for error detection. The timed transition T_{detect} represents how long the error detection activity takes. Note that this transition is abstracting the behaviour for detecting an error, so that it may be refined into a more complex model representing error detection in more detail (*Detection phase* in Figure 4.3(a)). After error detection activity takes place, the presence of an error is discriminated. When an error arises (transition t_{err}), then a token is put on place $p|_{eed}$. Otherwise, a token is put on place $p|_{ned}$.

The integration between the Error Detection model and the System model is done through labelled places $p|_{sed}, p|_{eed}$ (a labelled place p is defined as $p|_{label}$). We have followed the compositional rules over the places defined in [Donatelli and Franceschinis, 1996, Bernardi et al., 2001] to combine models using labelled places: pairs of places with matching labels are superposed. Figure 4.3(a) depicts the places $p|_{sed}, p|_{ned}$ added to the system model. The origin of the incoming arc of place $p|_{sed}$ depends on the type of error detection, and synchronises the execution of error detection model with the system model: when con-

current, the arc added is the red-dashed one (from t to $p|_{sed}$); otherwise (preemptive), the green-dotted arc is considered (from T_f to $p|_{sed}$). Note that the place $p|_{ned}$ is synchronised with $T_f^\bullet$ (which indeed is added to the system by transformation rule $\mathcal{TR}$).



(a) Error Detection model

(b) Places $p|_{sed}, p|_{ned}$ added to the system model

Figure 4.3: PN-based model of Error Detection and faulty activity inside the system.

This simple model allows us to represent the most common error detection techniques, e.g., to validate input data, or intermediate data generated and reused during a faulty transition (it can be concurrently done), or to validate output after a faulty transition execution (preemptive).

4.2.2 PN Recovery Model

The recovery phase involves two steps, a first (optional) step of error handling (rollback, rollforward or compensation) and a second one of fault handling technique (diagnosis, isolation, reconfiguration or reinitialisation).

Following the definitions given in [Avizienis et al., 2004], we have grouped the fault handling techniques in two groups: diagnosis and reinitialisation techniques; and isolation and reconfiguration. This decision is based on the abstracted behaviour of these techniques, as we explain henceforward. We have composed models that represent valid combinations of the recovery phase as it is shown in Table 4.1. This classification is made based on how the techniques work. For instance, we believe that a rollforward technnique cannot be combined

	Rollforward (& compensation)*	**Rollbackward (& compensation)***
Diagnosis	√	√
Isolation	√	√
Reconfiguration	X	√
Reinitialisation	X	√

Table 4.1: Valid combinations of error handling and fault handling techniques. The symbol * means optional.

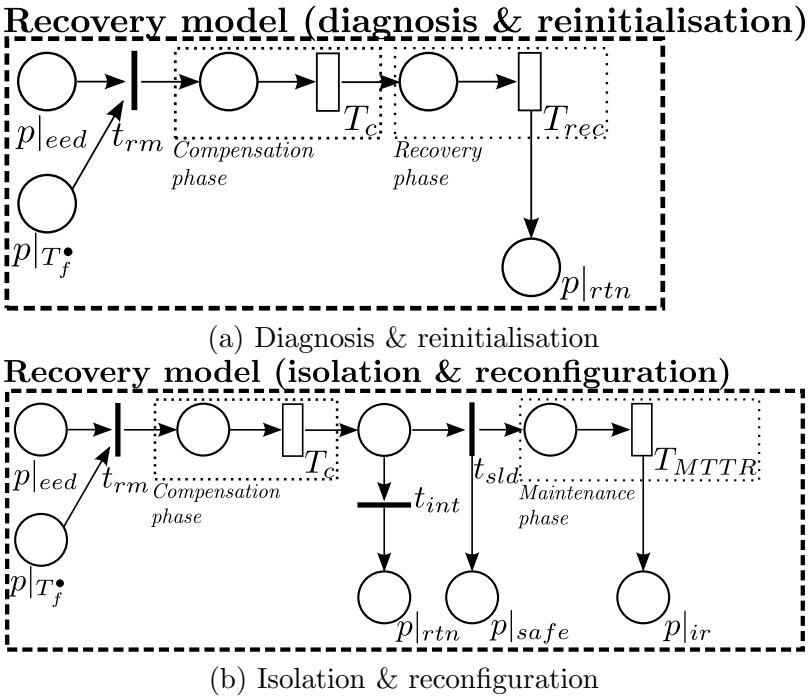
with reconfiguration or reinitialisation, because reconfiguration switches the request to spare components, while reinitialisation updates and records a new system configuration. Thus, we consider that to move to a future correct state after recovering is meaningless.

Figure 4.4(a) shows the PN model of diagnosis and reinitialisation FT recovery techniques. Place $p|_{eed}$ is superposed with the one of *Error Detection* model, and place $p|_{T_f^\bullet}$ is superposed with place $T_f^\bullet$ in the system model. A token in place $p|_{eed}$ indicates that an error has been detected. Once transition t_{rm} is fired, a (optional) compensation activity may take place (*Compensation phase*). Then, recovery activity takes place (abstracted in *Recovery phase*). As in the previous model of error detection, we have represented compensation and recovery phases as a single timed transitions (T_c and T_{rec}, respectively). These transitions may be refined into a more complex models representing compensation and recovery activities in more detail.

Finally, the token flow is redirected through place $p|_{rtn}$. The superposition of this place depends on the error handling technique used: it will be a place which becomes eventually marked *after* the faulty transition T_f is fired (rollforward), or which was eventually marked *before* its firing (rollback). In both cases and to keep conservativeness of the model, place $p|_{rtn}$ must belong to the p-semiflow associated to the resource r (we called it *faulty resource*), being r the inner resource used by faulty activity. Although a transition T_f can represent an activity where several resources are being used, for the sake of simplicity in this paper we assume that the fault is caused by the use of the inner resource (i.e., the last one acquired). Otherwise, note that after the recovering phase other resources acquired after faulty resource should be released to keep conservativeness.

The difference between diagnosis and reinitialisation technique can be established by the duration of the recovery phase. For instance, when diagnosis technique is considered, the recovery phase will have a much lower duration than when reinitialisation is taken into account due to the actions that are performed.

Figure 4.4(b) shows the PN model of isolation and reconfiguration FT recovery techniques. This case is identical to the previous until the (optional) compensation phase. After the compensation phase takes place, the type of the fault is discriminated [Avizienis et al., 2004] as intermittent (that is, the fault is transient) or solid (i.e.,



(a) Diagnosis & reinitialisation

(b) Isolation & reconfiguration

Figure 4.4: PN-based models of Recovery model: (a) and (b) isolation & reconfiguration.

the faults whose activation is reproducible). When the fault is intermittent, as proposed in [Avizienis et al., 2004], normal execution can keep going on and token is returned to place $p|_{rtn}$ (as before, the superposed place depends on the type of error detection). On the contrary, when a solid fault is detected, the faulty resource is excluded from normal service delivery – as indicated by both isolation and reconfiguration techniques – and the token is moved to the place $p|_{safe}$. We assume that place $p|_{safe}$ is superposed with the place previous to acquire the faulty resource r, i.e., $p|_{safe} = {}^\bullet t_{acq}$, where t_{acq} is the transition where faulty resource r is acquired.

In the case of isolation and reconfiguration, the recovery phase is called *Maintenance phase*, because it involves the participation of an external agent [Avizienis et al., 2004]. We have modelled maintenance phase as a single transition T_{MTTR} that represents the Mean Time To Repair (MTTR) spent on fixing the faulty resource. As in the previous case, this model can be refined to a more complex maintenance model. Anyhow, after maintenance phase takes place the fixed resource is returned to place $p|_{ir}$, which is superposed to the resource place p_r.

As in the previous techniques, the difference between isolation and reconfiguration technique can be established by the duration of the maintenance phase. For instance, when isolation technique is considered, the maintenance phase will have a much greater duration

than when reconfiguration is taken into account.

Finally, note that most of the FT techniques can be modelled with the proposed models. For instance, a *watchdog* can be modelled as a reconfiguration FT technique with concurrent error detection and rollforward (or rollback), and a *check-pointing and rollback* can be modelled as a reinitialisation FT technique. Unfortunately, other FT techniques, such as *n-version programming* or *combined proactive-reactive techniques* [Sousa et al., 2010a] cannot be adapted to the proposed model and some tweaks must be done. We aim to extend these models to cover all FT techniques as a future work.

Running example Let us consider a packet-routing algorithm inside a router where packets arrive and after checking source and destination of the packets, they are filtered following some defined rules. Figure 4.5 depicts a PN modelling such an algorithm. The PN marking represents the number nP of packets (initial marking of the process-idle place, p_0), the number nT of threads attending the incoming packets (initial marking of p_2) and the number nS of filtering-threads (initial marking of p_7). The number nC denotes the capacity of the system. We consider that this number is equal to the number nP of packets, therefore place p'_0 becomes implicit and we omit it for analysis. Packets arrive to the router following an exponential distribution of mean $\delta_0 = 5$ milliseconds[1]. The amount of time for checking packet headers (i.e., source, destination) is represented by transition T_2, which follows an exponential distribution of mean $\delta_2 = 2$ milliseconds. The algorithm's decision is represented by place p_5 and its outgoing arcs: either transition t_4 is fired (then the packet must be discarded, which happens with a probability of 0.75), or transition t_5 is fired. In the latter case, once some filtering-thread is available, it is used. Such a use is represented by T_7 and takes, on average, $\delta_7 = 1$ millisecond to complete. Finally, T_9 represents the final step of the algorithm, that consists in routing the packet(acknowledgement) properly to its destination(source) and takes, in terms of time, about 2 milliseconds, i.e., $\delta_9 = 2$.

This running example will be used henceforward to illustrate our approach. Suppose that the filtering activity may fail, i.e., the faulty transition is T_7. The router manufacturer is interested in adding a watchdog (recall it can be modelled as a reconfiguration FT technique) into the algorithm such that the threads that fail (they are hanged) are discarded, and they are cleaned with a fixed internal timer. In this case, the error detection model is concurrent, as the failure can be detected during normal operation; and the error handling technique used is rollback: when an error is detected, the packet is filtered by another thread, when available.

The resulting PN after adding the FT technique described above is depicted in Figure 4.6. In Section 6.4, this running example is used for sensitive performability analysis.

[1]We use δ_i as an abbreviation for $\delta(T_i)$

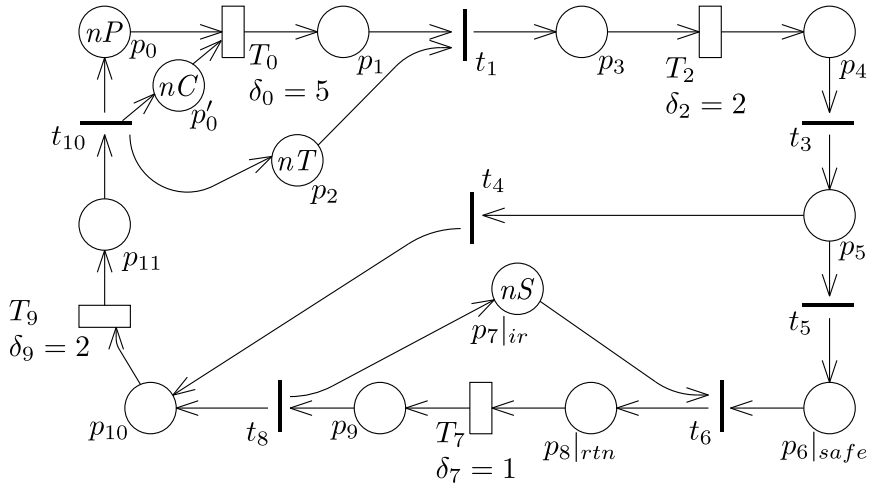


Figure 4.5: Petri net representation of a packet-routing algorithm.

$\mathbf{y}'_1 = \mathbf{y}_1 \cup \{{}^\bullet T_7, T_7^\bullet, p_4^1\}$	$\mathbf{x}'_2 = \mathbf{x}_2 \cup \{t'_1, t'_2, T^1_{detect}, t^1_{noError}\}$
$\mathbf{y}''_1 = \mathbf{y}_1 \cup \{p^1\|_{sed}, p_2^1, p_3^1, p^1\|_{eed}, p_4^1\}$	$\mathbf{x}_3 = \{t'_1, T_7, T^1_{detect}, t^1_{err}, t^1_{rm}, t^1_{int}\}$
$\mathbf{y}'_2 = \mathbf{y}_2 \cup \{{}^\bullet T_7, T_7^\bullet, p_4^1\}$	$\mathbf{x}_4 = \{t_6, t'_1, T_7, T^1_{detect}, t^1_{err}, t^1_{rm}, t^1_{sld}, T^1_{MTTR}\}$
$\mathbf{y}''_2 = \mathbf{y}_2 \cup \{p^1\|_{sed}, p_2^1, p_3^1, p^1\|_{eed}, p_4^1\}$	
$\mathbf{y}'_3 = \mathbf{y}_3 \cup \{{}^\bullet T_7, T_7^\bullet, p_4^1, p_5^1\}$	
$\mathbf{y}''_3 = \mathbf{y}_3 \cup \{p^1\|_{sed}, p_2^1, p_3^1, p^1\|_{eed}, p_4^1, p_5^1\}$	
(a)	(b)

Table 4.2: New (a) p-semiflows and (b) t-semiflows of the PN in Figure 4.6.

4.2.3 Analysis of PN-based FT Models

This subsection analyses how some structural properties are modified when the proposed FT models are added (namely, the p-semiflows and t-semiflows) and how visit ratios properties are as well affected.

P-semiflows Let us analyse how minimal p-semiflows are modified. The addition of the proposed FT models provokes that, for each p-semiflow $\mathbf{y}_r$ associated to a resource r that makes use of the faulty transition t_f (i.e, $\|\mathbf{y}_r\| \cap \{{}^\bullet t_f, t_f^\bullet\} \neq \emptyset$), $\mathbf{y}_r$ is transformed into two p-semiflows $\mathbf{y}'_r, \mathbf{y}''_r, \mathbf{y}'_r \neq \mathbf{y}''_r$ such that $\|\mathbf{y}_r\| \subset \|\mathbf{y}'_r\|, \|\mathbf{y}_r\| \subset \|\mathbf{y}''_r\|$. This transformation is due to the FT models consume/produce tokens from/to the original p-semiflows. These p-semiflows cover all places added by the FT technique, thus the net remains conservative.

For instance, the minimal initial p-semiflows are, in Figure 4.5: $\mathbf{y}_1 = \{p_0, p_1, p_3, p_4, p_5, p_6|_{safe}, p_8|_{rtn}, p_9, p_{10}, p_{11}\}$, $\mathbf{y}_2 = \{p_2, p_3, p_4, p_5, p_6|_{safe}, p_8|_{rtn}, p_9, p_{10}, p_{11}\}$ and $\mathbf{y}_3 = \{p_7|_{ir}, p_8|_{rtn}, p_9\}$. The minimal p-semiflows of the PN in Figure 4.5 that contain

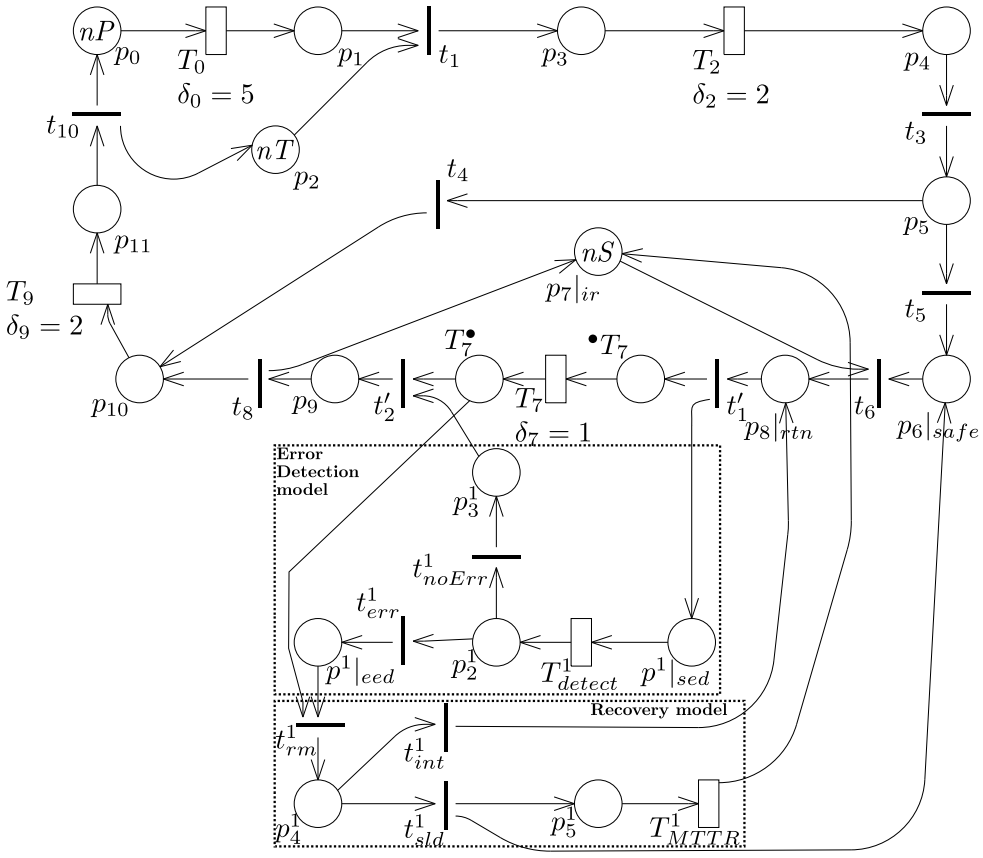


Figure 4.6: Petri net representation of a packet-routing algorithm with a FT technique.

places from/to transition T_7 ($p_8|_{rtn}$ and p_9, respectively) are $\mathbf{y}_1$,$\mathbf{y}_2$ and $\mathbf{y}_3$. Thus, the new p-semiflows of PN in Figure 4.6 are the ones showed in Table 4.2(a).

Note that these new p-semiflows violate the third property of definition of PPN (see Definition 5 in Section 2.1), given that there exist more than a single minimal p-semiflow containing the same resource, e.g., $\mathbf{y}'_2$ and $\mathbf{y}''_2$ contain the resource place p_2 on its support.

T-semiflows Let us focus now on t-semiflows. The addition of the proposal FT models provokes that, for each t-semiflow $\mathbf{x}_f$ containing transition t_f (i.e., $\|\mathbf{x}_f\| \cap \{t_f\} \neq \emptyset$), $\mathbf{x}_f$ is transformed into a t-semiflow $\mathbf{x}'_f$ such that $\|\mathbf{x}_f\| \subset \|\mathbf{x}'_f\|$. Besides, a new t-semiflow appears for each free-choice place added by the proposed FT models – i.e., in the case of an isolation or reconfiguration FT technique two new t-semiflows appear. These t-semiflows cover all transitions added by the FT technique, thus the net remains consistent.

For instance, the minimal t-semiflows are, in Figure 4.5: $\mathbf{x}_1 = \{T_0, t_1, T_2, t_3, t_4, T_9, t_{10}\}$ and $\mathbf{x}_2 = \{T_0, t_1, T_2, t_3, t_5, t_6, T_7, t_8, T_9, t_{10}\}$. The minimal t-semiflow of PN in Figure 4.5

that contains T_7 is $\mathbf{x}_2$. Thus, the new t-semiflows of PN in Figure 4.6 are the ones showed in Table 4.2(b).

Visit ratios Let us focus now on visit ratios. We assume that the transition chosen for normalisation is always the think time of customers, i.e., the timed transition that consumes the process-idle place tokens. We have summarised how the visit ratios of transitions in the original system are modified in Table 4.3. The transitions added by the proposed FT models have visit ratios equal to the new visit ratio of faulty transition. Besides, all transitions from the path modelling the error (i.e., for the FT technique i, from t^i_{err} to T^i_{rec}) has a visit ratio multiplied by w_e, where w_e is the probability of error, and by $1 - w_e$ the transitions on the other path. In the case of isolation and reconfiguration FT models, the transitions from t^i_{sld} to T^i_{MTTR} are as well multiplied by w_s, and by $1 - w_s$ the transitions in the other path.

For instance, we have applied an isolation FT technique in the running example. Therefore, the visit ratios are modified as follows. The transitions on the path from $p|^{\bullet}_{rtn}$ to the faulty transition, $\{p|^{\bullet}_{rtn} \rightarrow t_f\}$, (in the original model) have a visit ratio equal to the previous one times $\frac{1}{(1-w_e)}$, i.e., $\mathbf{v}(t) = \mathbf{v}'(t) \cdot \frac{1}{(1-w_e)}$, where $\mathbf{v}'(t)$ is the visit ratio of transition t before adding an FT technique and w_e the probability of error. The transitions on the path from $p|^{\bullet}_{safe}$ to ${}^{\bullet}p|_{rtn}$ (in the original model) have a visit ratio equal to the previous one times $\frac{(1 - w_e \cdot w_s)}{(1-w_e)}$, where w_s is the probability of having a solid fault.

4.3 UML Fault-Tolerant Techniques Library

This section introduces a bunch of Fault-Tolerant Techniques (FTTs) modelled with UML. The key idea of this "FTTs model library" is to provide a software engineer with UML models ready to use in the design of critical designs. These UML FTTs models have clearly defined interfaces, which make its use easier for practitioners.

The main goal 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.

This UML FTTs library is currently composed by three different models, each one modelling a different FT technique. In the sequel, we introduce each UML FT model and define it in more detail.

4.3.1 Proactive-Reactive Recovery Technique

Recall that Fault-Tolerant Techniques (FTTs) 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. Depending on when FTTs are applied (i.e., they act), they can be classified

	Error handling: Backward
(a)	$\forall t \in \mathbf{Q}_1, \mathbf{v}(t) = \mathbf{v}'(t) \cdot \frac{1}{1 - w_e}$
(b)	$\forall t \in \mathbf{Q}_1, \mathbf{v}(t) = \mathbf{v}'(t) \cdot \frac{1}{1 - w_e}$
	$\forall t \in \mathbf{Q}_3, \mathbf{v}(t) = \mathbf{v}'(t) \cdot \frac{(1 - w_e + w_e \cdot w_s)}{1 - w_e}$

	Error handling: Forward
(a)	$\forall t \in \mathbf{Q}_2, \mathbf{v}(t) = \mathbf{v}'(t) \cdot (1 - w_e)$
(b)	$\forall t \in \mathbf{Q}_2, \mathbf{v}(t) = \mathbf{v}'(t) \cdot \frac{1 - w_e}{(1 - w_e \cdot w_s)}$
	$\forall t \in \mathbf{Q}_4, \mathbf{v}(t) = \mathbf{v}'(t) \cdot \frac{1}{(1 - w_e \cdot w_s)}$

$\mathbf{Q}_1 = \{p|^{\bullet}_{rtn} \rightarrow T_f\}, \mathbf{Q}_2 = \{T_f^{\bullet\bullet} \rightarrow {}^{\bullet}p|_{rtn}\}$
$\mathbf{Q}_3 = \{p|^{\bullet}_{safe} \rightarrow {}^{\bullet}p|_{rtn}\}, \mathbf{Q}_4 = \{p|^{\bullet}_{safe} \rightarrow T_f\}$

Table 4.3: Visit ratios modification for different error handling techniques of recovery models: (a) diagnosis & reinitialisation and (b) isolation & reconfiguration. $\mathbf{v}'(t)$ means previous value of visit ratio of transition t; w_e is the probability of error and w_s the probability of having solid faults.

as [Avizienis et al., 2004]: (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 [Sousa et al., 2010a].

Proactive recovery transforms a system state containing one or more errors (or even visible faults) into a state without detected errors or faults. Proactive techniques were presented in [Canetti et al., 1997a] as a long-term protection against break-ins and implemented, for example, in the scope of an on-line certification-authority system [Zhou et al., 2002]. These techniques borrow ideas from proactive discovery protocols (e.g., IPSec [Tran, 2006]), session-key refreshment (SSL/TLS [Dierks and Rescorla, 2006]) or secret sharing algorithms [Shamir, 1979]. Hence, proactive security is defined as a combination of periodic refreshment and distribution [Canetti et al., 1997b, Ostrovsky and Yung, 1991].

On the contrary, a *reactive recovery* [Avizienis et al., 2004] performs a concurrent error detection, that is, errors in the system are detected meanwhile it is working. Then, a detection implies some actions must be performed in order to recover the system to a free-error state.

Proactive and reactive recovery techniques should not be considered as mutually exclusive but as complementary. Briefly, proactive techniques are worried about fault prevention (passive part of the system), while reactive ones are concerned with fault removal (active

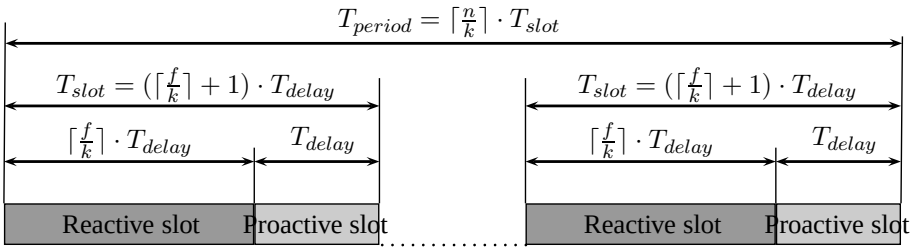


Figure 4.7: Schedule time-line showing activation of reactive and proactive recoveries (adapted from [Sousa et al., 2010a]).

part). Sousa et al. presented in [Sousa et al., 2010a] a real application of proactive and reactive recovery techniques to an existing critical system, which tolerates up to f failure nodes and is able to recover in parallel up to k nodes. The rationale behind this idea is a scheduled time-line, which will be modelled in the sequel (Figure 4.7, adapted from [Sousa et al., 2010a], depicts it) and it is explained in the following.

A system with n distributed devices is initially divided into $\lceil \frac{n}{k} \rceil$ groups, containing each one up to k devices, being k the number of simultaneous recoveries the system can support. Assuming a period of time T_{period}, then each one is divided in $\lceil \frac{n}{k} \rceil$ slices (called T_{slot} from now on) where both (i.e., proactive and reactive) recoveries have to be performed. In a T_{slot}, one proactive recovery will be activated for a selected group which has a duration equal to T_{delay}, being T_{delay} the maximum expected time for recovering a device. Regarding reactive recovery, if we assume up to f failures and k simultaneously recoveries, that implies a maximum of $\lceil \frac{f}{k} \rceil$ reactive activations may happen in a T_{slot}. As can be inferred, T_{slot} has a duration equal to $(\lceil \frac{f}{k} \rceil + 1) \cdot T_{delay}$. There exists a relation [Sousa et al., 2010a] between values of n, f and k as is shown in equation (4.1).

$$n \geq 2 \cdot f + k + 1 \tag{4.1}$$

A deeper description of the schedule time-line for proactive and reactive recoveries, as well as justification for inequality shown in equation (4.1), can be found in [Kalan et al., 2008].

UML Modelling Henceforward, we develop, a generic and reusable model of proactive and reactive recovery techniques. In first term, we model them using UML state-machine (UML-SM) diagrams annotated with the previously discussed profiles (MARTE-DAM and `SecAM`, see Chapter 3). Then, we transform it to a Coloured Petri Net (CPN) [Jensen, 1997] which maps the behaviour of these UML diagrams. In fact, this CPN accurately represents proactive and reactive recovery techniques. So, our proposal to reuse the "proactive-reactive" CPN within a given software design has to offer adequate "interfaces" to compose both CPNs. Then, we finally get a CPN that embeds both the proactive-reactive techniques

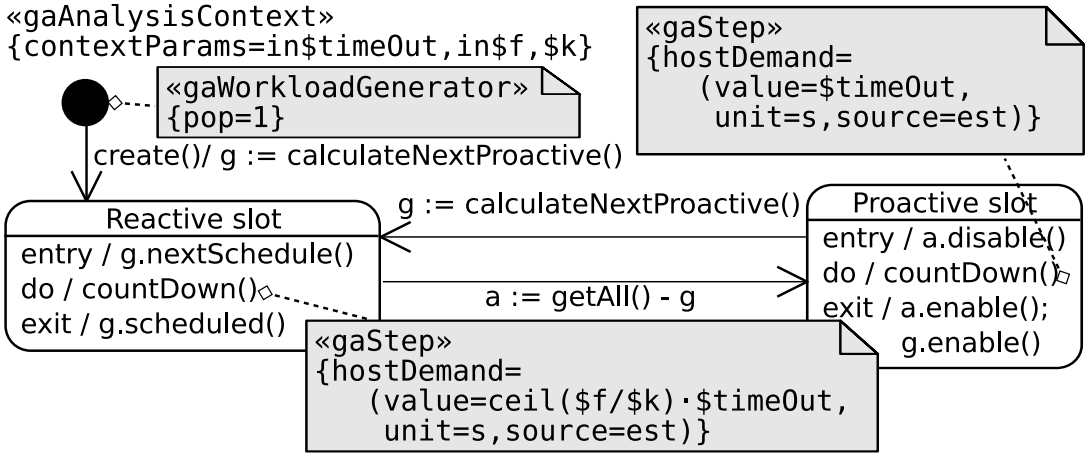


Figure 4.8: Scheduler UML state-machine diagram.

and the software design.

We have distinguished two components, one in charge of controlling the scheduled timeline presented early in this section, and the other controlling the device to be recovered. The latter has been called Proactive and Reactive Recovery (PRR) component following terminology in [Sousa et al., 2010a].

Schedule controller UML state-machine (UML-SM) diagram is depicted in Figure 4.8. Initial analysis variables (`gaAnalysisContext` stereotype) are: `tDelay`, which determines the duration of each recovery; `f`, number of faulty devices allowed; and `k`, number of devices recovered in parallel. Only one controller will be placed in the system (tag `pop` of `gaWorkloadGenerator` stereotype). Once created, it calculates in `g` the first group which will be proactively recovered. Upon entrance into *Reactive slot* state, it invokes event `nextSchedule()` for PRR devices in `g` to inform them that the components they control will be proactively recovered in the next proactive slot, so their monitoring activity will not be necessary since for sure they will be recovered. Then, it starts the `countDown()` activity with duration `hostDemand` equals to $\lceil \frac{f}{k} \rceil \cdot tDelay$ seconds (that is, it makes room for up to $\lceil \frac{f}{k} \rceil$ parallel recoveries). Completion of `countDown()` activity means to schedule elements in `g` and to change to *Proactive slot* state, where all PRR devices are disabled and it starts the proactive `countDown()` activity, in this case with a duration equal to `tDelay` seconds. Once finished, it enables all PRR devices and before entering again in the *Reactive slot* state, it calculates the next proactive group.

Figure 4.9 shows UML-SM for PRR component controller. Obviously, the population is equal to the number of effectively monitored devices, `nDevices`. It starts in *Enabled* state and executing the activity `monitor()`, which abstracts two processes: 1) detection of errors in the monitored device and 2) checking for room in current time slot for a reactive recovery. So, when it positively informs, then enters in *Reactive* state to perform a recovery

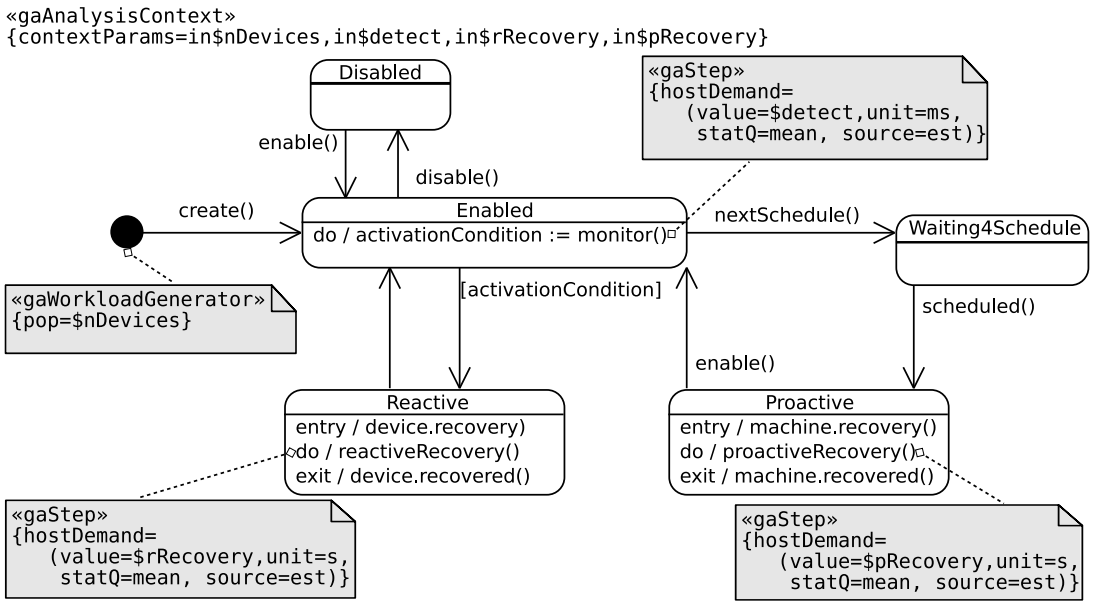


Figure 4.9: PRR controller UML state-machine diagram.

(`reactiveRecovery()` activity), which has a duration of `rRecovery` seconds on average. Once finished, it comes back to *Enabled* state. From there, event `nextSchedule()` evolves to *Waiting4Schedule* state, where the PRR will wait for event `scheduled()` invoked by the scheduler to start the proactive recovery. In both recovery states (*Reactive* or *Proactive*) the PRR invokes upon the entrance (`recovery()`) and on the exit (`recovered()`) events in the monitored device switching it off/on, respectively. Finally, note that events `enable()` and `disable()` received from the scheduler effectively prevent the PRR to monitor its device.

4.3.2 Switch Over Failing and Ping and Restore Techniques

In this section, we consider the following FTTs: *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 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 which is collecting the potentially harmful requests, and brings up a new (and clean) machine replica.

Ping And Restore provides a Monitor software component which observes the vulnerable system machines. When it finds some of these machines in an undefined state

Token colour definitions
$type\ D\ is\ \{1 \dots nDevices\}$
$type\ G\ is\ \{G_1 \dots G_{\lceil \frac{nDevices}{k} \rceil}\}$
$subtype\ G_i\ is\ \{(k \cdot (i-1)+1) \dots k \cdot i\}$
$var\ i:\ D,\ g:\ G$

Initial marking
$m_0(Enable) = \sum i \in D$
$m_0(nextGroup) = G_1$
$m_0(Idle) = 1$
$m_0(maxParallel) = k$

Functions definitions
$belonging(g:\ G) = \sum i \in G$
$cSubset(g:\ G) = \sum i \in D \vert i \ni G$
$allDevices() = \sum i \in D$

Table 4.4: CPN initial marking, token colour definition and functions.

(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.

UML Modelling Figure 4.10 illustrates the UML Sequence Diagram (UML-SD) of the *SwitchOverFailing* FTT interacting with a web server. 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 [Sousa et al., 2010a]), and then it switches off the server receiving the attacks, which has a cost of 0.6 seconds [Sousa et al., 2010a]. 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

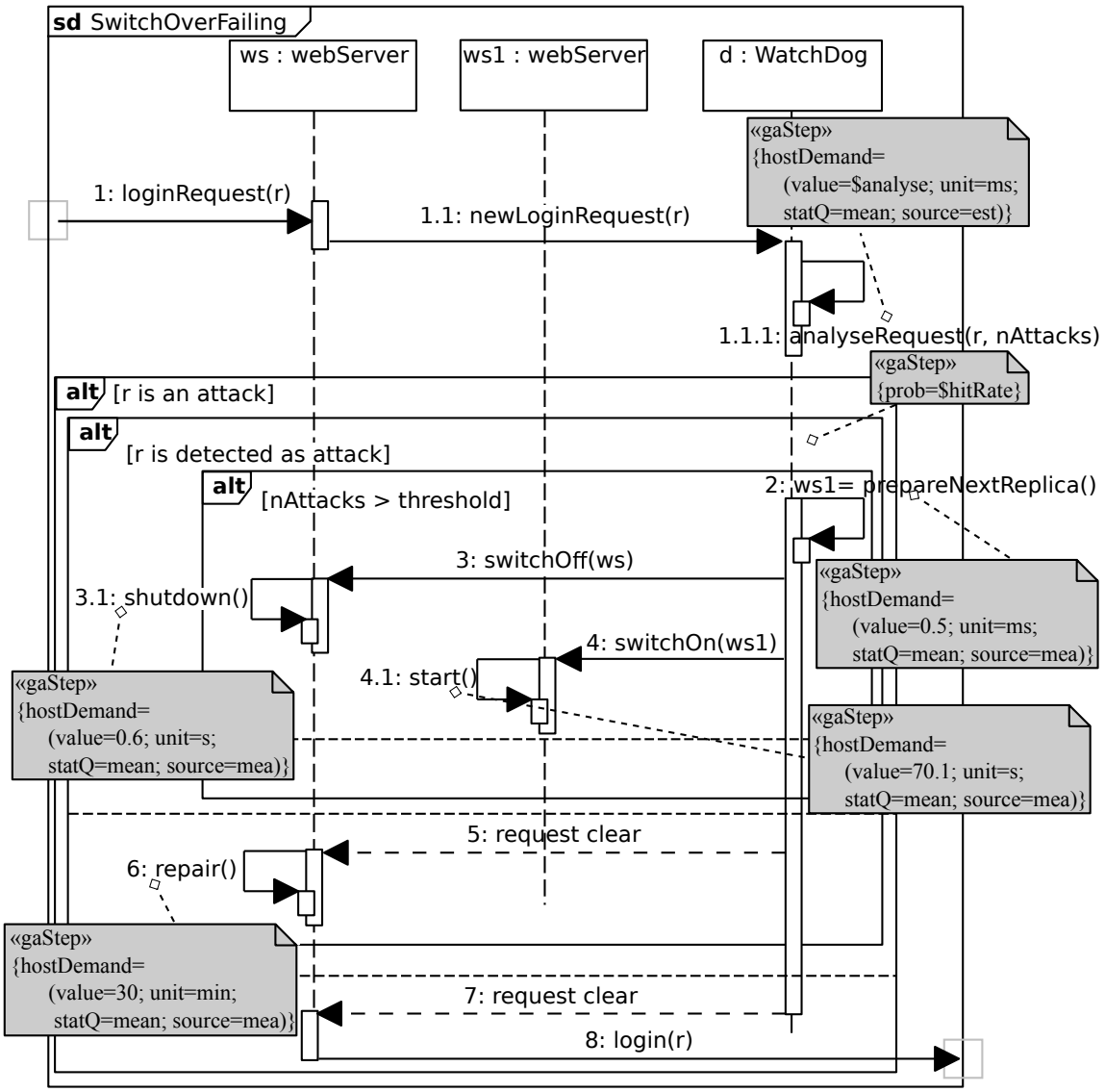


Figure 4.10: UML Sequence Diagram of the *SwitchOverFailing* Fault-Tolerant Technique.

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 4.11 depicts the UML-SD of the *Ping&Restore* FTT interacting as well with a web server. The cyclic behaviour of the monitor is as follows. It waits a certain amount of time (*$wait* ms) and then initialises a variable m, which counts the number of replicas in a non-functional state. The monitor sets a timeout having a duration of *$tOut* ms and iterates up to k machines to be recovered, asking whether each machine 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.

4.4 Concluding Remarks

Security attacks aim at 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).

In this chapter, we have proposed some models that represent common FT techniques, with the idea of combining such models with software behavioural designs. The combined model is useful for dependability assessment, as it will be shown in Chapter 5. The key point we look for is to gain a "library" of UML models representing FT techniques ready to use in critical designs.

The use of our approach should otherwise bring several benefits from the point of view of a software engineer. The easy integration of FT techniques into software designs and the existence of such "library" may allow to test different techniques for the same design to find the ones fitting better. Such "library" will also free the engineer of worrying about how to model FT and concentrate on the problem domain. Finally, it is well-known that the use of formal models early in the life-cycle to prove requirements is less expensive than other approaches.

As future work, we plan to develop a plug-in for common UML design tools, such as ArgoUML, Visual Paradigm or MagicDraw, which incorporates the UML FT models introduced in this chapter, thus to provide guidelines to software designers about the best choices of Fault-Tolerant techniques and security mechanisms for the attacks systems may suffer.

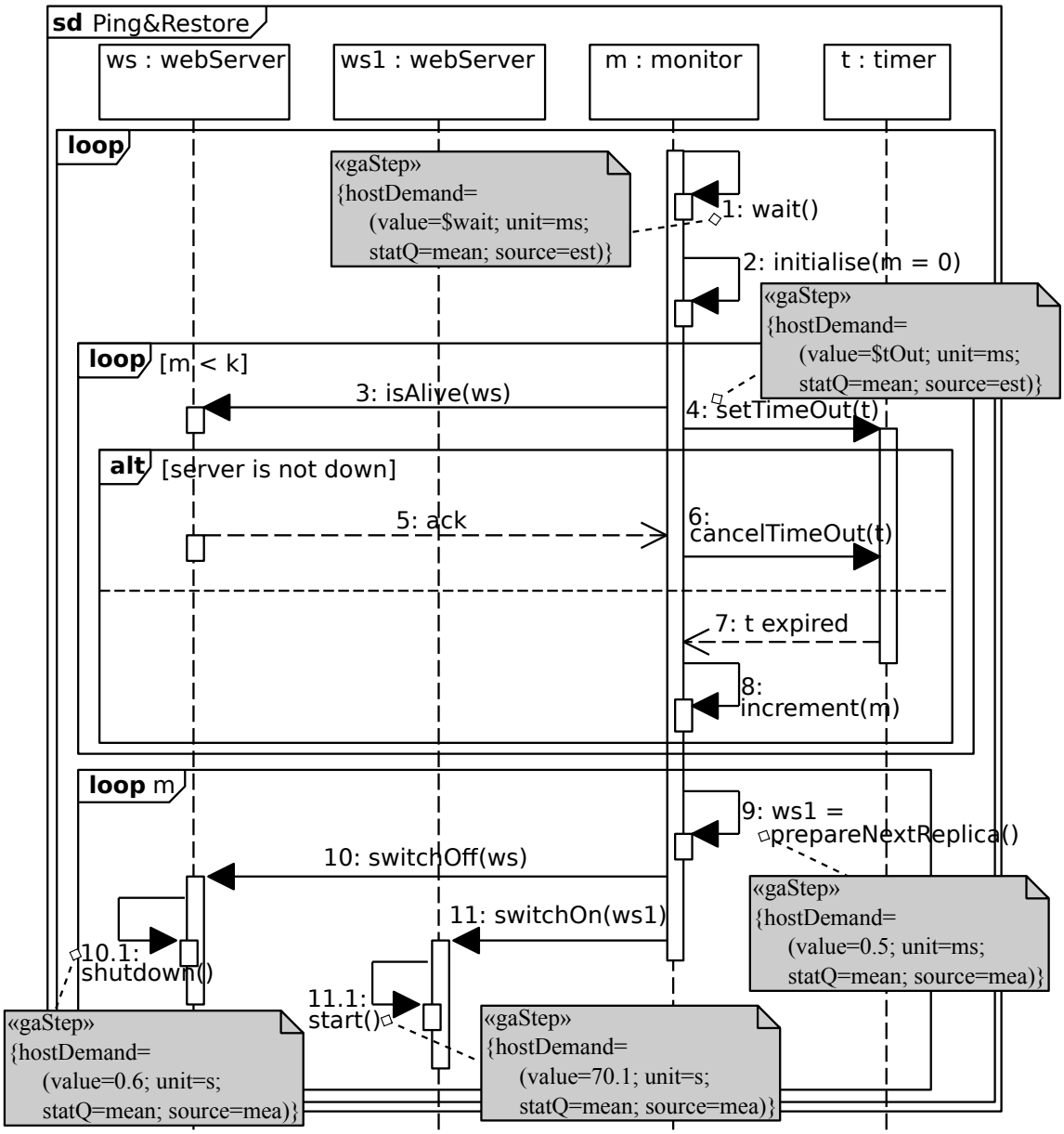


Figure 4.11: UML Sequence Diagram of the *Ping&Restore* Fault-Tolerant Technique.

Chapter 5

Model-Based Performance Prediction of Critical Systems

This chapter introduces a model-based methodology for performance prediction of critical systems which combine Fault-Tolerant Techniques (FTTs), such as recovery procedures, and/or Security Mechanisms (SMs), such as encryption of data, in order to react to intrusions. The proposed methodology was originally published in [Rodríguez et al., 2012d].

5.1 Motivation

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 [Garber, 2000] 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 [Menascé, 2003, Menascé and Virgilio, 2000]. 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.

The work in this chapter steps 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 (already introduced in Section 4.3) 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 [Rodríguez and Merseguer, 2010, Cortellessa et al., 2010a, Cortellessa and Trubiani, 2008], where we introduced a preliminary set of models aimed at representing the most common security strategies: models for FTTs have been introduced in [Rodríguez and Merseguer, 2010], whereas models for SMs have been presented in [Cortellessa et al., 2010a]. This work jointly considers FTTs and SMs 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) [OMG, 2005] for software modelling and Generalized Stochastic Petri Nets (GSPNs) [Ajmone Marsan et al., 1995] 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 [Lagarde et al., 2007,Selic, 2007]. A UML profile defines a set of stereotypes and tagged-values which are used to extend its semantic. In this work, we use two profiles: (i) the Modelling and Analysis of Real-Time and Embedded Systems (MARTE) profile [OMG, 2009] for the specification of performance properties that enable the performance analysis; (ii) and the Security Analysis and Modelling (SecAM) profile [Rodríguez et al., 2010] (see Section 3.2) 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 [Gómez-Martínez and Merseguer, 2006], ArgoPN [Delatour and de Lamotte, 2003] or ArgoPerformance [Distefano et al., 2011].

In the following, we firstly introduce the SMs that we consider for this work (namely, *Encryption*, *Decryption*, *Digital Signature* and *Verification*). The FTTs that we consider for this work have been previously introduced in Section 4.3. Lastly, we introduce a model-based methodology to quantify the security-performance trade-off in critical systems where FTTs and SMs are considered. Chapter 9 introduces a case study where this methodology is applied.

5.2 Security Mechanisms

The Security Mechanisms (SMs) were initially introduced in [Cortellessa and Trubiani, 2008, Cortellessa et al., 2010a], where Cortellesa and Trubiani introduced a set of UML models representing the most common security mechanisms. The SMs that we consider here 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.

The models of the aforementioned security mechanisms are not reported, for further details please refer to [Cortellessa et al., 2010a].

5.3 A Model-Based Methodology to Quantify Security-Performance Trade-off

In this section we recall the model-based methodology presented in [Rodríguez et al., 2012d] that allows to quantify the trade-off between the security strategies previously introduced to cope with the security attacks and the consequent performance degradation.

In Figure 5.1 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 Application Model* that is a static and dynamic representation of a software system. For the sake of simplicity, 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 [OMG, 2009] 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 [Rodríguez et al., 2010,Rodríguez et al., 2011] that en-

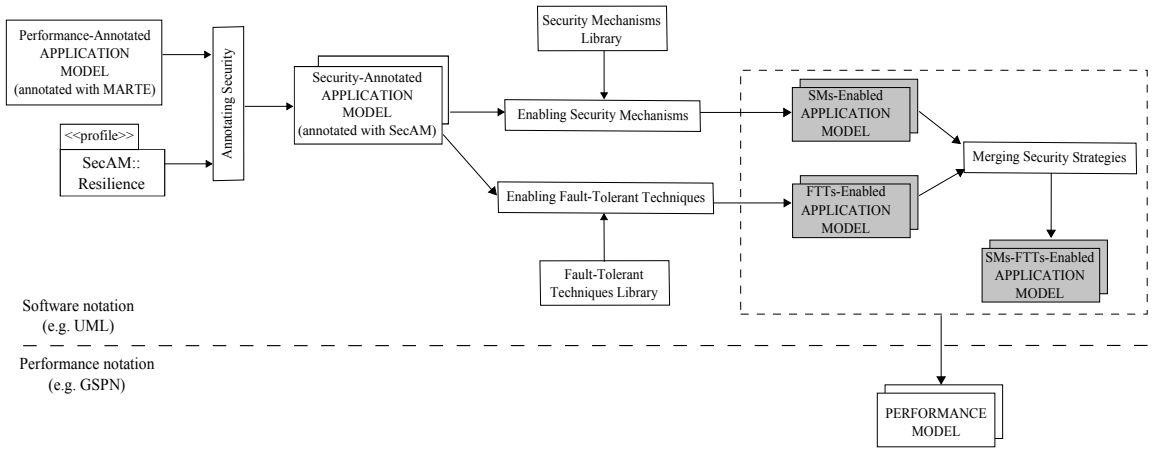


Figure 5.1: A process to estimate the system performance while adding Security Mechanisms and Fault-Tolerant Techniques.

ables the specification of attacks, vulnerabilities and intrusions in UML models (see Section 3.2.1 for more details). 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 [Cortellessa et al., 2010a]. 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 5.2).

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 4.3.2).

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 5.1 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` [Rodríguez et al., 2012a] simulator (Chapter 11 introduces `PeabraiN` in more detail), 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 5.1 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.

This model-based approach is put on evidence in a case study introduced in Chapter 9.

5.4 Concluding Remarks

In this chapter we provided a model-based methodology able to quantitatively estimate the system performance while introducing Fault-Tolerant Techniques (FTTs) and/or Security Mechanisms (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

[1]Well consolidated techniques have been exploited to transform software models (e.g. UML models) into performance models (e.g. GSPN), see [Balsamo et al., 2004] for an extensive survey on this topic.

meeting performance requirements.

There exist many security techniques that may affect system performance, such as the use of firewalls, security protocols, remote logging, etc. In this work, we have 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 trade-offs, for example it would be relevant to study the trade-off 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.

Part II

Performance Analysis

Chapter 6

Strategies for Upper Throughput Bound Computation in Petri Nets

Nothing happens in the universe
that does not have a sense of either
certain maximum or minimum.
(Leonhard Euler)

This chapter summarises the main contributions of this dissertation related to upper throughput computation of Stochastic Marked Graphs (SMGs) and of a special class of Petri net; more precisely, Stochastic Process Petri net (SPPN). We provide different strategies for computing upper throughput bounds that are more accurate, i.e., closer to the real throughput of the system, that the bounds that can be achieved with other methods (see Section 1.2). The main outcomes of this chapter have been mainly published in [Rodríguez and Júlvez, 2010] and [Rodríguez et al., 2013a].

6.1 Motivation

As it is claimed in Section 1.1, many artificial systems, such as the Fault-Tolerant (FT) systems that we consider in this thesis, can be naturally modelled as Discrete Event Systems (DES). Unfortunately, these systems are usually large and this makes the exact computation of their performance a highly complex computational task. The main reason for this complexity is the well-known state explosion problem. As a result, a task that requires an exhaustive state space exploration becomes unachievable in a reasonable time for large systems.

A way to overcome the state explosion problem is to provide performance bounds [Ramchandani, 1974, Chiola et al., 1993, Campos et al., 1992, Liu, 1995]. The use of performance bounds, on which our approaches are based, avoids the necessity of calculating the whole state space. The advantage of using performance bound computation is

the reduced computing time, but its drawback is the difficulty of assessing how accurate the computed bound is with respect to the real system performance.

This chapter explores the issue of upper throughput bound for Stochastic Marked Graphs (SMGs) and Stochastic Process Petri nets (SPPNs), and provides two different strategies for getting an improved upper throughput bound than the ones that those achieved in previous works [Chiola et al., 1993, Campos et al., 1992].

First of all, we introduce some basic concepts needed to follow the rest of the chapter, such as *tight marking*, which allows us to compute easily slacks of places with respect to the critical cycle (i.e., the slowest cycle in the system). Then, we introduce an iterative algorithm to obtain performance bounds on SMGs [Rodríguez and Júlvez, 2010] that are sharper, i.e., closer to the real performance, than the ones we can currently compute with some of the works previously mentioned in Section 1.2. In a few words, our method works as follows. Firstly, the algorithm calculates the most restrictive cycle (also called *bottleneck*) by applying well-known methodologies. Then, it adds to the bottleneck cycle those sets of places that are more likely to constraint the throughput of the system. The process of adding sets of places is repeated until the throughput of the resulting net does not vary significantly. Such a throughput cannot increase during the addition process since more constraints are added to the SMG. The proposed algorithm produces the following outputs:

- a performance bound for the steady-state throughput of a stochastic Marked Graph and
- a subnet representing the bottleneck of the stochastic Marked Graph.

As it will be explained, the method makes intensive use of linear programming techniques for which polynomial complexity algorithms exist. Given that the performance bound is refined in each iteration, the accuracy of the final bound depends on the number of iterations to be performed. The obtained results show that the proposed method offers a good trade-off between accuracy and computational complexity load.

Lastly, such an iterative algorithm is extended to be able to deal with Process Petri nets [Rodríguez et al., 2013a]. As in the previous algorithm, the strategy for getting sharper (i.e., closer to the real throughput) upper throughput bounds is based on the computation of *bottlenecks*. It calculates in a first step the slowest part of the system, that is, the initial bottleneck of the system. After that, in each iteration the most likely part of the system to be constraining the current bottleneck is calculated, and the union of both parts is considered to calculate the new upper throughput bound.

6.2 Little's Law and Upper Throughput Bounds

The Little's formula [Little, 1961] involves the average number of customers L in the system, the throughput, λ, and the average time spent by a customer within the system,

W.

$$L = \lambda \cdot W \tag{6.1}$$

Let p be a place such that $|p^\bullet| = 1$, and $p^\bullet = \{t\}$, then the pair (p,t) can be seen as a simple queueing system to which, if the limits of *average marking* and *steady-state throughput* exist, Little's formula can be directly applied [Campos and Silva, 1992]:

$$\overline{\mathbf{m}}(p) = (\mathbf{Pre}(p,t) \cdot \chi(t)) \cdot \upsilon(p) \tag{6.2}$$

where $\mathbf{Pre}(p,t) \cdot \chi(t)$ is the output rate of tokens from place p, which in steady state is equal to the input rate, and $\upsilon(p)$ is the average residence time at place p, i.e., the average time spent by a token in place p.

The average residence time, $\upsilon(p)$, is the sum of the average waiting time due to a possible synchronisation and the average service time, $\delta(t)$. Therefore, equation (6.2) becomes:

$$\overline{\mathbf{m}}(p) = (\mathbf{Pre}(p,t) \cdot \chi(t)) \cdot \upsilon(p) \geq (\mathbf{Pre}(p,t) \cdot \chi(t)) \cdot \delta(t) \tag{6.3}$$

where the service time $\delta(t)$ is a lower bound for the average residence time $\upsilon(p)$, i.e., $\delta(t) \leq \upsilon(p)$, since place p has only one output transition. Given that conflicting transitions are assumed to be immediate, equation (6.3) can also be applied to any pair (p,t), $t \in p^\bullet$ and t being a transition in conflict. Hence, the following system of inequalities can be derived [Campos and Silva, 1992] from (2.3) and (6.3):

$$\mathbf{\Gamma}(t_i) \cdot \overline{\mathbf{m}} \geq \mathbf{Pre} \cdot \mathbf{D}^{t_i} \tag{6.4}$$

where $\mathbf{\Gamma}(t_i)$ is the average interfiring time of transition t_i and $\mathbf{D}^{t_i}$ is the vector of *average service demands of transitions*, $\mathbf{D}^{t_i}(t) = \delta(t) \cdot \mathbf{v}^{t_i}(t)$ (the vector of visit ratios $\mathbf{v}^{t_i}$ is normalised for transition t_i). In the following, we omit the superindex t_i in $\mathbf{D}^{t_i}$ for clarity.

Let us notice that strongly connected SMGs have a single minimal t-semiflow that is equal to $\mathbf{1}$. This implies that the steady-state throughput is the same for every transition. Therefore, a single scalar variable $\Theta = \dfrac{1}{\Gamma}$ suffices to express the throughput bound to be computed for all transitions.

Proposition 1 *The solution* $\mathbf{\Theta}$ *of the following LPP provides an upper bound for the steady-state throughput of the transitions of a strongly connected Freely Related T-semiflows (FRT) net [Chiola et al., 1993]:*

$$\begin{aligned} &Maximize\ \Theta: \\ &\quad \overline{\mathbf{m}}(p) \geq \delta(p^\bullet) \cdot \Theta \quad \forall p \in P && \text{(6.5a)} \\ &\quad \overline{\mathbf{m}} = \mathbf{m_0} + \mathbf{C} \cdot \sigma && \text{(6.5b)} \\ &\quad \sigma \geq 0 && \text{(6.5c)} \end{aligned}$$

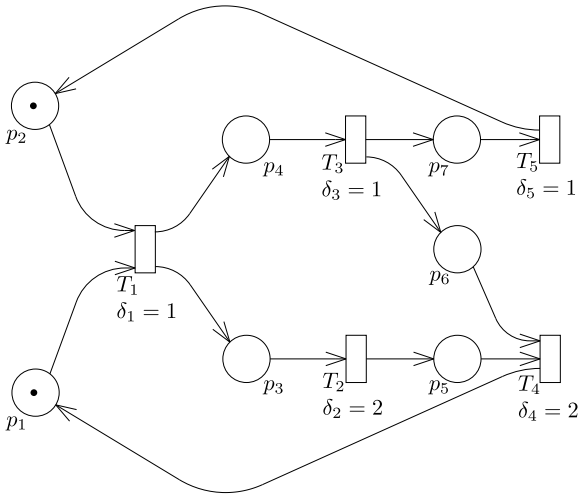


Figure 6.1: Example MG.

The first constraint (6.5a) is obtained from (6.3), while the second and third constraints (6.5b), (6.5c) establish that $\overline{\mathbf{m}}$ must be a solution of the state equation. The value of Θ is the exact throughput in the particular case of timed MG with deterministic delays associated to the firing delays [Ramchandani, 1974, Ramamoorthy and Ho, 1980].

The LP problem (LPP) in (6.5) can be transformed in its dual, which after some manipulations becomes in a LPP to compute a lower bound for the *average inter-firing time* of transition t_i, $\mathbf{\Gamma^{lb}}(t_i)$, [Campos and Silva, 1992]:

$$\begin{aligned} \mathbf{\Gamma}(t_i) \geq \mathbf{\Gamma^{lb}}(t_i) = \; & \mathit{maximum}\ \mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D} \\ & \mathit{subject\ to}\ \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & \mathbf{y} \cdot \mathbf{m_0} = 1 \\ & \mathbf{y} \geq \mathbf{0} \end{aligned} \tag{6.6}$$

As a side product of the solution of (6.6), $\mathbf{y}$ represents the *slowest* p-semiflow of the system, thus LPP (6.6) can also be seen as a search for the most constraining p-semiflow. This p-semiflow will be the one with the highest ratio $\dfrac{\mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D}}{\mathbf{y} \cdot \mathbf{m_0}}$. Sn upper bound $\Theta(t_i)$ for the steady-state throughput can be calculated as the inverse of the lower bound for the average inter-firing time $\mathbf{\Gamma^{lb}}(t_i)$, that is, $\Theta(t_i) = \dfrac{1}{\mathbf{\Gamma^{lb}}(t_i)}$.

For instance, let us consider the Marked Graph (MG) shown in Figure 6.1. The initial marking is: $\mathbf{m}(p_1) = \mathbf{m}(p_2) = 1$ and the rest of places have marking equal to 0. We assume that the firing delay of each transition follows an exponential distribution with mean $\delta_1 = \delta_3 = \delta_5 = 1, \delta_2 = \delta_4 = 2$, respectively. The net has three cycles: $\{p_1, p_3, p_5\}$, $\{p_1, p_4, p_6\}$

and $\{p_2, p_4, p_7\}$. The token/delay ratio of each cycle is $\frac{1}{5}$, $\frac{1}{4}$ and $\frac{1}{3}$, respectively. The critical cycle, or bottleneck, is the one with minimum token to delay ratio, thus in our case, the bottleneck cycle is the one composed of places $\{p_1, p_3, p_5\}$ whose throughput is equal to $\frac{1}{5}$. Hence, the initial throughput bound is $\frac{1}{5}$ and the initial bottleneck is $\mathbf{y}_{lb} = \{p_1, p_3, p_5\}$.

Assume again p be a place such that $|p^\bullet| = 1$, and $p^\bullet = \{t\}$. The equation (6.5a) can be also expressed as follows:

$$\overline{\mathbf{m}}(p) = \delta(t) \cdot \Theta(t) + \mu(p)$$

where $\mu(p) \geq 0$ is the *slack* of place p. For every place p in the critical cycle (i.e., bottleneck) of a SMG, it necessarily holds that $\mu(p) = 0$. For example, the slacks of the places of the SMG in Figure 6.1 are $\mu(p_1) = \mu(p_3) = \mu(p_5) = 0$, $\mu(p_2) = 0.16$, $\mu(p_4) = 0.08$, $\mu(p_6) = 0.12$ and $\mu(p_7) = 0.16$. In general, the same optimal value of the objective function in LPP (6.5) can be achieved for different slack vectors. In fact, the particular value of vector μ will depend on the algorithm used by the LP solver.

6.2.1 Tight Marking

This section takes advantage of the degree of freedom of slacks in order to produce a marking, called *tight marking* and denoted $\tilde{\mathbf{m}}$, such that each transition has at least one input place with null slack. This marking will greatly ease the task of adding to the initial bottleneck cycle those cycles that have low ratio token/delay.

Definition 10 *A marking vector $\tilde{\mathbf{m}} \in \mathbb{R}^{|P|}$ is called a* tight *marking vector of a SMG if it satisfies:*

$$\tilde{\mathbf{m}} = \mathbf{m_0} + \mathbf{C} \cdot \sigma \tag{6.7a}$$

$$\forall\ p: \quad \tilde{\mathbf{m}}(p) \geq \delta(p^\bullet) \cdot \Theta \tag{6.7b}$$

$$\forall\ t\ \exists\ p \in {}^\bullet t: \quad \tilde{\mathbf{m}}(p) = \delta(p^\bullet) \cdot \Theta \tag{6.7c}$$

where $\tilde{\mathbf{m}} \in \mathbb{R}^{|P|}$, $\sigma \in \mathbb{R}^{|T|}$, and $\Theta = \frac{1}{\mathbf{\Gamma^{lb}}}$ is the solution of (6.6)*. A place p satisfying the condition $\tilde{\mathbf{m}}(p) = \delta(p^\bullet) \cdot \Theta$ is called* tight*.*

Since the places of the critical cycle do not have slack, they fulfil (6.7c) and hence are tight. On the other hand, non-critical places may have some positive slack. The tight marking exploits this flexibility by adjusting the marking in such a way that each transition has at least one input place that is tight.

It can be shown that a tight marking exists for each SMG [Carmona et al., 2009]. Moreover it can be computed efficiently by solving an LPP.

Proposition 2 *[Carmona et al., 2009] A tight marking of a SMG can be computed by solving the following LPP:*

$$\begin{aligned} &Maximize\ \Sigma\sigma: \\ &\quad \delta(p^\bullet)\cdot\Theta \le \tilde{\mathbf{m}}(p) \quad \textit{for every } p\in P \\ &\quad \tilde{\mathbf{m}} = \mathbf{m_0} + \mathbf{C}\cdot\sigma \\ &\quad \sigma(t_p) = k \end{aligned} \tag{6.8}$$

where t_p is a transition that belongs to a critical cycle and k is any real constant number.

The proof of the Proposition 2 can be found in [Carmona et al., 2009]. Since we are dealing with MGs, each row of the incidence matrix $\mathbf{C}$ contains a single positive (1) and a single negative (-1) value, while all other values are zeros. Therefore, the first two constraints of (6.8) can be transformed into a system of *difference constraints* and hence the LPP (6.8) can be efficiently solved by using the Bellman-Ford algorithm [Cormen et al., 2001].

Recalling the SMG shown in Figure 6.1, if we calculate the tight marking we obtain $\tilde{\mathbf{m}}(p_1) = 0.2$, $\tilde{\mathbf{m}}(p_2) = 0.6$, $\tilde{\mathbf{m}}(p_3) = 0.4$, $\tilde{\mathbf{m}}(p_4) = 0.2$, $\tilde{\mathbf{m}}(p_5) = 0.4$, $\tilde{\mathbf{m}}(p_6) = 0.6$, $\tilde{\mathbf{m}}(p_7) = 0.2$.

6.3 Regrowing Strategy for Stochastic Marked Graphs

This section presents an iterative strategy to grow the critical cycle and to compute an upper throughput bound in SMGs. The idea of the strategy is to add in each iteration the cycle that is potentially more restrictive than the others and then calculate the throughput. Such a throughput cannot be higher than the one in the previous iteration, since more constraints have been added to the net. The iteration process will stop when no significant improvement of the bound is achieved.

Algorithm 1 represents the overall regrowing strategy used to compute throughput bounds. The algorithm needs as input data the Stochastic Marked Graph (SMG) to be analysed, $\langle \mathcal{N}, \delta \rangle$, and the degree of precision ($\varepsilon > 0$) to be achieved. As output data, the upper throughput bound, Θ, and the bottleneck cycle of the SMG, $sccN'$, are obtained.

Firstly, an upper throughput bound of $\langle \mathcal{N}, \delta \rangle$ is calculated according to (6.6), which will be the initial upper bound. Then, the tight marking of the system is computed by using the LPP shown in (6.8). The vector of slacks μ is computed in step 3. The iteration process (steps 7–14) is repeated until no significant improvement is achieved with respect to the last iteration.

In steps 8–11, a new set of places and transitions is added to the current bottleneck. To achieve this, steps 8–9 look for the place q that is connected to the current bottleneck $sccN'$, i.e., $q^\bullet \in sccN'$, and has minimum slack. Then steps 10–11 build the new bottleneck by adding place q and the tight places that connect the current bottleneck to q. For brevity, in the algorithm we use $p \in \mathcal{N}$ ($p^\bullet \in \mathcal{N}$) to denote that a place p (transition $p^\bullet$) is contained

Input: $\langle \mathcal{N}, \delta \rangle$, ε
Output: Θ, $sccN'$

1 Θ = Upper throughput bound of $\mathcal{N}$ according to (6.6)
2 $\tilde{\mathbf{m}}$ = Tight marking according to (6.8)
3 $\mu(p) = \tilde{\mathbf{m}}(p) - \delta(p^\bullet) \cdot \Theta, \quad \forall p \in P$
4 $\mathcal{N}'$ = Graph resulting of removing from $\mathcal{N}$ every arc $\{p, p^\bullet\}$ such that $\mu(p) > 0$
5 $sccN'$= Strongly connected component of $\mathcal{N}'$
6 $\Theta' = 0$
7 **while** $\left(\frac{\Theta - \Theta'}{\Theta} \geq \varepsilon\right)$ **do**
8 $\quad Q = \{q | q \in P, q \notin \mathcal{N}', q^\bullet \in sccN'\}$
9 $\quad p_m = \{q | \mu(q) = \min_{p \in Q} \mu(p)\}$
10 $\quad \mathcal{N}'$ = Graph resulting of adding arc $\{p_m, p_m^\bullet\}$ to $\mathcal{N}'$ where $\{p_m, p_m^\bullet\} \in \mathcal{N}$
11 $\quad sccN'$= Strongly connected component of $\mathcal{N}'$
12 $\quad \Theta' = \Theta$
13 $\quad \Theta$ = Throughput of $sccN'$
14 **end**

Algorithm 1: The regrowing strategy algorithm.

in the set of places (transitions) of $\mathcal{N}$. When there exist several identical critical cycles, i.e, with the same token to delay ratio, steps 5 and 11 choose any of them.

In step 13, the throughput of the new bottleneck is taken as the new upper bound. In the next iteration, this new upper bound will be compared with the previous one in order to, depending on the degree of improvement achieved, either continue or finish the iteration process.

Let us illustrate how the algorithm 1 works by applying it to the SMG depicted in Figure 6.2. The delays are $\delta_1 = 1.2, \delta_2 = 1, \delta_3 = 1.5, \delta_4 = \delta_5 = 1, \delta_6 = 0.75, \delta_7 = 1, \delta_8 = 1.25$ and $\delta_9 = 0.5$, and the initial critical cycle is composed by $\{P_{cb}, T_{cb}\} = \{\{p_2, p_4\}, \{t_1, t_3\}\}$. The throughput bound of the critical cycle is $\Theta_{cb} = 0.370370$ and the places which are connected (through a transition $t \in T$) to the critical cycle are p_1 and p_{14}, having slacks $\mu(p_1) = 0.1852$ and $\mu(p_{14}) = 1.0556$. Hence, the place with minimum slack is p_1. By regrowing the current bottleneck the new one is obtained, composed by $\{P_{cb'}, T_{cb'}\} = \{\{p_1, p_2, p_3, p_4\}, \{t_1, t_2, t_3\}\}$, which has a throughput of $\Theta_{cb'} = 0.322581$, which is 12.9% lower than the throughput of the previously bottleneck $\{P_{cb}, T_{cb}\}$.

Let us assume that $\varepsilon = 0.001$. As the relative difference between Θ_{cb} and $\Theta_{cb'}$ is 0.12903 (as commented previously), the iteration process carries on. At this moment, the places connected to the current bottleneck are p_{10} and p_{14}. The addition of the place p_{10} which has minimum slack produces a new bottleneck compounded of $\{\{p_1, p_2, p_3, p_4, p_6, p_7, p_8, p_9, p_{10}\}, \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}\}$, being the new throughput $\Theta =$

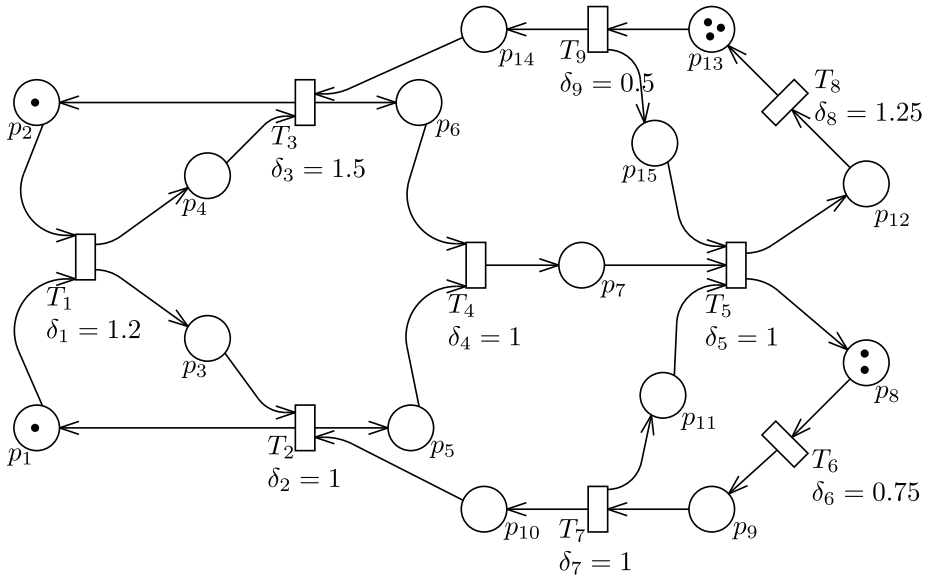


Figure 6.2: Another MG example.

0.297914, which is an improvement of 7.647% with respect to the previous bottleneck $\{P_{cb'}, T_{cb'}\}$ and 19.563% with respect to the original bottleneck $\{P_{cb}, T_{cb}\}$.

Again, a new regrowing is possible because the relative difference is greater than ε. In this case, the candidate places to be chosen are p_5, p_{11} and p_{14}, which have slacks $\mu(p_5) = 0.0556$, $\mu(p_{11}) = 0.9815$ and $\mu(p_{14}) = 1.0556$. The addition of p_5 produces a new bottleneck with $\Theta = 0.297914$, which is an improvement of 3.193% with respect to the previous bottleneck. For the next regrowing, the candidate places are p_{11}, p_{14} and p_{15}. By adding the place p_{11} ($\mu(p_{11}) = 0.9815$) we obtain a bottleneck whose relative throughput is lower than ε with respect to the previous bottleneck, thus, the algorithm finishes. In summary, after four iterations, the throughput bound obtained is 22.132% lower than the original Θ calculated by LPP in (6.6).

6.3.1 Experiments and Discussion

In this section we test the algorithm given in previous section on a set of SMGs of the ISCAS benchmarking [Brglez et al., 1989]. After applying the regrowing strategy, the obtained results are discussed.

Experimental Setting

The structure of the SMGs to be analysed is obtained from the strongly connected components of the ISCAS graphs. The initial marking of each place is a natural number which

is randomly selected in the interval $[1 \dots 10]$. The value of the $\delta(t)$ of each transition t is a real number randomly selected from the interval $[0.1 \dots 1]$. The overall strategy has been implemented on MATLAB[1], while simulations of SMGs have been performed by the GreatSPN [Baarir et al., 2009] simulation tool using a confidence level of 99% and an accuracy of 1%. The simulations have been run in a machine with a Pentium IV 3.6GHz processor and 2GB DDR2 533MHz RAM.

[1] http://www.mathworks.com/products/matlab/

Graph	Size		% Size		Regrowing steps	Initial thr. bound	Θ
	$\|P\|$	$\|T\|$	$\|P'\|$ (%)	$\|T'\|$ (%)			
s1423	1107	792	79 (7.13%)	76 (9.59%)	3	0.236010	0.235213 (0.34%)
s1488	1567	1128	91 (5.8%)	86 (7.62%)	6	0.201300	0.173127 (13.99%)
s208	27	24	27 (100%)	24 (100%)	3	0.409390	0.377683 (7.75%)
s27	54	44	19 (35.18%)	18 (40.9%)	1	0.305960	0.304987 (0.31%)
s349	187	146	26 (13.9%)	24 (16.44%)	2	0.340320	0.327867 (3.66%)
s444	92	68	14 (15.21%)	12 (17.64%)	2	0.181670	0.181260 (0.22%)
s510	1038	734	45 (4.33%)	40 (5.45%)	5	0.133030	0.117819 (11.43%)
s526	113	92	18 (15.93%)	16 (17.39%)	2	0.313490	0.305860 (2.43%)
s713	271	208	11 (4.06%)	10 (4.8%)	1	0.428720	0.427840 (0.2%)
s820	1162	848	40 (3.44%)	38 (4.48%)	2	0.161060	0.147483 (8.43%)
s832	1293	948	84 (6.5%)	78 (12.04%)	5	0.239429	0.208798 (12.79%)
s953	415	312	88 (11.36%)	82 (26.28%)	6	0.369214	0.337811 (8.50%)

Table 6.1: Experiment results showing improvement of upper bound.

Graph	Original thr. CPU time (s)	Θ CPU time (s)	Original thr.	Θ	% thr.
s1423	59948.980	8.283	0.222720	0.235270	5.63%
s1488	36717.156	7.165	0.168760	0.172154	2.01%
s208	0.492	0.492	0.376892	0.376892	0%
s27	2166.002	0.954	0.305082	0.306166	0.35%
s349	141.210	0.441	0.328340	0.327398	−0.28%
s444	2278.231	0.205	0.181069	0.181260	0.11%
s510	13669.814	1.358	0.117500	0.118040	0.46%
s526	129.181	0.344	0.270010	0.305860	13.27%
s713	628.503	0.405	0.411630	0.427840	3.94%
s820	20775.811	0.788	0.144770	0.147699	2.02%
s832	16165.863	1.914	0.196920	0.208873	6.07%
s953	453.850	19.155	0.327910	0.338644	3.27%

Table 6.2: Graph throughput and CPU time comparative.

Experimental Results

Table 6.1 shows the obtained results by our approach. The degree of accuracy for Algorithm 1 has been set to $\varepsilon = 0.005$. The first column is the graph name, followed by its size (number of places, $|P|$, and transitions, $|T|$). In the next column, it is shown the size of the net $sccN'$ ($|P'|$,$|T'|$) produced by the algorithm. The column *Regrowing steps* shows the number of regrowing steps needed by the algorithm. The last columns of Table 6.1 show the initial upper throughput bound calculated by using the LPP (6.6), and the improved upper throughput bound, Θ, computed by the algorithm. Such a bound is computed by solving the Markov Chain associated to $sccN'$ when it is handleable by the computer, and by simulation otherwise (see [Ajmone Marsan et al., 1995] for an example of Markov Chain analysis). The last column shows the percentage of improvement with respect to the original upper throughput bound.

As it can be seen, our method is able to get a sharper upper bound than the original bound in a few regrowing steps, and the improvement varies from 0.2% (which indicates that the original upper bound is already very tight) up to 14%. We conjecture that the improvement depends on the structure of the graph. It is also worth mentioning that our approach uses a very low percentage of the size of the original graph, in most of cases this percentage is lower than 10%.

Table 6.2 summarises a comparative between the original throughput bound and the improved upper throughput bound and between the CPU time needed for both computations. The first column is the graph name, followed by the CPU time consumed to calculate the original throughput and to calculate the improved upper throughput bound Θ. The

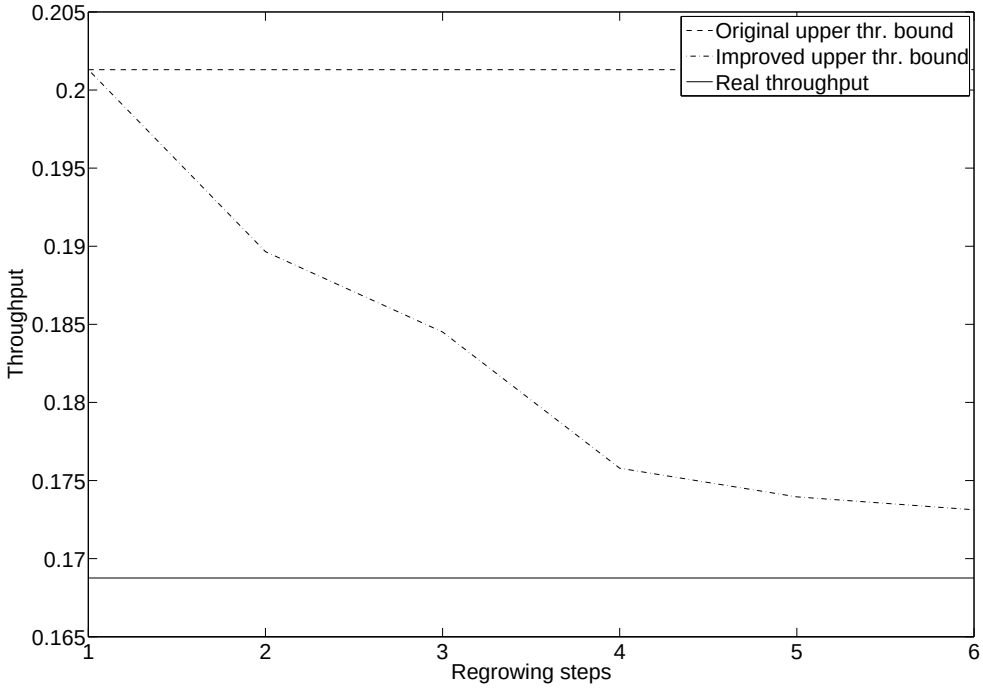


Figure 6.3: Throughput of graph `s1488`.

next columns are its original throughput and the improved upper throughput bound, Θ. The last column shows the relative error of Θ with respect to the original throughput. Due to the size of original graphs, the task of calculating their throughput is an unfeasible task in reasonable time. For this reason, the simulation parameters have been set to a confidence level of 95% and an accuracy of 4%. Owing to this reason, the values of Θ in Table 6.1 and in Table 6.2 can slightly vary. The negative relative errors are caused by such confidence level and accuracy degree. As it can be observed in the results shown in Table 6.2, the improved throughput bound varies from a value really close to the real throughput, to a value which is 13% over the real throughput. The latter case, which deserves further analysis, might be due to the existence of slow cycles far away from the critical cycle.

Finally, Figure 6.3 shows the real throughput of the graph `s1488` (solid line), the original upper throughput bound (dashed line, result of LPP (6.6)) and the improved upper throughput bound (dot-dashed line) in each step of the strategy. As it can be observed, the improved bound gets close to the real throughput after few steps.

The main results that can be extracted from both tables can be summarised as follows:

- a sharp upper bound is obtained after few regrowing steps;

- the size of the obtained bottleneck is very low compared to the size of the original graph and
- the obtained bottleneck represents the actual constraint for the system throughput, and therefore it can be considered as a potential target to carry out performance optimisation.

6.4 Regrowing Strategy for Process Petri Nets

In this section, we extend the theory behind algorithm 1 to be able to apply to more general nets than Stochastic Marked Graphs (SMGs), namely, to Stochastic Process Petri nets (SPPNs, see Section 2.1). The LPP shown in (6.6) was the basis in [Rodríguez and Júlvez, 2010] for developing an iterative algorithm to compute upper bounds in SMGs. Unfortunately, the proposed algorithm is not applicable to more general nets than MGs, hence our search for an alternative method.

The new algorithm will follow a similar strategy. Firstly, the initial bottleneck is computed using (6.6). Then, in each iteration step the next slowest p-semiflow connected to the subnet associated to the current bottleneck is added to it.

Let us suppose the p-semiflow $\mathbf{y}^*$ represents the initial bottleneck, i.e., $\mathbf{y}^*$ is obtained from the solution of (6.6). The following constraint imposes that some other p-semiflow $\mathbf{y}$, $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$, is connected to $\mathbf{y}^*$: $\sum_{p \in V} \mathbf{y}(p) > 0$, where $V = \{v | v \in {}^\bullet(\|\mathbf{y}^*\|^\bullet) \setminus \|\mathbf{y}^*\|\}$ (that is, there exist places in the support of $\mathbf{y}$ which share output transitions with places in the support of $\mathbf{y}^*$). Hence, the p-semiflow $\mathbf{y}$ with the highest ratio $\frac{\mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D}}{\mathbf{y} \cdot \mathbf{m_0}}$ connected to $\mathbf{y}^*$ can be searched for by solving the following LPP:

$$\begin{aligned} &\textit{maximum}\ \mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D} \\ &\textit{subject to}\ \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ &\qquad \mathbf{y} \cdot \mathbf{m_0} = 1 \\ &\qquad \mathbf{y}(p) > 0,\ \forall p \in Q \\ &\qquad \sum_{p \in V} \mathbf{y}(p) > 0 \end{aligned} \tag{6.9}$$

where $V = \{v | v \in {}^\bullet(\|\mathbf{y}^*\|^\bullet) \setminus \|\mathbf{y}^*\|\}$, and $Q = \{q \in P, q \in \|\mathbf{y}^*\|\}$.

As a result of LPP (6.9), we will obtain the p-semiflow $\mathbf{y}$, which will be a linear combination of $\mathbf{y}^*$ and the next most constraining p-semiflow.

The strict inequality in (6.9) could lead us to numerical problems since the lower the value of $\sum_{p \in V} \mathbf{y}(p)$, the higher the value of the optimisation function. This issue is discussed deeply in Section 6.4.2 and also shows that the solution proposed in the following can be applied in practice. A way to solve this is by reformulating $\sum_{p \in V} \mathbf{y}(p) > 0$ into

$\sum_{p\in V} \mathbf{y}(p) \geq h$, where h is strictly positive. The problem now is to set an appropriate value for h. A high value can make constraints $\mathbf{y}\cdot\mathbf{m_0} = 1$ and $\sum_{p\in V} \mathbf{y}(p) \geq h$ incompatible leading to an infeasible LPP. A valid value of h can be obtained by searching for a real number that is lower than each component of a p-semiflow $\mathbf{y}$ that covers all places and satisfies $\mathbf{y}\cdot\mathbf{m_0} = 1$. Such a value can be obtained by means of the following LPP:

$$\begin{aligned} &maximum\ h \\ &subject\ to\ \mathbf{y}\cdot\mathbf{C} = \mathbf{0} \\ &\qquad\qquad \mathbf{y}\cdot\mathbf{m_0} = 1 \\ &\qquad\qquad \mathbf{y} \geq h\cdot\mathbf{1} \\ &\qquad\qquad h > 0 \end{aligned} \tag{6.10}$$

where $\mathbf{1}$ is a vector with all entries equal to one.

The obtained value h ensures the feasibility of the following LPP, which is just a reformulation of (6.9):

$$\begin{aligned} &maximum\ \mathbf{y}\cdot\mathbf{Pre}\cdot\mathbf{D} \\ &subject\ to\ \mathbf{y}\cdot\mathbf{C} = \mathbf{0} \\ &\qquad\qquad \mathbf{y}\cdot\mathbf{m_0} = 1 \\ &\qquad\qquad \mathbf{y}(p) \geq h,\ \forall p \in Q \\ &\qquad\qquad \sum_{p\in V} \mathbf{y}(p) \geq h \end{aligned} \tag{6.11}$$

where $V = \{v | v \in {}^\bullet(\|\mathbf{y}^*\|^\bullet) \setminus \|\mathbf{y}^*\|\}$, and $Q = \{q \in P, q \in \|\mathbf{y}^*\|\}$.

As has been said, the last constraint, $\sum_{p\in V} \mathbf{y}(p) \geq h$, imposes that the support of $\mathbf{y}$ corresponds to the p-semiflow connected to $\mathbf{y}^*$ with the highest $\dfrac{\mathbf{y}\cdot\mathbf{Pre}\cdot\mathbf{D}}{\mathbf{y}\cdot\mathbf{m_0}}$.

6.4.1 An Iterative Strategy to Compute Upper Throughput Bounds

This subsection presents an iterative strategy to obtain an improved upper throughput bound in SPPNs. In a first step, the strategy calculates the initial throughput bound of the system with the LPP (6.6) and takes the subnet associated to $\mathbf{y}$ as the initial bottleneck. In each iteration the subnet associated to the p-semiflow that is potentially more constraining than the others is added to the bottleneck. The throughput is then calculated. Note that such an addition in each iteration restricts the behaviour of the system, which implies a lower throughput. The iteration process stops when no significant improvement of the bound is achieved.

Input: $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle$, ε
Output: Θ, $\mathbf{Q}$

1 $\{\Theta, \mathbf{y}\}$ = Upper throughput bound and components of the initial bottleneck of $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle$ according to (6.6)
2 Calculate value h by solving LPP (6.10)
3 $\Theta' = 0$; $\mathbf{Q} = \{p \in P, p \in \|\mathbf{y}\|\}$
4 **while** $\dfrac{\Theta - \Theta'}{\Theta} \geq \varepsilon$ **and** $\mathbf{Q} \neq P$ **do**
5 $\quad \mathbf{V} = \{v | v \in {}^\bullet(\mathbf{Q}^\bullet) \setminus \mathbf{Q}\}$
6 $\quad$

$$
\begin{aligned}
&\textit{maximum}\ \mathbf{y}' \cdot \mathbf{Pre} \cdot \mathbf{D} \\
&\textit{subject to}\ \mathbf{y}' \cdot \mathbf{C} = \mathbf{0} \\
&\qquad \mathbf{y}' \cdot \mathbf{m_0} = 1 \\
&\qquad \mathbf{y}'(p) \geq h,\ \forall p \in \mathbf{Q} \\
&\qquad \sum_{p \in V} \mathbf{y}'(p) \geq h
\end{aligned}
$$

7 $\quad \Theta' = \Theta$
8 $\quad \Theta$ = Throughput of the net composed by the p-semiflow $\mathbf{y}'$
9 $\quad \mathbf{Q} = \{p \in P, p \in \|\mathbf{y}'\|\}$
10 **end**

Algorithm 2: The iterative strategy algorithm for computing upper throughput bounds.

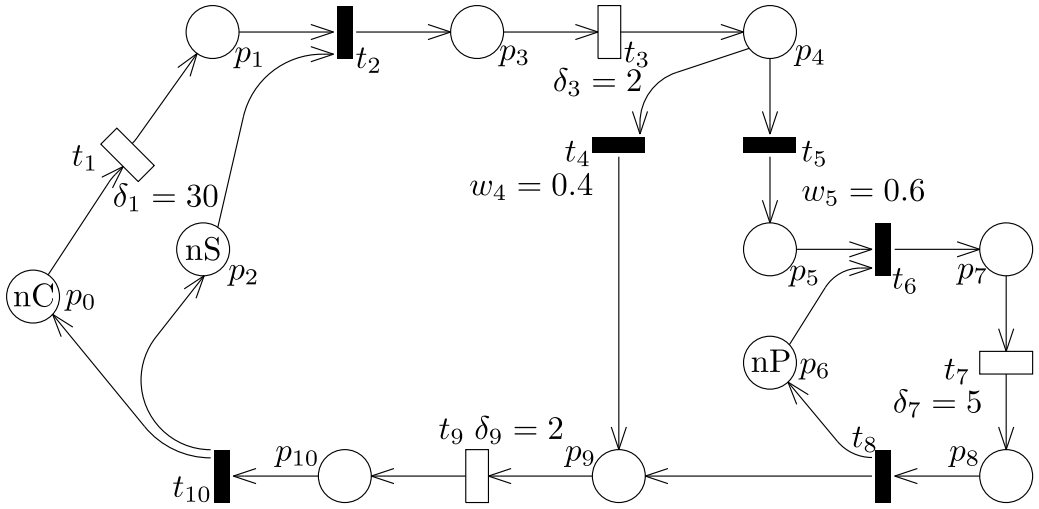


Figure 6.4: Example of a supermarket system.

Algorithm 2 represents the strategy used to compute throughput upper bounds. As input, the algorithm needs the SPPN system to be analysed, $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle$, and a degree of precision ($\varepsilon > 0$) to be achieved. As output, the upper throughput bound, Θ, and the places belonging to the bottleneck of the SPPN, $\mathbf{Q}$, are obtained. The degree of precision ε will be used for the stopping criterion of the iterative strategy.

In the first place, the initial upper throughput bound is calculated by LPP (6.6) (step 1). Then, the value of h is computed by using the LPP shown in (6.10) so that the feasibility of the LP is ensured. The iteration process (steps 4–9) is repeated until no significant improvement is achieved with respect to the last iteration or until the last obtained bottleneck contains all places in its support. In the worst case, only one place will be added in each iteration. Therefore, the algorithm complexity is polynomial due to the LPP.

In step 5, the places that share output transitions with some place contained in the support of $\mathbf{y}$ are calculated. Step 6 corresponds to the LPP (6.11). Finally, in step 8 the throughput of the subnet associated to the new bottleneck is considered as the new upper bound. The throughput is calculated by solving the Markov Chain [Murata, 1989] associated to the current bottleneck when it can be computed within a practical time, or by simulation otherwise.

Example. Recall the PN depicted in Figure 4.5 that represents a packet-routing algorithm. Such a PN models an algorithm inside a router where packets arrive and after checking source and destination of the packets, they are filtered following some defined rules.

Let the initial marking be $nP = 21$, $nT = 4$ and $nS = 2$. The vector of visit ra-

tios $\mathbf{v}$ normalised for transition t_1, is $\mathbf{v}^{t_1} = \{1.0, 1.0, 1.0, 0.4, 0.6, 0.6, 0.6, 0.6, 1.0, 1.0\}$. According to LPP (6.6) (step 1 of the Algorithm 2) the critical bottleneck is composed of $\|\mathbf{y}\| = \{p_0, p_1, p_3, p_4, p_5, p_6, p_8, p_9, p_{10}, p_{11}\}$, that is, the p-semiflow which corresponds to the packets' life-cycle. Such a result indicates that the system has, on average, enough resources to attend to the expected incoming packets. The upper throughput bound (normalised for transition T_0) of the critical bottleneck is $\Theta(T_0) = 0.567521$ (result of LPP (6.6)) and the value which guarantees the feasibility of the problem is $h = 0.037037$ (step 2). The places sharing output transitions with places in $\|\mathbf{y}\|$, i.e., connected to the critical bottleneck, are p_2 and p_7 (calculated in step 5). Each one corresponds to the resources of the system, the threads and filtering-threads, respectively. The result of the LPP in step 6 allows to regrow the current bottleneck, imposing that $\mathbf{y}'(p_2) + \mathbf{y}'(p_7) \geq h$ (that is, one of them, at least, must be contained on the support of $\mathbf{y}'$), and gives the new bottleneck which is composed of $\|\mathbf{y}'\| = \{p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_8, p_9, p_{10}, p_{11}\}$. The new throughput is $\Theta'(t_1) = 0.514220$ (step 8), which represents an improvement of 9.3919% with respect to the previous bottleneck. Note that the place added is that representing the number of threads (i.e., p_2).

Let us assume that $\varepsilon = 0.001$. As the relative difference between Θ and Θ' is 0.093919 (as commented previously), the iteration process carries on. At this point, the only place that is not connected to the critical bottleneck is p_7, which corresponds to the number of filtering-threads. By solving the LPP in step 6 the new bottleneck is obtained, which has all the places of the system in its support (i.e., $\|\mathbf{y}\| = P$), and the new throughput is $\Theta = 0.480642$. So, as the support of the new bottleneck contains all places of the net, the iteration process finishes. The new throughput Θ represents an improvement of 6.5299% with respect to the previous bottleneck, and a total improvement of 15.3085% with respect to the initial bottleneck.

The proposed iterative strategy is applied to a larger system in Chapter 8.

6.4.2 Numerical Problems in LPP (6.9)

The strict inequality $\sum_{p \in V} \mathbf{y}(p) > 0$ in (6.9) is used to compel the components of places which belong to the next slowest p-semiflow to be positive. Once the LPP (6.9) is solved, only the strictly positive components are selected. When the solver precision is not very high, zero components might not be distinguishable from positive components with low values. To avoid this, $\sum_{p \in V} \mathbf{y}(p) > 0$ is replaced by $\sum_{p \in V} \mathbf{y}(p) > h$, with a strictly positive h. Thus, we need to find a value $h > 0$ that retains the feasibility of constraints $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$, $\mathbf{y} \cdot \mathbf{m_0} = 1$. An alternative way to LPP (6.10) to compute a value h such that $\mathbf{y} \geq h \cdot \mathbf{1}$ and which fulfils both equations is following:

Recall that by the process PN structure, the number of p-semiflows is equal to $n + 1$, where $n = |P_R|$ is the number of resources in the process PN system. Note as well that the initial marking $\mathbf{m_0}$ of the system will be $\mathbf{m_0}(p) > 0,\ \forall p \in P_R \cup P_0$, $\mathbf{m_0}(p) = 0,\ \forall p \in P_S$. A p-semiflow $\mathbf{y}$ that covers all places can be computed by a linear combination of all

minimal p-semiflows. Remember that each resource has an associated minimal p-semiflow (see Definition 5).

Let us consider a system with n resources. Then, a linear combination of all minimal p-semiflows is $\mathbf{y} = \alpha_1 \cdot \mathbf{y}_1 + \alpha_2 \cdot \mathbf{y}_2 + \cdots + \alpha_{n+1} \cdot \mathbf{y}_{n+1},\ \alpha_i > 0, \forall i \in \{1 \ldots (n+1)\}$. As $\mathbf{y}$ is a linear combination of p-semiflows, then $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$ is fulfilled. However, factors α_i must be adjusted in order to properly fulfil equation $\mathbf{y} \cdot \mathbf{m_0} = 1$. An intuitive idea for doing this is the following: as $\mathbf{y}(p) \cdot \mathbf{m_0}(p) > 0 \Leftrightarrow p \in P_R \cup P_0$, then $\mathbf{y} \cdot \mathbf{m_0} = 1$ can be reformulated as $\alpha_1 \cdot \mathbf{y}_1(p_{r_1}) \cdot \mathbf{m_0}(p_{r_1}) + \alpha_2 \cdot \mathbf{y}_2(p_{r_2}) \cdot \mathbf{m_0}(p_{r_2}) + \cdots + \alpha_{n+1} \cdot \mathbf{y}_{n+1}(p_{r_{n+1}}) \cdot \mathbf{m_0}(p_{r_{n+1}}) = 1$, where p_{r_i} represents the place associated to resource r_i, $\forall i \in \{1 \ldots n\}$, and $p_{r_{n+1}}$ is the process-idle place.

By the process PN structure, all positive values of $\mathbf{y}_i$ will be equal to one. Therefore, the values α_i that fulfil the equation $\mathbf{y} \cdot \mathbf{m_0} = 1$ can be easily calculated as:

$$\alpha_i = \frac{1}{\mathbf{m_0}(p_{r_i}) \cdot (n+1)},\ \forall i \in \{1 \ldots (n+1)\}$$

Hence, a possible value h that fulfils $\mathbf{y}(p) \geq h,\ \forall p \in P$ is, in this case, $h = min(\alpha_i),\ \forall i \in \{1 \ldots (n+1)\}$. Such a value relates the number of resources in the system and the number of resource instances. Thus, the value of h for most systems of interest in practice is much higher than the numerical tolerance of the LPP solver (in this paper, the numerical tolerance of the LPP solver has been set to 10^{-5}).

As the objective function in LPP (6.10) is maximised, the value h obtained from that LPP will be at least equal to $min(\alpha_i),\ \forall i \in \{1 \ldots (n+1)\}$, that is: $h \geq min(\alpha_i),\ \forall i \in \{1 \ldots (n+1)\}$.

6.5 Concluding Remarks

Current system requirements often impose tight constraints on time properties such as system performance. In order to check such requirements, it is necessary to have methods that accurately evaluate the system performance. Unfortunately, in most cases of interest it is not possible to compute the exact performance of a system in a reasonable time due to the state explosion problem inherent to large discrete systems. The state explosion problem poses difficulties not only for computing exactly the performance of an existing system, but also for correctly designing new systems. Thus, those methods must, not only be accurate, but also efficient in order to be applicable to the increasingly complex systems existing in practice.

In this chapter, we have proposed two methods that can be applied to Stochastic Marked Graphs and Process Petri nets, respectively. Both methods are based on an iterative algorithm that takes an initial throughput bound and refines it in each iteration. The initial bound is given by the most constraining (or bottleneck) cycle, i.e., the one with minimum token to delay ratio. The refinements are achieved by adding to the bottleneck cycle places

and transitions with low token to delay ratio. The bound is refined until no significant improvement is obtained.

The outputs of both methods are an accurate estimate for the steady state throughput, and as a by-product, a subnet representing the bottleneck of the system. The first approach has been applied to a set of Stochastic Marked Graphs of different sizes, where the results show that few iterations suffice to obtain accurate bounds and that, in general, such bounds are due to relatively small subnet bottlenecks of the system. The second approach has been applied to a running example.

Given that both techniques make intensive use of linear programming techniques and the number of required iterations is usually low, their complexity and computational time are also low. Such system bottlenecks represent the targets on which potential methods for performance optimisation might focus.

Chapter 7

Compensation of Throughput Degradation in FT Systems

This chapter introduces the main contributions of this dissertation related to the compensation of throughput degradation caused by any activation of faults in a degradable system. Recall that degradable systems usually incorporate Fault-Tolerant (FT) techniques to mitigate the consequences of fault activations, then conforming a FT system. As it is claimed in Section 1.1, many of these FT systems are complex systems using shared resources, and can be naturally modelled as Discrete Event Systems (DES), more precisely as Resource Allocation Systems (RAS) [Colom, 2003]. Recall that we focus on FT systems using shared resources modelled as a special class of Petri nets (PNs) called Process Petri nets (PPNs).

The outcome of this chapter have been mainly published in [Rodríguez et al., 2013a] and [Rodríguez et al., 2013b].

7.1 Motivation

The throughput of a FT system can be degraded (that is, it becomes lower) by the activation of faults, or the presence of errors or failures into the system. Thus, it is important to know the expected failure rate of the overall system when designing it, because some analysis might be carried out aiming at minimising the throughput degradation caused by faults.

Compensation of a throughput degradation in a FT system can be performed by two main actions: either the number of items of resources is increased, or the timing of FT techniques is decreased. However, neither the number of resources (for example, the number of servers in a web system) can always be increased as desired, nor the timing of FT techniques can be performed in zero time (ideal time). In the real world, each project of a new system manages a budget, and this budget limits the number of resources that can be acquired and the time of FT techniques that can be improved.

The major findings of this chapter are threefold. Firstly, we propose an iterative heuris-

tics to gauge in the best possible way the number of resources needed so that the overall system throughput is maximised for Stochastic Process Petri nets (SPPNs). The other results target to FT systems modelled as SPPN where the compositional PN models for FT (introduced in Section 4.2) are added: we propose an iterative algorithm to compute the number of resources that mitigate the impact of activation of faults in a FT system; and lastly, we propose an Integer Linear Programming Problem (ILPP) that minimises the cost of compensation needed for maintaining a given throughput in a FT system.

7.2 Maximising Throughput through Resource Optimisation

In this section we propose a heuristic strategy to gauge the number of resources a system, modelled as a Process Petri net, should allocate. Our approach for resource optimisation is similar to Goldratt's principle [Goldratt and Cox, 1986]: once the system's bottleneck is identified, the associated resource is increased.

7.2.1 Calculating the Next Constraining Resource

Let us recall LPP (6.6) to calculate an upper throughput bound of a SPPN. The most constraining p-semiflow, $\mathbf{y}$, will have just one marked place in its support due to the net structure (see Definition 5). Assume that the marked place corresponds to a resource place (not the process-idle place), then given that $\mathbf{y}$ constrains the throughput of the whole system, the addition of more instances to the resource place will result in an increase in the system throughput. At a certain moment, the resource becomes saturated and adding more instances does not improve the throughput. This occurs because the constraining p-semiflow has changed. Note that the upper throughput bound will linearly increase with the number of tokens of the resource place because it is the only place in $\|\mathbf{y}\|$ having tokens and the equation $\mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D}$ is linear.

Hence, the resource r_1 contained in the support of the most constraining p-semiflow $\mathbf{y}_{r_1}$, can be increased until $\mathbf{y}_{r_1}$ is no longer the bottleneck p-semiflow. Let ${\mathbf{m_0}}^{\Delta}$ be the initial marking vector $\mathbf{m_0}$ with an increase α_1 of the resource r_1, i.e.,

$$ {\mathbf{m_0}}^{\Delta} = \begin{cases} \mathbf{m_0}(p), & p \neq r_1 \\ \mathbf{m_0}(p) + \alpha_1, & p = r_1 \end{cases} \tag{7.1} $$

The p-semiflow $\mathbf{y}_{r_1}$ is not the only constraining p-semiflow if the following equation holds:

$$ \frac{\mathbf{y}_{r_1} \cdot \mathbf{Pre} \cdot \mathbf{D}}{\mathbf{y}_{r_1} \cdot {\mathbf{m_0}}^{\Delta}} \leq \frac{\mathbf{y}_{r_2} \cdot \mathbf{Pre} \cdot \mathbf{D}}{\mathbf{y}_{r_2} \cdot {\mathbf{m_0}}^{\Delta}} \tag{7.2} $$

where $\mathbf{y}_{r_2} \neq \mathbf{y}_{r_1}$ is a p-semiflow. Note that the p-semiflow $\mathbf{y}_{r_2}$ will contain in its support the next most constraining resource r_2, and, by definition, $r_1 \neq r_2$.

The number α_1 of instances of the resource place r_1, contained in the most constraining p-semiflow $\mathbf{y}_{r_1}$, which need to be added to obtain the next constraining resource r_2, contained in the next most constraining p-semiflow $\mathbf{y}_{r_2}$, can be easily computed by solving the following LPP:

$$
\begin{array}{lrll}
\textit{minimum} & \alpha_1 & & \\
\textit{subject to} & \mathbf{y}_{r_2} \cdot \mathbf{Pre} \cdot \mathbf{D} & = \mathbf{y}_{r_1} \cdot \mathbf{Pre} \cdot \mathbf{D} & \\
& \mathbf{y}_{r_2} \cdot \mathbf{C} & = 0 & \\
& \mathbf{y}_{r_2}(r_1) & = 0 & \\
& \mathbf{y}_{r_2} \cdot \mathbf{m_0}^{\Delta} & = \mathbf{y}_{r_1} \cdot \mathbf{m_0}^{\Delta} & \\
& \mathbf{m_0}^{\Delta} & = \begin{cases} \mathbf{m_0}(p), & p \neq r_1 \\ \mathbf{m_0}(p) + \alpha_1, & p = r_1 \end{cases} & \\
& \alpha_1, \mathbf{y}_{r_2} & \geq 0 &
\end{array}
\tag{7.3}
$$

where $\mathbf{y}_{r_1}$ is the p-semiflow which contains r_1 in its support, $\mathbf{y}_{r_2}$ is the p-semiflow which contains r_2 in its support and $\mathbf{m_0}^{\Delta}$ represents the initial marking vector $\mathbf{m_0}$ with the increase α_1 in r_1.

Constraints $\mathbf{y}_{r_2} \cdot \mathbf{Pre} \cdot \mathbf{D} = \mathbf{y}_{r_1} \cdot \mathbf{Pre} \cdot \mathbf{D}$ and $\mathbf{y}_{r_2} \cdot \mathbf{m_0}^{\Delta} = \mathbf{y}_{r_1} \cdot \mathbf{m_0}^{\Delta}$ are both parts (dividend and divisor, respectively) of equation (7.2) equalled. Constraint $\mathbf{y}_{r_2} \cdot \mathbf{C} = 0$ ensures that $\mathbf{y}_{r_2}$ is a left annuler of the incidence matrix, hence a p-semiflow of the net. Finally, constraint $\mathbf{y}_{r_2}(r_1) = 0$ is added to avoid a product of two optimisation variables (the variable α_1 and the variable $\mathbf{y}_{r_2}(r_1)$ in equation $\mathbf{y}_{r_2} \cdot \mathbf{m_0}^{\Delta} = \mathbf{y}_{r_1} \cdot \mathbf{m_0}^{\Delta}$). Moreover, the variable $\alpha_1 \in \mathbb{R}_{\geq 0}$ therefore, the linearity of the optimisation problem is ensured.

Both α_1 and the next constraining p-semiflow $\mathbf{y}_{r_2}$ are obtained when the LPP is solved. Note that the increase of a resource r_1 does not affect the ratio $\frac{\mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{D}}{\mathbf{y} \cdot \mathbf{m_0}}$ of any other minimal p-semiflow $\mathbf{y}$ which contains another resource in its support (see definition of the process Petri nets class in Section 2.1). Notice that, as in Section 6.4, a LPP is used to solve a problem that deals with integer values as the number of resources. This relaxation of the real domain remarkably decreases the complexity of the approach (the complexity of solving a LPP is polynomial), at the cost of some loss of precision in the results. Once both α_1 and the next constraining p-semiflow $\mathbf{y}_{r_2}$ are obtained, LPP (7.3) can easily be extended to calculate the next constraining resource and the number of tokens, i.e., instances, to be increased of both places:

$$
\begin{array}{lrll}
\textit{minimum} & & \alpha_1 + \alpha_2 & \\
\textit{subject to} & \mathbf{y}' \cdot \mathbf{Pre} \cdot \mathbf{D} & = \mathbf{y}_{r_1} \cdot \mathbf{Pre} \cdot \mathbf{D} & \\
& \mathbf{y}' \cdot \mathbf{C} & = 0 & \\
& \mathbf{y}'(r_1) = 0, & \mathbf{y}'(r_2) = 0 & \\
& \mathbf{y}' \cdot \mathbf{m_0}^{\Delta} & = \mathbf{y}_{r_1} \cdot \mathbf{m_0}^{\Delta} & \\
& \mathbf{y}' \cdot \mathbf{m_0}^{\Delta} & = \mathbf{y}_{r_2} \cdot \mathbf{m_0}^{\Delta} & \\
& \mathbf{m_0}^{\Delta} & = \begin{cases} \mathbf{m_0}(p), & p \notin \{r_1, r_2\} \\ \mathbf{m_0}(p) + \alpha_1, & p = r_1 \\ \mathbf{m_0}(p) + \alpha_2, & p = r_2 \end{cases} & \\
& \alpha_1, \alpha_2, \mathbf{y}' & \geq 0 &
\end{array}
\tag{7.4}
$$

where $\mathbf{m_0}^{\Delta}$ represents the initial marking vector $\mathbf{m_0}$ with the increase α_1 of place r_1 and the increase α_2 of place r_2, and $\mathbf{y}_{r_1}$ ($\mathbf{y}_{r_2}$) is the p-semiflow which contains r_1 (r_2) in its support.

As in LPP (7.3), constraint $\mathbf{y}' \cdot \mathbf{C} = 0$ ensures that $\mathbf{y}'$ is a left annuler of the incidence matrix, and hence $\mathbf{y}'$ is a p-semiflow of the net. Besides, constraints $\mathbf{y}'(r_1) = 0$ and $\mathbf{y}'(r_2) = 0$ ensure linearity of the optimisation problem. Constraints $\mathbf{y}' \cdot \mathbf{m_0}^{\Delta} = \mathbf{y}_{r_1} \cdot \mathbf{m_0}^{\Delta}$, $\mathbf{y}' \cdot \mathbf{m_0}^{\Delta} = \mathbf{y}_{r_2} \cdot \mathbf{m_0}^{\Delta}$ are the key of this LPP because both values of α_1 and α_2 can be obtained from these equations.

Note that $\mathbf{y}' \cdot \mathbf{Pre} \cdot \mathbf{D} = \mathbf{y}_{r_2} \cdot \mathbf{Pre} \cdot \mathbf{D}$ is not a constraint in LPP (7.4). This is a consequence of the result of LPP (7.3): from the latter LPP where r_1 is calculated, it is imposed that $\mathbf{y}_{r_2} \cdot \mathbf{Pre} \cdot \mathbf{D} = \mathbf{y}_{r_1} \cdot \mathbf{Pre} \cdot \mathbf{D}$. The addition of this constraint does not add new information to LPP (7.4).

LPP (7.4) can be generalised for more resources, as is shown in step 5 of the Algorithm 3.

7.2.2 An Iterative Strategy for Resource Optimisation

This subsection presents an iterative heuristics that aims at maximising the throughput by increasing the number of resources appropriately. The main idea of the strategy is to estimate the *inflexion points* where the constraining p-semiflows change, and hence to estimate the increase in resources needed. More precisely, each unit of a resource has an associated cost and the strategy establishes how to spend a given budget such that the throughput is maximised. The strategy ends either when there is no budget to spend, all resources have been dimensioned, or the last computed p-semiflow indicates an increase in the process-idle place.

Algorithm 3 shows the resource optimisation heuristics. For the input, the algorithm needs the SPPN system to be analysed, $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle$, the set of resources and the process-idle place of the system, R and p_0 (respectively), the assigned budget to be spent, *budget*, and

Input: $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle, R, p_0, budget, c$
Output: n

Calculate initial bottleneck $\mathbf{y}_1$ by solving LPP (6.6)
$k = 0$; $cost = 0$; $n^{'} = \mathbf{0}$
while $cost < budget$ **and** $k \neq |R|$ **and** $\|\mathbf{y_{k+1}}\| \cap \{p_0\} = \emptyset$ **do**
  $k = k + 1$; $cost' = cost$; $n = n^{'}$; $\mathbf{A} = \{p | p \in P, p \in \|\mathbf{y}_j \cap R\|\},\ \forall j \in \{1 \dots k\}$

$$
\begin{aligned}
& minimum \sum_{j=1}^{k} \alpha_j \\
& subject\ to\ \mathbf{y_{k+1}} \cdot \mathbf{Pre} \cdot \mathbf{D} = \mathbf{y_1} \cdot \mathbf{Pre} \cdot \mathbf{D} \\
& \mathbf{y_{k+1}} \cdot \mathbf{C} = \mathbf{0} \\
& \mathbf{y_{k+1}} \cdot \mathbf{m_0^{\Delta}} = \mathbf{y_j} \cdot \mathbf{m_0^{\Delta}},\ \forall j \in \{1 \dots k\} \\
& \mathbf{m_0}^{\Delta} = \begin{cases} \mathbf{m_0}(p) + \alpha_j, & p \in \mathbf{A} \\ \mathbf{m_0}(p), & otherwise \end{cases} \\
& \mathbf{y_{k+1}}(p) = 0,\ p \in \mathbf{A} \\
& \mathbf{y_{k+1}},\ \alpha_j \geq 0,\ \forall j \in \{1 \dots k\}
\end{aligned}
$$

  $cost = 0$; $n^{'} = \mathbf{0}$
  for $\alpha_j,\ \forall j \in \{1 \dots k\}$ **do**
    $r_j = \|\mathbf{y}_j\| \cap R$; $n_j^{'} = \lceil \alpha_j \rceil$
    $cost = cost + \lceil \alpha_j \rceil \cdot c_i$
  end
end
if $k \leq |R|$ **and** $cost \leq budget$ **then**
  $n = n^{'}$
end
if $k < |R|$ **and** $cost \leq budget$ **and** $\|\mathbf{y_{k+1}}\| \cap \{p_0\} = \emptyset$ **then**
  $assignRestOfBudget(budget - cost, \langle \mathcal{S}, \mathbf{s} \rangle, R, c, n)$
end

Algorithm 3: The resource optimisation heuristics.

the vector of cost c, which assigns a cost c_i to each resource r_i contained in R. The output is the number of items n_i needed to increase each resource r_i.

Firstly, an upper throughput bound $\mathbf{y}_1$ of $\langle \mathcal{S}, \mathbf{s}, \mathbf{r} \rangle$ is calculated according to LPP (6.6). After that, the iteration process (steps 3–10) is repeated either until the last assignment of resources has spent the available budget, or until all resources have been dimensioned, or until the last computed resource to be increased matches with the process-idle place.

Step 5 calculates, in each iteration, the number of items of a resource which need to be increased to obtain the next restrictive resource. It should be noted that the LPP in step 5 is a generalisation of LPP (7.3). After that, the cost of increasing such a number of instances of the resources is computed. Note that the ceiling integer of the value α_j is taken as the result. There are two reasons for this: firstly, we assume that the number of instances of the resources must be a natural number; and secondly, when the resource is not saturated it will still be the restrictive resource.

Finally, step 12 checks whether all the resources have been assigned and that the cost of new resources does not exceed the given budget. When these conditions are fulfilled, the last resource assignment is taken as the valid one. Step 15 checks whether there is a resource that has not been assigned, the last resource assignment does not exceed the given budget and the last computed p-semiflow does not contain the process-idle place. When these conditions are fulfilled, the remaining budget may be spent on increasing the system throughput. A procedure is invoked ($assignRestOfBudget$, step 16) for spending the rest of the assigned budget to increase the resources as much as possible. Note that the assignment of the remaining budget is an NP-problem, similar to the Bounded Knapsack Problem (BKP) [Kellerer et al., 2004]. To solve it, several heuristics can be used. For instance, a "round-trip" algorithm which tries to increase all the resources per round until it cannot longer increase them.

Let us illustrate the use of this strategy through the packet-routing algorithm example, depicted in Figure 6.4. Suppose an initial marking of $nP = 30$, $nT = 2$ and $nS = 2$, and an initial budget of $\$30,000$ dollars. The deployment of each new thread costs $\$5,000$ dollars, while a new filtering-thread deployment has a price of \$700 dollars. The initial bottleneck is $\|\mathbf{y_1}\| \cap R = \{p_2\}$, that is, the subnet associated to the threads. Therefore, this result gives us the following information: to attend to 30 packets whose think time follows an exponential distribution of a mean of 30 minutes, more threads are needed. The LPP at step 5 gives, in the first iteration, the increase in new threads needed, $\alpha_1 = 2.666$, and the new constraining p-semiflow, which corresponds to the use of filtering-threads. So, at least three new threads ($\lceil \alpha_1 \rceil$) are needed to attend to the incoming packets.

As the cost of deployment of a new thread is $\$5,000$ dollars and the initial budget is $\$30,000$ dollars, the new deployments can be done and there is still money which remains to be spent, so a new iteration can take place. The LPP at step 5 gives, in the second iteration, the values of $\alpha_1 = 3.6752$ and $\alpha_2 = 0.4322$. Hence, to attend to the packets, four new threads and one more filtering-thread are needed. As the cost of these are $\$20,700$ dollars in total, the increase in resources can be carried out. Now, the unassigned budget

is $\$9,300$ and we can continue increasing both resources in parallel. Indeed, the relation between both resources is known thanks to the equalities of the ratios.

In this case, even though part of the budget remains to be spent, the new constraining p-semiflow contains the process-idle place, that is, the place representing packets. Thus, the resources of the system (threads and filtering-threads) have been optimally calculated to attend to 30 packets whose think time follows an exponential distribution of a mean of 30 minutes. In this way, the algorithm has computed that to attend to the customers, at least four more threads and one filtering-threads are needed.

Note that it may happen that the LPP at step 5 returns the p-semiflow containing in its support the process-idle place in the first iteration. This would indicate that the system has enough resources to attend to such a number of customers with such a think time. Therefore, the strategy is also able to compute when a system with an initial configuration is able to support the estimated workload, or otherwise, to compute the number of instances of resources needed to be able to support such a workload.

7.3 Minimising Cost of Compensating Throughput Degradation

This section introduces an iterative strategy that computes the number of resources needed to maintain a given upper throughput bound in a degradable system where our proposed FT models are added (see Section 4.2).

Such a strategy is presented in Algorithm 4. As input, it needs the description of the PN model with the FT techniques added to it with the initial marking and the vector of service times of transitions, $\langle \mathcal{N}, \mathbf{m_0}, \delta \rangle$; the upper throughput bound Θ before adding the FT techniques; and the set $\mathbf{Y}^{FT}$ of minimal p-semiflows that are modified after adding the FT techniques. As output, it returns the initial marking $\mathbf{m'_0}$ such that the upper throughput bound Θ' of the FT system is greater than or equal than Θ.

Input: $\langle \mathcal{N}, \mathbf{m_0}, \delta \rangle, \Theta, \mathbf{Y}^{FT}$
Output: $\mathbf{m'_0}$
1 $\mathbf{m'_0} = \mathbf{m_0}$
2 **for** $\mathbf{y}_i \in \mathbf{Y}^{FT}$ **do**
3 $\quad \mathbf{m'_0}(r_i) = maximum(\mathbf{m_0}(r_i), \lceil (\mathbf{y}_i \cdot \mathbf{Pre} \cdot \mathbf{D}) \cdot \Theta \rceil)$
4 **end**

Algorithm 4: An iterative algorithm to compute initial marking needed to maintain a certain upper throughput bound with a probability of error.

Algorithm 4 works as follows. It iterates in the content of the set $\mathbf{Y}^{FT}$ of minimal p-semiflows that have been modified when adding a proposed FT model. For each minimal p-semiflow $\mathbf{y}_i \in \mathbf{Y}^{FT}$, the value of the initial marking for associated resource r_i is com-

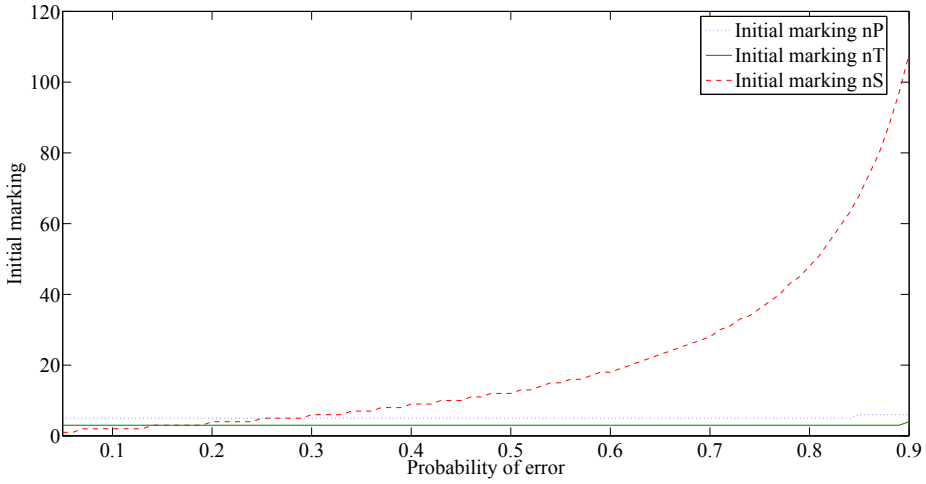


Figure 7.1: Results of initial marking with respect to probability of error.

puted as the maximum of the previous initial marking of the resource (i.e., $\mathbf{m_0}(r_i)$) or the $\lceil(\mathbf{y}_i \cdot \mathbf{Pre} \cdot \mathbf{D}) \cdot \Theta\rceil$. The latter equation comes from solving $\Theta = \dfrac{\mathbf{m_0}(r_i)}{\mathbf{y}_i \cdot \mathbf{Pre} \cdot \mathbf{D}}$. The ceiling is needed because $\mathbf{m'_0}(r_i) \in \mathbb{N}$.

Let us apply the Algorithm 4 in the Petri net example depicted in Figure 4.6 (the packet-routing algorithm). The previous upper throughput bound is $\Theta = 0.470588$, and the set of minimal p-semiflows that are modified after adding isolation FT is $\mathbf{Y}^{FT} = \{\mathbf{y}'_1, \mathbf{y}'_2, \mathbf{y}'_3\}$. For a given initial marking $\mathbf{m_0}(p_0) = 10, \mathbf{m_0}(p_2) = 2, \mathbf{m_0}(p_7) = 2$, Algorithm 4 returns as solution: $\mathbf{m'_0}(p_0) = \mathbf{m_0}(p_0) = 10, \mathbf{m'_0}(p_2) = 3, \mathbf{m'_0}(p_7) = 4$. That is, it is needed another thread and two more filtering-threads to compensate a 20% of errors (and a 5% of them deriving in solid faults) using reconfiguration as FT technique.

We have plotted in Figure 7.1 the initial marking needed to support the given throughput of $\Theta = 0.470588$ varying the probability of error $r_e, r_e \in [0 \ldots 1]$, taking steps of 0.01. The dotted line is the initial number of tokens of p_0 (packets, nP), the solid line corresponds to the initial number of tokens of p_2 (threads, nT) and the dashed line is the initial number of tokens of p_7 (filtering-threads, nS). The results show that the number of packets and threads remain more or less equal, i.e., there is no need to increment too much units to be able to maintain the given throughput, even with high probability of errors. However, the number of filtering-threads needed increases rapidly with respect to the probability of error.

7.3.1 An ILPP for Minimising the Cost of Compensating

In this section, we present an Integer-Linear Programming Problem (ILPP) that minimises the cost of compensating throughput degradation caused by the presence of errors.

We are able to compute the initial marking needed to maintain a given throughput with the previous Algorithm 4. However, the increment of items of resources can have a cost in real systems and we may not be able to increment as much as it is desired. Recall that equation $\frac{\mathbf{m_0}(r_i)}{\mathbf{y}_i \cdot \mathbf{Pre} \cdot \mathbf{D}}$ relates not only the number of items of resources ($\mathbf{m_0}(r_i)$) but also activity timings and error (and solid faults) probabilities ($\mathbf{D}$). If we consider a given error probability r_e and solid faults probability r_s, a compensation may be done in two ways: either the number of resources in the system can be incremented, or the timing of FT activities (detection, compensation and recovery phases) can be decremented. Both ways can have some cost associated.

Let us assume that FT phases are abstracted in single timed transition, i.e., a FT technique j adds to the system three timed transition: T^j_{detect} (detection phase), T^j_c (compensation phase) and T^j_{rec}/T^j_{MTTR} (recovery/maintenance phase). Let c^r_i the cost of an increment of one unit of the resource r_i, and c^d_j the cost of a decrement of one unit of time of detection phase of FT technique j, while $c^c_j(c^{rm}_j)$ is the cost of a decrement of one unit of time of compensation(recovery/maintenance) phase.

We can build an Integer-Linear Programming Problem (ILPP) to compute the minimum cost that guarantees a compensation of the throughput system after adding a number m of FT techniques as follows:

$$
\begin{aligned}
minimum \quad & \left(\sum_{i=1}^{n} c^r_i \cdot \alpha_i + \sum_{j=1}^{m} \left(c^d_j \cdot \beta^d_j + c^c_j \cdot \beta^c_j + c^{rm}_j \cdot \beta^{rm}_j \right) \right) \; subject\ to \\
\mathbf{m_0}(r_i) + \alpha_i &\geq \Theta \cdot \mathbf{y}_i \cdot \mathbf{Pre} \cdot \mathbf{D'} \\
\delta'(T^j_{detect}) &= \delta(T^j_{detect}) - \beta^d_j \\
\delta'(T^j_c) &= \delta(T^j_c) - \beta^c_j \\
\delta'(T^j_{rec}) &= \delta(T^j_{rec}) - \beta^{rm}_j \\
\delta'(t) &\geq \delta_{min}(t), \forall t \in T \\
\alpha_i, \beta^d_j, \beta^c_j, \beta^{rm}_j &\geq 0, \alpha_i \in \mathbb{N}, \forall i \in [1 \ldots n], \forall j \in [1 \ldots m]
\end{aligned}
\tag{7.5}
$$

where n p-semiflows have been modified by the addition of m FT techniques to the original system; $\mathbf{D'}(t) = \delta'(t) \cdot \mathbf{v}(t), \forall t \in T$; and $\delta_{min}(t)$ is a lower bound for the service time of transition t (that is, we impose a minimum service time for transitions). The new number of resources and firing of transitions will be given by the values of $\alpha_i, \beta^d_j, \beta^c_j, \beta^{rm}_j$, respectively.

This ILPP is applied to the case study shown in Chapter 8.

7.4 Concluding Remarks

Software systems are usually subject to faults that may lead to the existence of error and failures. Normally, Fault-Tolerant (FT) techniques are incorporated to these systems (then called FT systems) to mitigate the impact of activations of faults. FT systems can be naturally modelled as Discrete Event Systems (DES) where sharing resources are used. Usually, the number of resources is the key for the system to obtain a good throughput (defined as jobs completed per unit of time) for a large number of users/clients. However, the number of resources (for example, the number of servers) cannot always be increased as desired. In the real world, each project of a new system manages a budget, and this budget limits the number of resources that can be acquired.

In this chapter, we firstly provide a strategy whose goal is, given an initial budget and a cost of each resource, to gauge the number of instances of each resource so that the system performance is maximised and the budget is not exceeded. This has been achieved by exploiting the linear dependence of the performance bounds with respect to the number of resources, and can be applied to any FT system modelled as Process Petri nets. Secondly, we have presented an iterative algorithm that computes the initial marking needed to maintain a given upper throughput bound in a system model within our proposed FT models. Thirdly, we present an Integer-Linear Programming Problem (ILPP) that minimises the cost of compensating throughput degradation caused by the presence of faults and errors). The use of linear programming techniques guarantees its efficiency and scalability to large models.

We have developed a tool, `PeabraiN` [Rodríguez et al., 2012a], which implements the first of the strategies here presented to make their use easier for practitioners. It enables both performance estimation and resource optimisation to be computed in systems modelled with Petri nets. The `PeabraiN` tool is explained in more detail in Chapter 11. We aim at extending `PeabraiN` functionality with the second proposed strategy.

Part III

Applications

Chapter 8

Case Study: a Secure Database System

This chapter introduces a case study where the approaches presented in Chapters 6 and 7 are tested. We consider the design of a Secure Database System (SDBS) deployed as a Web Service which stores sensible information. Besides, there exist users which are eventually accessing to this information. A real application of this kind of system is, for instance, a web server keeping customer's data of an insurance company or a bank web server keeping customer's balance accounts. This case study has been used in [Rodríguez et al., 2013a] and [Rodríguez et al., 2013b] for testing purposes.

8.1 System Description

Using standard UML [OMG, 2005] notation, Figure 8.1 shows the UML Deployment Diagram (UML-DD) of the SDBS, which includes the hardware resources (depicted as cubes) and their network links (arrows between cubes or proper cubes in the case of intranets). Software modules stereotyped as `artifact` (and depicted as squares) are deployed into hardware resources. We have also used the standard MARTE [OMG, 2009] profile to introduce the system performance properties (e.g., workload, throughput or activities duration): input parameters (pale grey notes in Figure 8.1 and 8.2) and the metric to be computed (grey note in Figure 8.2). The architecture of the system is as follows: there exist a policy host, a security host, a provider host, an application host and a database (DB) host. Moreover, the latter is isolated and reachable only through a secure intranet connected to the application host. Note that each of these hosts deploys a concrete service or software module.

The workload is defined by the number of requests from users concurrently accessing the SDBS, which is parametrised by the variable *$nRequests*, an input parameter for the analysis. The number of hosts (security host, policy host, etc.) has been indicated using

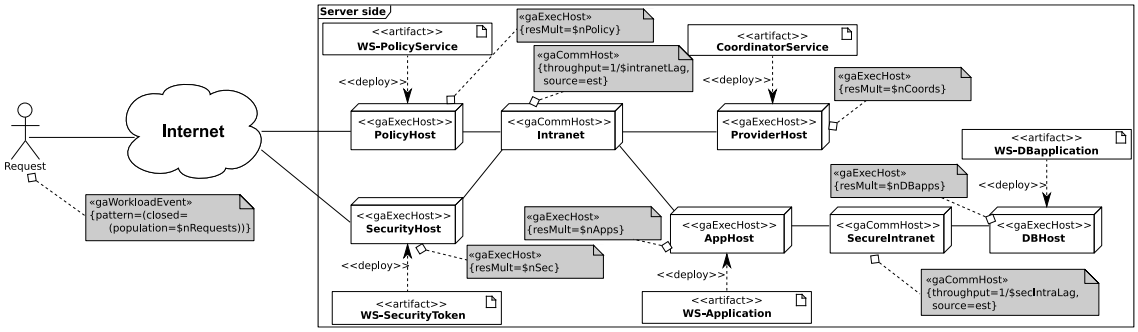


Figure 8.1: SDBS Deployment.

variables (*$nSec*, *$nPolicy*, etc.) by the tag `restMult`, also in the notes in Figure 8.1. Finally, the throughput of the intranets is considered through variables *$intranetLag* and *$secIntraLag*. As output parameter for the analysis, the top-most note in Figure 8.2 defines the variable *$rTime* as the response time of the SDBS system. Values for all these input variables are set in Table 8.1.

The SDBS works as follows: a user interacts with an application outside the system, which collects its personal data and the type of operation required (let us assume it will be an update of personal user data) by the user. This information is summarised on a request. Once generated the request, it needs a security token to be identified before accessing the system. Once the security token is retrieved, following the UML Sequence Diagram (UML-SD) diagram depicted in Figure 8.2, the policy host is requested for accessing, which checks the request, and if the permission is granted then it will invoke the service. The web service coordinator will communicate with the application service (located in the application host), which has access to the database application. Then, the DB application definitively updates the user request into the DB. Finally, the application service informs the coordinator service that the updating process has finished and sends to it the obtained result.

The acquire (release) of a resource has been indicated through the `gaAcqStep` (`gaRelStep`) stereotype (see notes in Figure 8.2), also denoting the number of resources acquired (released). To avoid cluttering we only show the first acquire (release) of resource *WS-SecurityToken*. The rest of acquires and releases of resources will be annotated in the same way. Activities have been annotated with `gaStep` stereotype to specify how long takes, on average, each activity using the tagged-value `hostDemand`. As in the previous case, for illustrative purposes we have just annotated two activities duration. Table 8.1 shows the duration of each activity for the experiments and the number of instances of each host in the system.

Description as a Petri net. Figure 8.3 shows the Petri net (PN) obtained from the transformation of the UML-SD shown in Figure 8.2. The transformation from UML to PN is documented in [Merseguer et al., 2002, López-Grao et al., 2004,

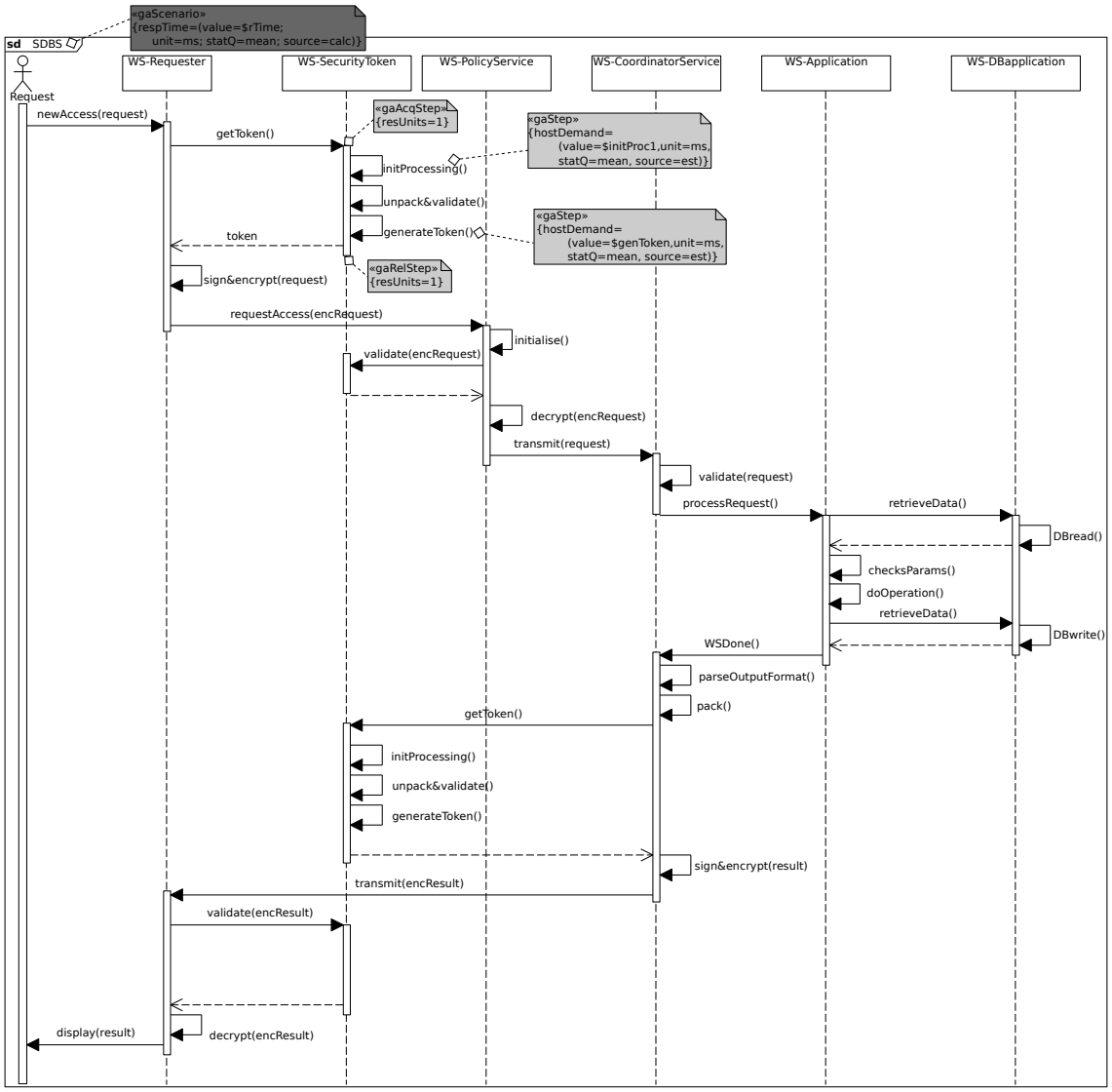


Figure 8.2: SDBS Update Customer's Data scenario.

Transition	Method	Value(s)
T_0	`newAccess()`	$0.2ms$
T_2, T_8, T_{10}, T_{49}	*$delayNet*	$2.5ms$
$T_{13}, T_{16}, T_{19}, T_{23}$, T_{36}, T_{41}, T_{46}	*$intranetLag*	$0.2ms$
$T_{26}, T_{29}, T_{32}, T_{34}$	*$secIntraLag*	$0.5ms$
T_4, T_{43}	`initProcessing()`	$1ms$
T_5, T_{44}	`unpack&validate()`	$0.1ms$
T_6, T_{45}	`generateToken()`	$0.5ms$
T_9, T_{48}	`sign&encrypt()`	$0.8ms$
T_{12}	`initialise()`	$0.3ms$
T_{15}, T_{22}, T_{52}	`validate()`	$0.3ms$
T_{18}, T_{54}	`decrypt()`	$1ms$
T_{28}, T_{33}	`DBread()`	$0.2ms$
T_{30}	`checkParams()`	$0.6ms$
T_{31}	`doOperation()`	$0.2ms$
T_{39}	`parseOutputFormat()`	$0.3ms$
T_{40}	`pack()`	$0.1ms$
T_{55}	`display()`	$1.5ms$

(a) Activity times

Place	Meaning	Value(s)
p_0	No. users	$15, 20, 21, 22, 23 \ldots 30$
p_2	No. request capacity	$\geq \mathbf{m_0}(p_0)$
p_5	No. security hosts	5
p_{13}	No. policy hosts	10
p_{24}	No. coordinator hosts	10
p_{29}	No. application hosts	5
p_{32}	No. DB hosts	2

(b) Initial number (no.) of resources

Table 8.1: Experimental parameters.

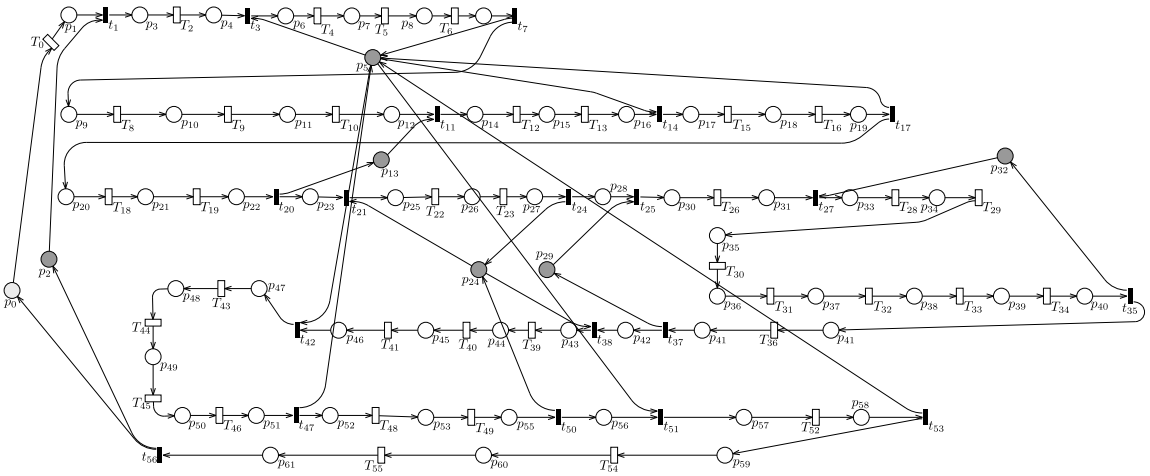


Figure 8.3: Petri net of the SDBS. Resource places are depicted in dark grey, whilst process-idle place in light grey.

Distefano et al., 2011], and can be carried out by several tools, such as ArgoPN, ArgoPerformance [Distefano et al., 2011] or ArgoSPE. In this case, ArgoSPE tool[1] has been chosen to carry out this transformation because the ArgoSPE output net format is compatible with GreatSPN tool [Baarir et al., 2009] input net format (used later for analysis in the experiments). Note that as software engineers usually work with UMLs diagrams, ArgoSPE is useful in this context for obtaining the PN models we need to work with.

Each resource annotated in Figure 8.1 is represented by a place in the PN: the resource places (depicted in dark grey) are p_7 (security service), p_{18} (policy service), p_{26} (coordinator service), p_{28} (application service) and p_{31} (database service), while the process-idle place (user's requests, depicted in light grey) is represented by place p_0. As in the running example of Figure 4.5, we consider that there is a place p_0' with the same initial marking that p_0, thus it becomes implicit and it is not considered for the analysis (indeed, we omitted it in the Figure 8.3). The number of instances of each resource is summarised in Table 8.1, and they will be represented by tokens in the respective place.

The acquire (release) of a resource has been transformed into an immediate transition with an input (output) arc. For example, transition t_3 represents the acquire of the security host, while t_7 represents the release of such a resource.

Each one of the activities, self-messages in Figure 8.2, has been transformed into an exponential transition in the Petri net with its corresponding duration (given in Table 8.1). Each message exchanged through a net among two resources (e.g., *getToken()*) gives rise in the PN to an exponential transition (e.g., T_2) whose delay is that of the net involved (e.g., *$delayNet*). We have assumed that the operations/messages needed for establishing

[1] https://argospe.tigris.org

communication through the secure intranet are more expensive (in computing time terms). For this reason, we have set an upper delay for the secure intranet (*$intranetLag*) than for the insecure intranet (*$secIntraLag*). For simplicity, we have assumed the same delay for each message on the intranet communication independent from its size. process-idle place (p_0). Its values are shown in Table 8.1. The throughput of the system will be calculated by exact analysis when it can be computed, or by simulation otherwise.

8.2 Experiments and Discussion

In this section we test our approach by performing a set of experiments in the Petri net that accurately represents the SDBS. After applying our approach, the results obtained will be discussed.

8.2.1 Performance Estimation

We have carried out the regrowing strategy (Algorithm 2, Section 6.4.1) to estimate the throughput of the SDBS system with a different number of requests. The overall strategy has been implemented in MATLAB, while the throughput computation of the SDBS has been performed with the GreatSPN tool. The GreatSPN tool has been run in an Intel Pentium IV 3.6GHz with 2GiB RAM DDR2 533MHz host machine.

Table 8.2 shows the results obtained in the set of experiments with the parameters set as described above. The first column indicates the number of requests, followed by the number of *regrowing steps*. We have applied the name regrowing step to each iteration of the loop of the Algorithm 2. For each number of requests considered in the experiments, we have simulated the whole system. Such results are indicated in the first row of each experiment. The next column shows the size of the bottleneck (in terms of the number of places and transitions) produced by the algorithm and its percentage with respect to the total size. Then, the result of the upper throughput bound computed by the algorithm is shown. Such a bound is computed by solving the underlying Markov Chain when this is computationally feasible [Ajmone Marsan et al., 1995] or by simulating the net otherwise. Note that in the case of simulation, the upper throughput bound value is the mean of the simulation values, and the real upper throughput bound value is within an interval of $\pm 4\%$ with a confidence level of 95%. The next two columns show, in the first place, the percentage of increasing/decreasing improvement of one bound with respect to the previous upper throughput bound, and secondly, the accuracy of the computed bound with respect to the throughput of the whole system. The negative relative errors are caused by the confidence level and degree of accuracy used in the experiments. Finally, the last column shows the execution time consumed for computing the upper throughput bound of the PN system. We have distinguished whether the computation of the upper throughput bound has been achieved by exact analysis († symbol) or by simulation (no symbol).

Number of requests	Regrowing step	Size $\lvert P\rvert$ (%)	Size $\lvert T\rvert$ (%)	Through-put	Partial improvement	Bound error	Execution time (s)
15	(full system)	61 (100%)	56 (100%)	0.525685			$> +1$ day
	(initial bound)	56 (91.80%)	56 (100%)	0.551637	-	4.7045%	$5.87s$
	1	57 (93.44%)	56 (100%)	0.533037	3.3718%	1.3792%	$122.94s$
	2	58 (95.08%)	56 (100%)	0.522379	1.9995%	−0.6330%	$751.20s$
	3	59 (96.72%)	56 (100%)	0.522346	0.0063%	−0.6393%	$34256.97s$
20	(full system)	61 (100%)	56 (100%)	0.652313			$> +1$ day
	(initial bound)	56 (91.80%)	56 (100%)	0.735930	-	11.3621%	$5.80s$
	1	57 (93.44%)	56 (100%)	0.675957	8.1493%	3.4979%	$302.60s$
	2	58 (95.08%)	56 (100%)	0.637812	5.6431%	−2.2735%	$300.17s$
	3	59 (96.72%)	56 (100%)	0.637860	−0.0075%	−2.2658%	$3166.09s$
21	(full system)	61 (100%)	56 (100%)	0.671806			$> +1$ day
	(initial bound)	9 (14.75%)	9 (16.07%)	0.740741	-	9.3063%	$0.18s^{\dagger}$
	1	57 (93.44%)	56 (100%)	0.697133	5.8871%	3.6331%	$826.82s$
	2	58 (95.08%)	56 (100%)	0.653556	6.2509%	−2.7924%	$280.46s$
	3	59 (96.72%)	56 (100%)	0.653116	0.0673%	−2.8616%	$2216.06s$
22	(full system)	61 (100%)	56 (100%)	0.687808			$> +1$ day
	(initial bound)	9 (14.75%)	9 (16.07%)	0.740741	-	7.1459%	$0.18s^{\dagger}$
	1	57 (93.44%)	56 (100%)	0.713762	3.6422%	3.6362%	$2763.5s$
	2	58 (95.08%)	56 (100%)	0.666148	6.6709%	−3.2515%	$502.95s$
	3	59 (96.72%)	56 (100%)	0.667222	−0.1612%	−3.0853%	$1502.62s$
23 . . . 30	(full system)	61 (100%)	56 (100%)	0.700056			$> +1$ day
	(initial bound)	9 (14.75%)	9 (16.07%)	0.740741	-	5.4925%	$0.18s^{\dagger}$
	1	14 (22.95%)	13 (23.21%)	0.740733	0.0011%	5.4915%	$0.262s^{\dagger}$

Table 8.2: Experimental results for number of requests $\{15, 20, 21, 22, 23 \ldots 30\}$.

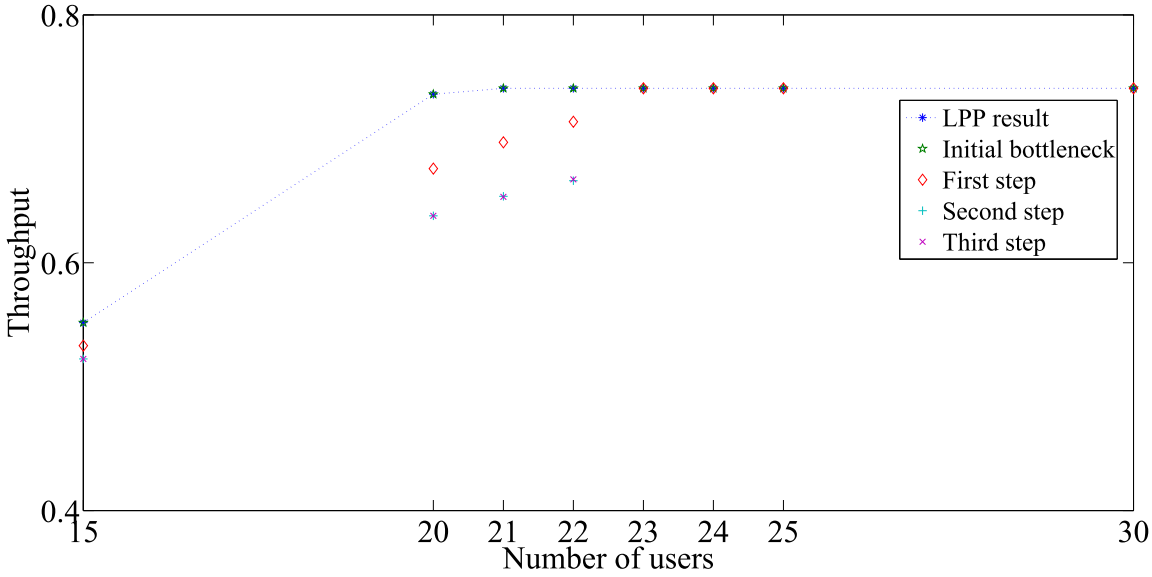


Figure 8.4: Throughput of the SDBS with variable number of users.

Note that in all cases the computation of the throughput of the whole system takes longer than one day of simulation time to finish, even though the evaluated system is an academic example. For larger systems, simulations may need a long convergence time, and therefore the usefulness of bounds computation is proved.

The degree of precision (ε) of the Algorithm 2 has been set to 10^{-3}. As can be observed, the initial bottleneck with the lowest number of requests (15, 20) corresponds to the underlying state machine (this is the result of removing resource places from the net in Figure 8.3). Again, this result indicates that the system's resources are well-dimensioned for attending to such a number of requests. In the case of 15 requests, in each iteration step there is no significant improvement (near to 6% in two iterations) and the regrowing strategy finishes in few steps. However, the greatest improvement occurs when the requests reach 20 units. In such a case, the first regrowing achieves an improvement near to 8%, reaching over 13% in the next iteration.

It is interesting to note what happens when the requests are increased to 21. For this value, the initial bottleneck is produced by one of the system's resources (specifically, the number of DB application hosts). This implies that the throughput bound of the system will remain the same for any number of requests over 21 (see *Average thr.* of first regrowing step for a number of requests greater than 21). In other words, requests will start waiting to be attended to if their number is equal to or higher than 21. Besides, note that when the number of requests is greater than 23, in the second iteration step there is an improvement in the upper throughput bound lower than 10^{-3}%.

As stated previously, the most significant improvement occurs when the number of

requests is 20. In just one iteration step, the initial throughput bound is improved by a value of nearly 8%. This indicates that the proposed method is more useful (i.e., it achieves a significant improvement in the upper throughput bound in few iterations) if the resources and requests are more well-balanced. Besides, it should be noted that the simulation of the whole PN becomes unfeasible for large systems, as indicated by the execution time.

The throughput results have been plotted in Figure 8.4. The throughput is drawn for each number of requests and for each step. Besides, the result of LPP (6.6) has also been drawn (dotted line). The LPP values match the throughput values of the initial bottleneck. As expected, the result of solving the LPP (6.6) (dotted line) is an upper bound of all the rest of the values. As can be seen, the improvement in the upper throughput bound for each regrowing step is almost insignificant in the case of requests lower than 20 or greater than 25. While the number of requests is near to 20, the relative difference between the throughput of the initial upper throughput bound and the first iteration becomes greater, reaching its maximum in the case of 20 requests. After that point, it becomes lower even tending towards a minimal difference near to zero (see, for instance, the case of 30 requests).

Finally, the execution time shown in last column in Table 8.2 indicates that the bigger the size of the net, the longer it takes to complete the simulation. Note that small additions to the net (i.e., just one place) normally cause an execution time of one or two orders of magnitude greater than previous executions. However, the improvement of the upper throughput bound is not so significant as to justify such an amount of execution time.

The main conclusions that can be extracted from both experiments can be summarised as follows:

- there exists a number of requests (*inflexion point*) at which the initially most restrictive p-semiflow of the system changes. Around such an inflexion point, the accuracy of the initial throughput bound is low. This occurs because when the slowest p-semiflow of the system is much slower than the others, it predominates over them and the system throughput is determined by the throughput of such a p-semiflow. The initial throughput bound is therefore usually quite accurate. However, when several p-semiflows have similar speeds, none of them predominates over the others. Hence the initial throughput bound, which considers just one p-semiflow, is less accurate;

- the improvement in the upper bound is specially significant in the proximity of the inflexion point.

As future work, we aim to continue researching into performance estimation based on performance bounds, seeking to obtain some quality bound characterisation. The use of LP problems and the token/delay ratio between p-semiflows in a PN system could be useful for this goal.

As the reader can imagine, it would be of great interest to be able to compute such inflexion points directly. This is the goal in the next set of experiments.

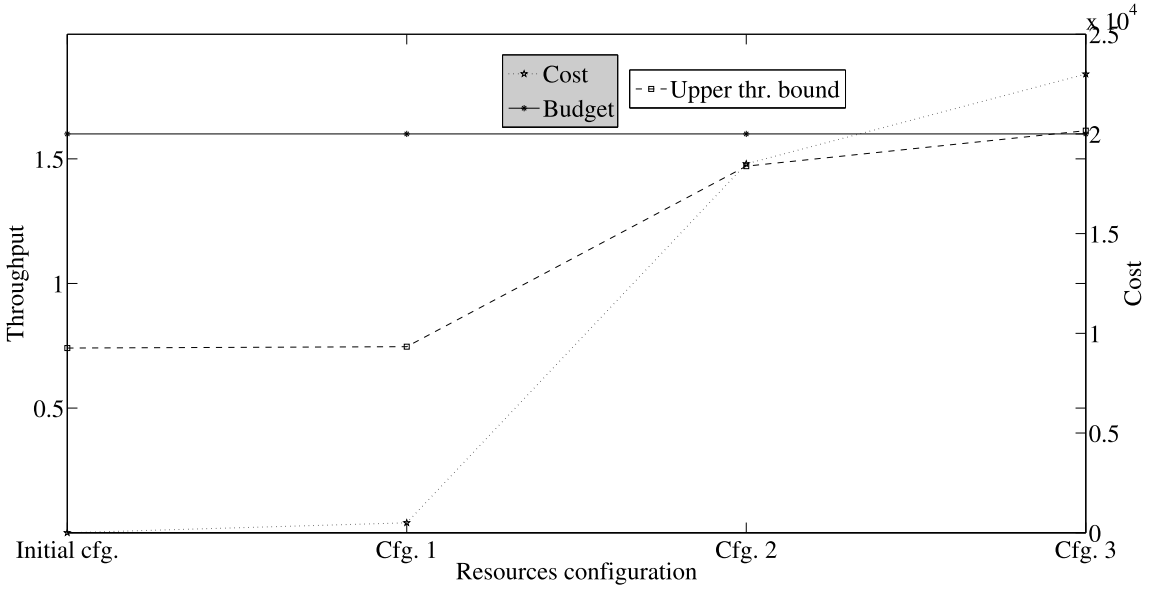


Figure 8.5: Different resources configurations and their associated cost.

8.2.2 Resource Optimisation Maximising Throughput

For these experiments, the number of requests has been set to $nRequests = 100$, whilst the initial number of resources remains unchanged: 5 security hosts, 10 policy hosts, 10 coordination hosts, 5 application hosts and 2 DB application hosts (summarised in Table 8.1). Let the budget be $\$20,000$ and the costs per resource be: $\$3,500$ per security host (represented by place p_7), $\$1,000$ per policy host (place p_{18}), $\$2,000$ per coordinator host (place p_{26}), \$500 per application host (place p_{28}) and \$500 per DB application host (place p_{31}). The prices of the hosts reflect either the cost of the physical hardware or the cost of reimplementing the services.

Applying the optimisation strategy introduced in Section 7.2.2, the initial restrictive resource is the number of DB application hosts, $\$nDBapps$ (initial tokens of place p_{31}). The algorithm in Figure 3 computes the new restrictive resource, the security hosts, and the number of DB application hosts needed to be increased (which is just one host). As the cost is \$500 per DB application host and there is a budget of $\$20,000$, the increase is possible. The strategy continues looking for the next restrictive resource. The second iteration gives as a result the new restrictive resource (application host) and the new instances of DB application and security hosts, respectively, 2 and 5 units. The increase of such resources has a cost of $\$18,500$, so it can be afforded. The new restrictive resource after the third iteration is the number of coordinator hosts. This time, it is necessary to increase the security hosts by 6 units, the DB application hosts by 3 units and the application hosts

by 1 unit with respect to the initial configuration. This last assignment has a cost greater than the initial budget, so the iteration process finishes and the previous assignment is taken as the valid one (5 security hosts and 2 DB application hosts). Moreover, there is no possibility of spending the rest of the budget (which amounts to $\$1,500$) Therefore, the optimisation strategy ends.

Hence, with the initial configuration and the given budget, the number of security hosts needs to be increased by 5 units and the number of DB application hosts by 2 units in order for the system resources to be optimally distributed and the throughput maximised.

Figure 8.5 plots the upper throughput bound (dashed line) of each configuration of resources, its associated cost in dollars (dotted line) and the total assigned budget (solid line). *Initial cfg.* (configuration) is 5 security hosts, 10 policy hosts, 10 coordination hosts, 5 application hosts and 2 DB application hosts. *Cfg. 1* refers to the increase by one unit of DB application hosts, whilst *Cfg. 2* indicates the last assignment of resources computed: the increase of 5 security hosts and of 2 DB application hosts. Finally, *Cfg. 3* refers to the configuration which cannot be afforded with such a budget ($\$20,000$): an increase in the security hosts by 6 units, the DB application hosts by 3 units and the application hosts by 1 unit with respect to the initial configuration. As can be observed in Figure 8.5, the cost of the last resources configuration exceeds the assigned budget, so the solution for the resource distribution is the previous configuration.

The evolution of the upper throughput bound is worth remarking. With the initial configuration, the upper throughput bound is $\Theta = 0.740740$. In the first configuration, the upper throughput bound increases by 0.75% ($\Theta = 0.746271$), while in the second configuration it increases by almost 100% ($\Theta = 1.470598$). Finally, with the third configuration the upper throughput bound increases by 9.68% ($\Theta = 1.612920$).

8.2.3 Resource Optimisation Minimising Cost while Adding FT Techniques

In this section, we consider the addition of a Fault-Tolerant (FT) technique PN-based model as described in Section 4.2, and we apply to the combined model the resource optimisation strategies presented in Section 7.3.

Consider that transition that represents an operation on data after reading the DB, T_{31}, may fail with a probability of 0.15. We decide to add a reinitialisation FT technique FT^1, without compensation phase and with a concurrent error detection that takes, on average, $\delta(T^1_{detect}) = 0.5ms$. The recovery time, i.e., the time needed for reconfiguring DB service takes, on average, $\delta(T^1_{rec}) = 20ms$. Lastly, place p_{36} (the one before faulty transition T_{31}) is labelled as $p_{36}|_{rtn}$.

The upper throughput bound of the system is, before adding the FT technique, $\Theta = 1.481481$, and it is associated to the minimal p-semiflow of p_{32} – i.e., `WS-DBApplication`. When adding the FT technique described, the minimal p-semiflows that are modified correspond to the ones that use T_{31}, i.e., $\mathbf{y}_{p_0}, \mathbf{y}_{p_2}$, $\mathbf{y}_{p_{29}}$ and $\mathbf{y}_{p_{32}}$, and the upper throughput

bound decreases near to a 133.98%, that is, $\Theta' = 0.633147$ and it is related as well to `WS-DBApplication`.

Let us apply now Algorithm 4 to compute the initial marking needed to compensate the throughput degradation. The minimal p-semiflows under study here are: $\mathbf{y}'_{p_0} = \mathbf{y}_{p_0} \cup \{^\bullet T_{31}, T_{31}^\bullet, p_4^1\}, \mathbf{y}'_{p_2} = \mathbf{y}_{p_2} \cup \{^\bullet T_{31}, T_{31}^\bullet, p_4^1\}, \mathbf{y}'_{p_{29}} = \mathbf{y}_{p_{29}} \cup \{^\bullet T_{31}, T_{31}^\bullet, p_4^1\}, \mathbf{y}'_{p_{31}} = \mathbf{y}_{p_{31}} \cup \{^\bullet T_{31}, T_{31}^\bullet, p_4^1\}$ (the other p-semiflows $\mathbf{y}''_{p_0}, \mathbf{y}''_{p_2}, \mathbf{y}''_{p_{29}}, \mathbf{y}''_{p_{31}}$ are not of interest due to $\delta_{detect} <= \delta_{31}$). The computation of value of $\mathbf{y}^1_{p_i} \cdot \mathbf{Pre} \cdot \mathbf{D}$ is, respectively, $41.9520, 41.6557, 10.3965, 9.3594$. Thus, the solution of Algorithm 4 is $\mathbf{m}'_\mathbf{0}(p_0) = 100, \mathbf{m}'_\mathbf{0}(p_2) = 50, \mathbf{m}'_\mathbf{0}(p_{29}) = 11, \mathbf{m}'_\mathbf{0}(p_{31}) = 10$. That is, the number of `WS-Application` (p_{29}) and `WS-DBApplication` (p_{31}) must be incremented to 11 and 10 units, respectively, to maintain the given throughput of $\Theta = 1.481481$ and a probability of error of 0.15. If resources are incremented as it is given by the solution of this algorithm, the new upper throughput bound has a value of $\Theta' = 1.567476$.

Let us consider that the addition of new resources has some associated cost, more precisely, the cost of adding new instances of any host service is \$350 each (for instance, because new licenses for deploying more virtual servers must be purchased). In the case of recovery method, it can be improved having a cost, on average, of \$250 per each millisecond, and the minimum required time for recovering is $5ms$ (i.e., $\delta_{min}(T_{rec}) = 5ms$).

With this configuration, we apply now the proposed ILPP (7.6) for computing the minimal cost that compensate a probability of error of 0.15. The result of applying ILPP (7.6) is that 4 more resources of `WS-Application` (p_{29}), 5 more resources of `WS-DBApplication` (p_{32}) and recovery time must be decremented in $2ms$. The cost associated to these actions is $\$3,650$. After applying these changes, the upper throughput bound is $\Theta'' = 1.500441$, which represents an improvement near to 1.28% of the previous upper throughput bound Θ.

Note that as the number of resources and the timing must be natural numbers, we will always obtain an upper throughput bound in the FT system where results of ILPP (7.6) are applied (slightly) better than in the original system model.

In summary, the solution of Algorithm 4 has an associated cost of $\$3,850$, because 11 more resources must be added, whilst the solution giving by minimising cost through ILPP (7.6) costs $\$3,650$.

8.3 Concluding Remarks

The formalism of Petri nets allows one to model the behaviour of a large class of artificial systems in which resources are shared by the different tasks. The performance of these systems, which is usually measured as the number of completed operations per time unit, is often a system requirement. Unfortunately, in most cases of interest it is not possible to compute the exact performance of a system in a reasonable time due to the state explosion problem inherent to large discrete systems. To overcome this issue, performance estimation

is based on bounds computations.

Chapters 6 and 7 propose several strategies for estimating efficiently the performance of a given system and for, given an initial budget and a cost of each resource, gauging the number of instances of each resource so that the system performance is maximised and the budget is not exceeded. Such strategies have been applied to a process Petri net modelling a Secure Database System in this chapter. The performance of such a system has been evaluated for different workloads, and a distribution of resources that maximises the throughput for a given budget has been estimated by using several algorithms that consider different initial conditions.

Chapter 9

Case Study: an E-Commerce System

This chapter introduces an E-Commerce System (ECS) where the model-based methodology previously introduced in Chapter 5 is applied. This case study has been used published in conjunction with the model-based methodology in [Rodríguez et al., 2012d].

9.1 System Description

The model-based approach introduced in Chapter 5 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 5.1) is depicted in Figure 9.1.

Figure 9.1(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 9.1(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 9.1(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 [OMG, 2009] 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 9.1(c). The verification of user credentials (*verifyUserCredentials* method) consumes, on average, 12.4ms, as specified in the `hostDemand` tagged value, see Figure 9.1(c).

9.2 Experiments and Discussion

In this section, we introduce the experiments we have carried out and discuss the obtained results. We firstly describe the experimental setting (see Section 9.2.1) of our case study: the step-wise application of the approach presented in Section 5.3 is discussed as well as the input parameters used for the experimentation. Then we collect the experimental results (see Section 9.2.2) 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.

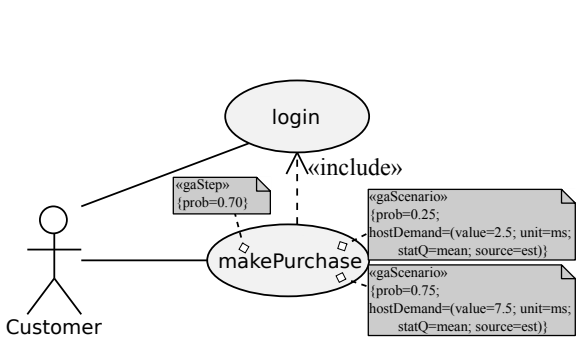
9.2.1 Experimental Setting

The first step of our approach (see Figure 5.1) is to annotate the performance-annotated application model by means of the SecAM profile [Rodríguez et al., 2010, Rodríguez et al., 2011] in order to specify attacks, vulnerabilities and intrusions in UML models.

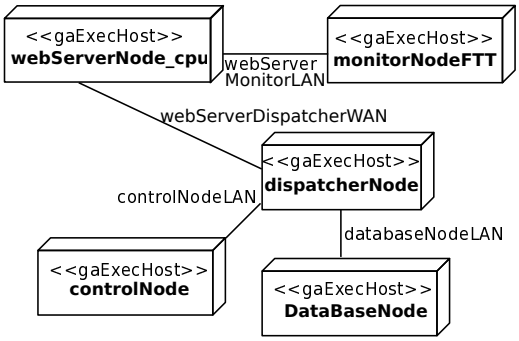
We assume that the *login* service can be the objective of external attackers that have the objective of bringing down the system while consuming its resources. The incoming requests are annotated through `secaAttackGenerator` stereotype describing an attack occurrence probability of rate *$attRate*. Hence, we obtain a *Security-Annotated Application Model* (see Figure 5.1).

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 5.1) is depicted in Figure 9.2.

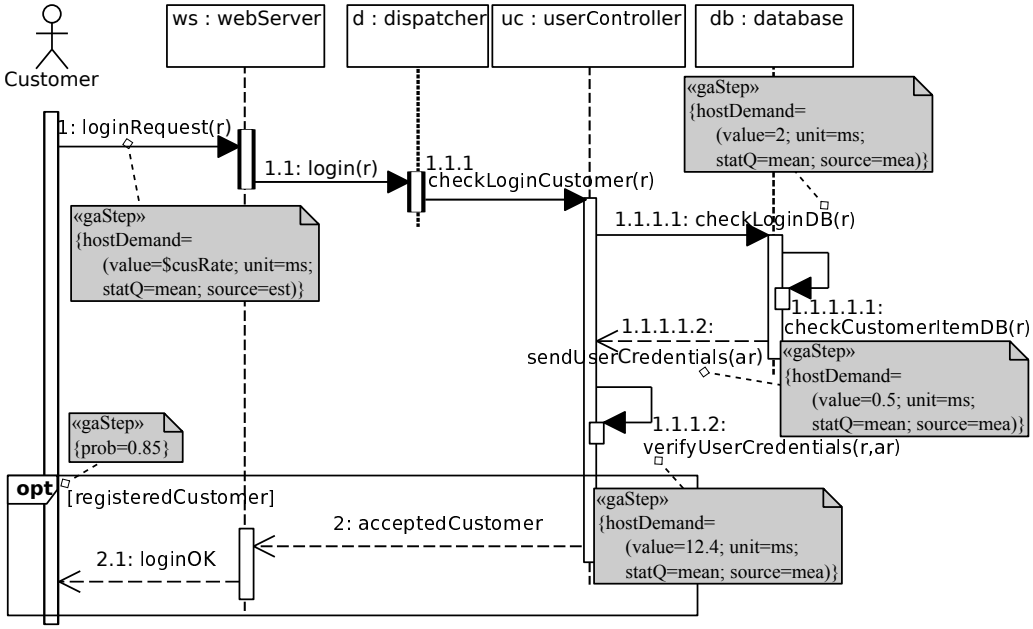
The activity of enabling Security Mechanisms (see Figure 5.1) leads to introduce SMs in the communication between the web server and the dispatcher. Figure 9.2 shows that before



(a) UML Use Case Diagram.



(b) UML Deployment Diagram.



(c) UML Sequence Diagram.

Figure 9.1: ECS Performance-Annotated Application Model.

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 5.1). We do not execute the performance analysis of such model because such experimentation has been already performed in [Cortellessa et al., 2010a], 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 5.1) leads to introduce FTTs in order to protect the web server. Both the FTTs we presented in Section 4.3.2 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 9.2, whereas a monitor node is intentionally added to implement the P&R technique, as reported in Figure 9.1(b). Hence, we obtained two *FTTs-Enabled Application Model*s (see Figure 5.1).

The input parameters used for our experimentation have been reported in Table 9.1 (system resources and number of their instances) and Table 9.2 (timing of system actions).

As already mentioned in Section 5.3, the definition of security parameters is embedded in the Security-Annotated Application Model (see Figure 5.1) 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 [Ariu et al., 2011, Sousa et al., 2010a, Menascé, 2003] in order to determine reliable numerical values, whereas application-dependent parameters come from the experimentation we conducted in [Cortellessa et al., 2010b].

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 [Ariu et al., 2011] (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 [Sousa et al., 2010a]. The timing values of the referenced UML-SD SMs have been chosen from [Cortellessa et al., 2010b] and [Menascé, 2003] 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

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

Table 9.1: Experimental 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 9.2: Experimental parameters: execution times of system actions.

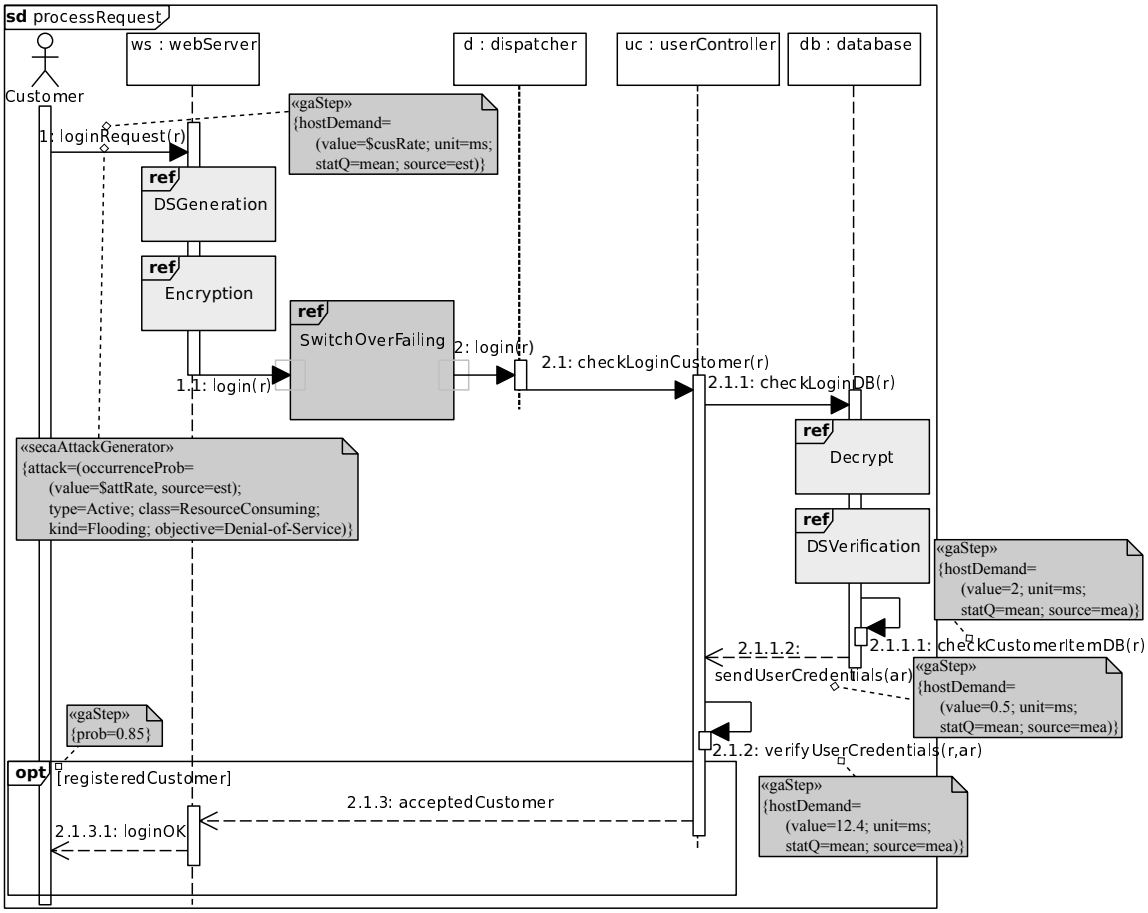


Figure 9.2: ECS SMs-FTTs-Enabled Application Model.

a probability of 0.3 and 0.7, respectively. As it is shown in Figure 9.1(a), a new purchase may have two different scenarios, each one with different duration.

9.2.2 Experimental results

The experimentation has been conducted while considering the following scenarios: (i) the Performance-Annotated Application Model (see Figure 9.1); (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 9.2); (v) the SMs-FTTs-Enabled Application Model (SoF + P&R), i.e., with SMs and both FTTs.

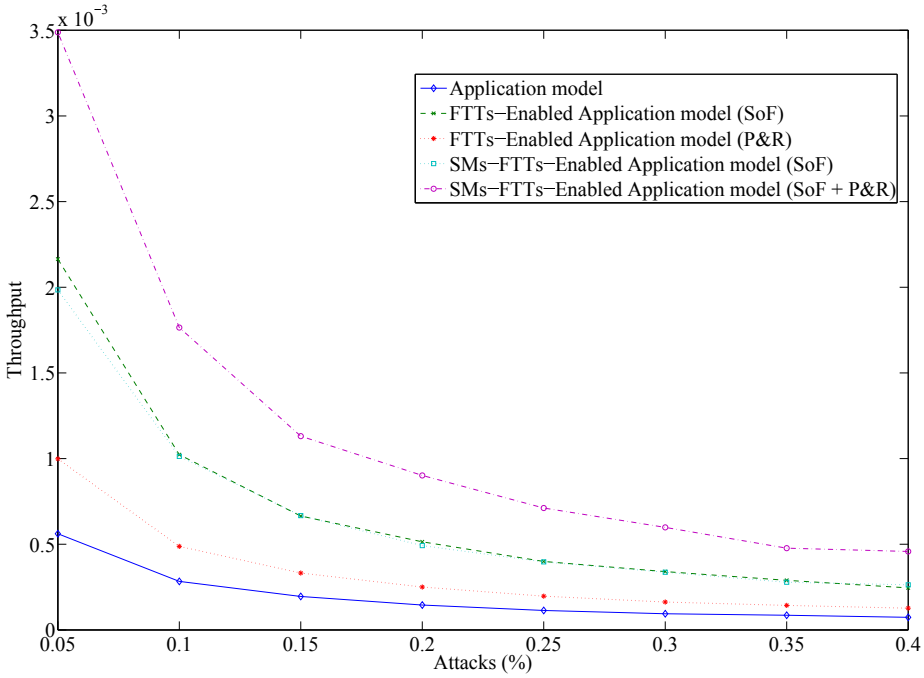
The transformation from UML software models to GSPN performance models has been carried out by ArgoSPE [Gómez-Martínez and Merseguer, 2006] tool. We have used the PeabraiN simulator (introduced in Chapter 11), which is a PNML-compliant tool and allows to simulate GSPNs in transient mode. We have simulated an execution of the system of 2 hours with the experimental parameters reported in Tables 9.1 and 9.2.

Figure 9.3 shows the experimental results. Figure 9.3(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.

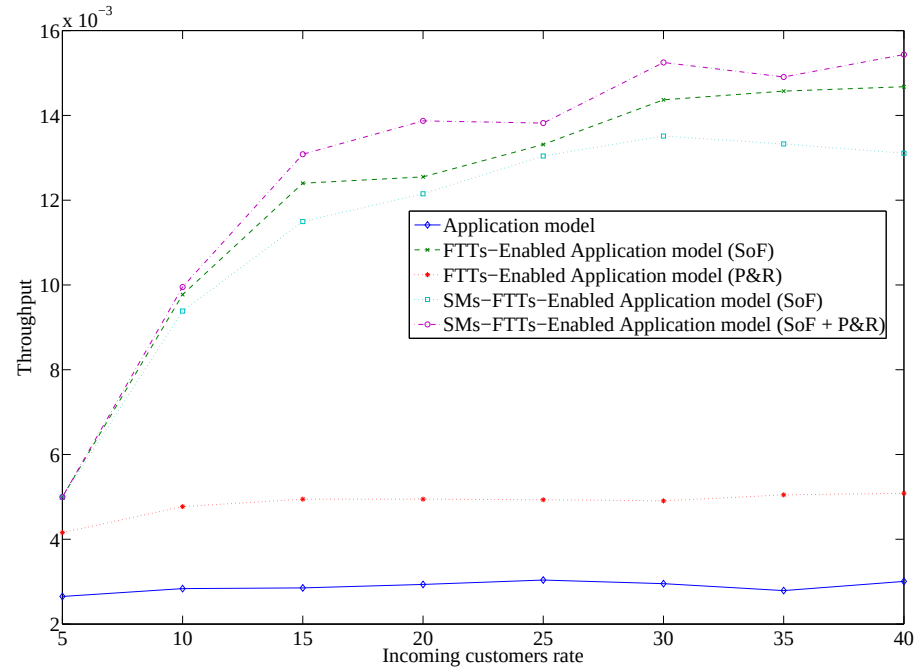
On the contrary, when FTTs are enabled, the system is able to mitigate the effects of attacks, maintaining a certain level of server availability. However, the throughput of the FTTs-Enabled Application Model (SoF) is greater than the throughput of the FTTs-Enabled Application Model (P&R). Finally, we can observe that 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 9.3(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



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



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

Figure 9.3: ECS Performance Analysis Results.

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 the goal of this chapter 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 experimental parameters. As future work, we plan to apply our approach to other real world examples in order to assess the scalability of the framework.

9.3 Concluding Remarks

In Chapter 5 we have introduced a model-based methodology for performance prediction of critical systems which combine Fault-Tolerant Techniques (FTTs) and/or Security Mechanisms (SMs).

In this chapter, we validated our proposal given in Chapter 5 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.

Chapter 10

Performance Analysis of Data-Intensive Workflows

This chapter addresses the main contributions of this dissertation related to performance analysis applied to a more specific domain, namely, scientific workflows. The major findings of this chapter have been published in [Rodríguez et al., 2012b, Rodríguez et al., 2012c] and [Rodríguez et al., 2013].

10.1 Motivation

Using workflow techniques, scientists can specify their computational experiments by means of a control/data flow graph, consisting of a set of tasks and the dependencies between them. Workflow enactors can subsequently interpret these specifications, enabling tasks in the graph to be mapped onto distributed resources. There are, however, several efficiency limitations of the workflow system in performance and resource usage [Park and Humphrey, 2008]. Such limitations may be due to limited parallelism within the application, or due to the workflow enactment engine. In data-intensive workflows, the enactors must be efficient in both mapping tasks to resources and in transferring large data files between tasks. Previous approaches exploit data location and link bandwidth information to minimise data movement or move data via higher capacity links whenever possible. Such an approach of moving the largest files via the highest-capacity links can result in sub-optimal workflow execution [Park and Humphrey, 2008].

This data movement policy generally involves moving the output data of a task to its successor node immediately after completing its execution. However, if a task needs multiple files to be made available before it can begin execution, it will remain idle until all the required data files from other predecessor nodes have been delivered. It is therefore not how fast each file can be moved to the task, but the interval from the delivery of the first file to the last one that is most significant. Even if the first file is delivered quickly, the task must

still remain idle until others are also available. In consequence, the effective use of network bandwidth and the buffer/storage at the receiving task is not made. If one file arrives too early taking up all of the network bandwidth for one task, it may be at a determent to other tasks (which may have to wait for their data to be delivered) – even though the receiving task still has to wait for other files. Similarly, if there is limited buffer capacity at the receiving task and the buffer needs to be shared between tasks, a quick delivery of one file (while waiting for other files to be delivered) for a task excludes other tasks from using the same buffer. In general therefore, the current practise of moving data from one location to another as early as possible is often either: (i) unnecessary when viewed in isolation (both in terms of networking and buffering): any data file that arrives much before the last needed data file or ii) harmful when viewed in-the-large: there is only finite capacity on each link and a limited buffer capacity – multiple concurrent data movement operations can significantly slow each other down, or an inappropriate buffer usage may lead to a buffer overflow. Park & Humphrey analysed this problem in [Park and Humphrey, 2008] and proposed a data-throttling framework that allows a workflow programmer/engine to describe the requirements on the data movement delay. This technique can be utilised to balance the execution time of workflow branches and eliminate unnecessary bandwidth usage, resulting in more efficient buffer and network usage. However, they do not propose any mechanism for analysing and automatically deriving the values needed to throttle data exchange between nodes involved in the transfer.

In this chapter, we firstly introduce a metric for quantitatively measuring the impact that applying an intelligent data movement policy can have on buffer/storage in comparison with existing approaches. This metric considers a workflow structure expressed as a Directed Acyclic Graph (DAG), and performance information collected from historical past executions of the considered workflow. It is intended for being used at the design-stage, comparing various DAG structures and evaluating their potential for optimisation (of network bandwidth and buffer usage). Then, we propose an automated analysis method for DAG that may be used to derive data-throttling values. The method utilises Petri nets for the workflow specification and combines the abstract representation with performance information instrumented from past executions, obtaining a performance model of the workflow: an iterative method is used to compute data-throttling values associated with different links within the workflow. Subsequently, a performance analysis is conducted using the throttling values and the result is compared with the performance achieved without throttling. Lastly, we introduce dynamism on the environment where a scientific workflow is executed, i.e., the network bandwidth of the links or/and the power of hosts machines where workflow tasks are executed significantly vary over the time.

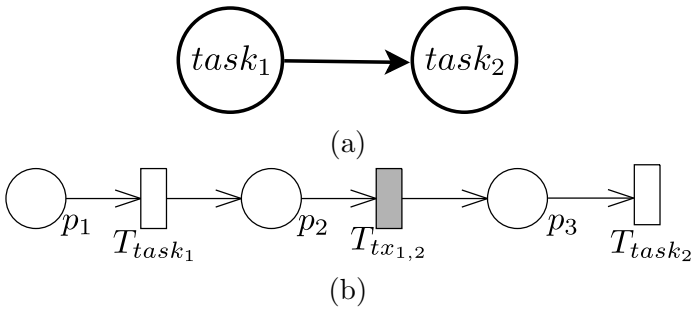


Figure 10.1: (a) Workflow tasks and (b) its transformation to PN.

10.2 Model Transformation: From a DAG to a SMG

Before going forward more in detail about performance analysis on workflows, let us explain in this section the model transformation that we perform on a Directed Acyclic Graph (DAG) by transforming it to a performance model based on Petri nets (PNs), namely, Stochastic Marked Graphs (SMGs, see Definition 8).

In this paper, we make use of SMGs with exponential random distributions associated with transitions in order to model scientific workflows. In particular, we are interested in workflows expressed as DAGs, where vertices represent tasks and edges represent data dependencies between them. Figure 10.1 depicts how we derive a PN model from a DAG. Figure 10.1(a) shows two workflow tasks of a DAG, $task_1$ and $task_2$ and a data-link dependence from $task_1$ to $task_2$, while Figure 10.1(b) illustrates the transformation to a PN. Note that such a PN model fulfils the definition of a SMG model: there are timed transitions and each place has exactly one input and exactly one output arc.

Hence, each task of the workflow DAG is transformed to a place and a transition (represented by a white rectangle), joined by an arc. A task transmission is also transformed to a place and a transition (grey rectangle). For instance, $p_1 \rightarrow T_{task_1}$ represents $task_1$ of the workflow. Note that place p_1 models the input buffer of $task_1$. Finally, the data dependency between $task_1$ and $task_2$ is modelled by adding a place and a transition, p_2 and $T_{tx_{1,2}}$, respectively. Transition $T_{tx_{1,2}}$ represents the time spent in sending output data from $task_1$ to the input buffer of $task_2$ (place p_3).

10.3 A Metric for Quantifying the Effectiveness of Throttled Data Transfers

A task in a workflow DAG cannot start its execution until all its inputs are available. The strategy of receiving these input values as fast as possible is often not appropriate. As some inputs may arrive earlier than others, these inputs have to be buffered locally at the task, resulting in unnecessary use of buffer space. If such buffer space is a shared resource and of

limited capacity, it remains blocked by the task, waiting for the remaining data to arrive. Hence, the greater the variation between arrival times of the different input data sets, the greater the inefficiency in buffer use. Intuitively, the objective of an effective data transfer is that each task with multiple inputs has all its data sets arrive simultaneously.

Our approach is therefore relevant for a workflow which has: (i) multiple synchronisation points (identified as tasks in the workflow containing more than one input, where all inputs are needed before the task can begin execution); (ii) difference in arrival times between the different inputs to such synchronisation point. The higher the value of (i) and (ii), the greater the possible optimisation we are likely to see with our approach. Both of these aspects depend on the structure of the workflow and the environment within which a workflow is enacted. Our approach could be used to re-write a workflow DAG that has a *structural imbalance*, i.e. a DAG containing multiple paths whose execution times differ significantly. Such an imbalance [Park and Humphrey, 2008] may also arise due to a scheduler binding tasks to resources, faults or unexpected performance degradation, such as a slow network connection or limited storage for the dataset.

In order to compute the metric, we convert the workflow DAG specification into a Petri net (as explained in Section 10.2), we subsequently feed the Petri net model with performance information on computational tasks, and network, as well as data size [Rodríguez et al., 2012b]. Petri net theory is subsequently used to analyse the Petri net model, and to obtain *slack* (μ) [Rodríguez and Júlvez, 2010] values (a key concept in our analysis, see Section 6.2). Intuitively, a *slack* is a positive value associated with each input link to a synchronisation (sync.) point and captures the time taken for an input data to be delivered to such a synchronisation point. The higher the slack, the more likely to have a higher input delay, thereby delaying the execution of the task at the synchronisation point.

A more formal description of a synchronisation point and the associated slack, in term of workflows, is as follows. Let W be a workflow represented as a cyclic Petri net and composed of a set T of tasks $T = \{t_1, \ldots, t_n\}, |T| = n$. Let ψ^t be the number of inputs of task $t \in T$. Let $T' \subseteq T$ be a set of tasks with multiple inputs, i.e., $\forall t \in T', \psi^t > 1$. A task $t \in T'$ is then called *synchronisation task* (or *synchronisation point*). Let $\delta_{i,j}$ be the time taken for an input $j \leq \psi^{t_i}$ to arrive at synchronisation task t_i. As each input j arrives at different times, we can determine the value of $max(\delta_{i,j})$ for a synchronisation task t_i that represents the time taken for the slowest arriving input.

Considering the entire workflow W, we can find the slowest path from the input to the output of the workflow, which also represents the workflow makespan $\mathcal{M}$ – represented as $\mathcal{M} = \sum_{t \in T} execution\ time(t) + \sum_{t_i \in T'} max(\delta_{i,j}), j \leq \psi^{t_i}$. The slack $\mu_{i,j} > 0$ for input $j \leq \psi^{t_i}$ of task t_i, can be calculated as:

$$\mu_{i,j} = \frac{(max_{j=1}^{\psi^{t_i}}(\delta_{i,j}) - \delta_{i,j})}{\mathcal{M}} \tag{10.1}$$

where $\mathcal{M}$ is the workflow makespan, i.e., $\mathcal{M} = \sum_{t \in T} execution\ \ time(t) + \sum_{t_i \in T'} max(\delta_{i,j}), j \leq \psi^{t_i}$.

The slack, therefore, is a measure of the time arrival of inputs with respect to the critical path of a workflow. Note that the slack value for the slowest input $max(\delta_{i,j})$ is always 0.

Let us present a metric α that makes use of *slack* theory and incorporates both structural and execution environment aspects. The metric α can be expressed as:

$$\alpha = |T'| \cdot \sum_{t_i \in T', j \leq \psi^{t_i}} \mu_{i,j} \tag{10.2}$$

where $\sum_{t_i \in T', j \leq \psi^{t_i}} \mu_{i,j}$, represents the sum of all the slacks that appear in the workflow. The value of α quantifies the potential benefit that a data-throttling strategy may achieve in comparison with a transmit as-fast-as-possible strategy. The metric indicates that the greater the number of tasks with multiple inputs, the greater the potential to save buffer/storage space. The second aspect the metric considers is the sum of all the slack values that appear in the workflow in case a transmit as-fast-as-possible strategy is applied. Ideally, an intelligent transfer strategy would make all the slack values equal to 0.

10.3.1 Metric Evaluation

In order to evaluate how our metric α measures the effectiveness of utilising a data throttling strategy, we conducted several experiments using the Montage [Berriman et al., 2007] workflow. This workflow is used for creating image mosaics in astrophysics (using data from different scientific instruments) and a specification of Montage structure can be found in Fig. 10.6. The structure of the workflow is quite regular, therefore the workflow imbalance, if any, must be due to the execution environment. However, Montage workflow structure depends on the number of input files to assemble. We considered 3 versions of Montage, with 25, 50 and 100 tasks, and having each workflow 5, 10 and 100 input files, respectively. A DAX description of this workflow, along with performance information collected from past executions is available at the Pegasus workflow system [Deelman et al., 2007] Web site[1].

We have simulated the enactment of these 3 workflows by making use of the SimGrid [Casanova et al., 2008] tool. We assumed an environment with the number of machines being the same as the number of nodes in the workflow. As for the network topology, we assumed a single output data-link per host, a network bandwidth of 100Mbps and a latency of 10^{-4}s.

For each Montage workflow, Table 10.1 shows the mean values and standard deviation, σ, of the buffer waiting time of the Montage tasks with multiple input data, namely *mDiffFit*,

[1] `https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator`

Montage task	25 tasks		50 tasks		100 tasks	
	Mean	σ	Mean	σ	Mean	σ
mDiffFit	1.7726	1.1800	1.5739	1.2044	1.9819	1.6399
mConcatFit	15.9589	–	57.5424	–	226.1896	–
mBackground	15.0307	1.1194	19.0069	1.4327	25.1063	1.8597
mImgTbl	2.9567	–	1.4934	–	4.0822	–

Table 10.1: Mean & standard deviation values of buffer waiting time for Montage workflow.

No. tasks	No. tasks with inputs	$\sum \mu$	Metric α
25	15	1.5173	22.7591
50	32	2.7681	88.5785
100	75	5.3371	400.2842

Table 10.2: Metric values computed for the considered Montage workflows.

mConcatFit, *mBackground*, and *mImgTbl*. A buffer waiting time was computed as the elapsed time between the arrival of the last and the first input. The results show that for some tasks, the higher the number of synchronisation points, the longer the buffer waiting times. The main reason for this is that the synchronisation points of larger Montage workflows have a larger number of inputs in comparison with smaller sized workflows. In particular, this fact can be observed in task *mConcatFit*.

The metric values for each considered Montage workflow are shown in Table 10.2. The first column indicates the number of total tasks in the workflow, while the second shows the number of tasks with multiple input dependencies. Finally, the third column, $\sum \mu$, represents the sum of slacks and the last column presents the value of the α metric computed. As it can be expected, and it is remarked by the buffer waiting time given in Table 10.1, the more value of α, the better expected reduction of waiting time by using a data-throttling strategy. For the Montage workflow, it can be then concluded that the higher the number of task in the Montage version, the more important a data throttling strategy will be, so that buffer occupancy can be utilised more effectively.

10.4 An Automating Data-Throttling Analysis Method

This section describes our analysis method, which consists of a sequence of four steps that are illustrated in Figure 10.2. As an input, the method receives a PN-based DAG workflow model (as described in Section 10.2), and performance information obtained from past executions (if any), or estimations provided by the user (i.e. annotations given in a DAX file). As an output, it generates throttling values for network and buffer usage as well as the impact of these values have on the overall workflow performance.

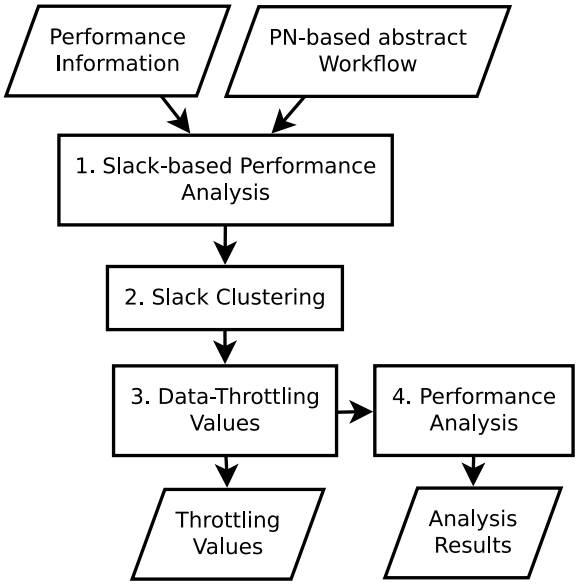


Figure 10.2: Automated Data-Throttling Analysis Flowchart.

The first step merges the PN model and the performance information (execution times for computational tasks and transfer times for transmission tasks) and builds a SMG model. Then, it performs the *Slack-based Performance Analysis* step described in previous sections. The critical path (i.e. the path with the longest delay) is obtained and slack values derived for each input link of a task (primarily for those tasks which have multiple input links) in the workflow.

As we parse the workflow graph, calculation of a data-throttling value at one data transmission may impact a calculation of a data-throttling at other data transmissions. Hence, if we start at the output node of a graph and work towards the input nodes, the slack value (used for computing data-throttling values) previously calculated along links close to the output node would be influenced by those closer to the input node. For instance, in the example DAG illustrated in 10.3, calculating data-throttling values for all incoming links at $task_6$ first and then moving on to $task_4$ would imply that if incoming links at $task_4$ would be delayed, then $task_4 \rightarrow task_6$ might be implicitly delayed and the previous adjustment done at incoming links at $task_6$ is no longer valid.

In order to minimise the complexity of this process, our approach classifies the data transfers in a workflow into groups of elements that are mutually independent and not affecting one another. We refer to this as *Slack Clustering* in Figure 10.2. In the *Data-Throttling Values* step, the method chooses one of the clusters and derives throttling values from the calculated slacks. Afterwards, the throttling values are used in the SMG model and the remaining slacks in the workflow are re-calculated accordingly. The process is repeated

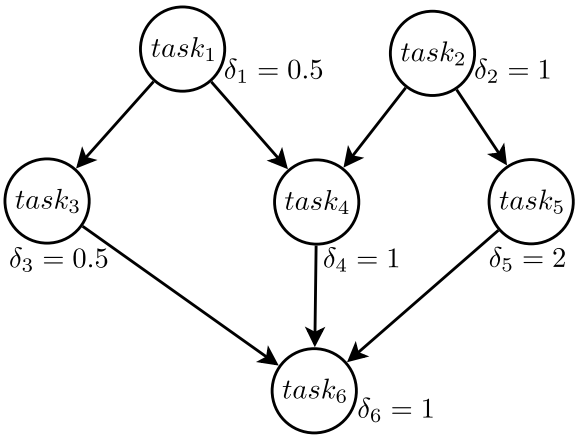


Figure 10.3: A workflow with 6 task and multiple inter-tasks dependencies.

until there are no more clusters to be regulated. For the computation of the throttling values, the third step considers a point-to-point network topology between tasks, i.e., all tasks are interconnected through dedicated links. Finally, the *Performance Analysis* step is carried out with (and without) data-throttling values to determine the impact of data throttling on the workflow makespan.

The steps of the method can be better illustrated using the workflow example of Figure 10.3. The derived Petri net model is shown in Figure 10.4. As for the performance information, for each computational task in Figure 10.3, δ_i represents the average execution delay for $task_i$. We assume that each task is executed on a different host (resource), therefore there is no *workflow imbalance*. For data transfers, we assume a dedicated network topology, i.e., each host is inter-connected to the others. The available bandwidth is considered to be 100Mbps with a latency of $1e^{-4}$s and the initial transfer policy is to transmit as fast as possible. For the sake of simplicity in the example, we also assume that all datasets are of equal size, 10MB.

The first step of the method, *Slack-based Performance Analysis*, obtains the workflow's slowest path: $task_2 \rightarrow task_5 \rightarrow task_6$ in Figure 10.3. Such a workflow has three different inputs where some slack will appear, namely $task_1 \rightarrow task_4$ (because $task_4$ needs data from $task_2$, and $task_2$ takes longer than $task_1$ to execute), $task_3 \rightarrow task_6$, and $task_4 \rightarrow task_6$ (in both latter cases, the slacks happen because of the delay of the input $task_5 \rightarrow task_6$). Using the SMG model in Figure 10.4, the following slack values are derived: $\mu(p_9) = 0.089392$, $\mu(p_{15}) = 0.178451$ and $\mu(p_{16}) = 0.357129$.

Let us consider that we avoid the *Slack Clustering* step and start the regulation by throttling the input $task_1 \rightarrow task_4$: the regulation would result in a decrease in transfer time for $task_1 \rightarrow task_4$, so that data elements from $task_1$, and $task_2$ arrive to $task_4$ simultaneously. At $task_4 \rightarrow task_6$: an effective regulation would involve increasing the

transfer time from $task_2 \rightarrow task_5$, while decreasing the transfer from $task_2 \rightarrow task_4$. Due to this regulation, the transfer time $task_2 \rightarrow task_4$ is modified, but the throttling value obtained in the the first regulation was obtained with the initial regulation, where the transfer $task_2 \rightarrow task_4$ was actually faster.

In order to avoid such an effect, we propose the *Slack Clustering* step, where inputs with slack that are not affecting each other are grouped together. For instance, in Figure 10.3, the input $task_1 \rightarrow task_4$ is at slack level 2 (because it contains the input $task_4 \rightarrow task_6$, which has some slack), while $task_3 \rightarrow task_6$ and $task_4 \rightarrow task_6$ are at slack level 1. In terms of the SMG model in Figure 10.4, the *Slack Clustering* step groups p_{15}, p_{16} in slack level 1, while p_9 is grouped in slack level 2. Such a slack clustering is therefore determined by the number of slack values in each workflow path. For instance, the number of slack places in the path $task_1 \rightarrow task_4 \rightarrow task_6$ is 2, i.e., $\mu(p_9), \mu(p_{15}) > 0$, where p_9, p_{15} belong to path $task_1 \rightarrow task_4 \rightarrow task_6$.

Updated data-throttling values are obtained in step *Data Throttling Values*, with all slack values being re-calculated after each adjustment of a cluster. The adjustment is done in increasing order of slack level (cluster), starting from level 1. Thus, in the example, we would increase the transmission bandwidth for $task_2 \rightarrow task_5$ (because it is the slowest path and if there is available bandwidth in the network link), and we would decrease the transmission bandwidth for $task_2 \rightarrow task_4$. As a result of conducting this step for the model in Figure 10.4, the following values are obtained: transmission between $task_1$ and $task_3$ (i.e., $task_1 \rightarrow task_3$) must be adjusted to a 28.57% of the total bandwidth (that is, to 28.57Mbps), while $task_1 \rightarrow task_4$ must be adjusted to 35.15Mbps, and $task_2 \rightarrow task_4$ to 44.55Mbps.

To summarise, data-throttling values are derived from the slack values as follows. A slack value $\mu(p)$ indicates that a transmission with duration tx_{old} must be delayed, i.e., $tx_{new} = tx_{old} + \alpha$. The value of α depends on the PN structure and relates to $\mu(p)$ and Θ (the upper performance bound of the PN model): $\alpha = \mu(p)/\Theta$. The bandwidth BW of a network link depends on the transmission time tx, $tx = d/BW$, where d is the size of the data sent through the network link. Hence, the new bandwidth BW_{new} can be computed as:

$$BW_{new} = \left(\frac{1}{BW_{old}} + \frac{\mu(p)}{\Theta \cdot d} \right)^{-1} \tag{10.3}$$

where BW_{old} is the current bandwidth assigned to that transmission.

Finally, *Performance Analysis* must be conducted with and without the obtained data-throttling values, so that the impact on performance can be determined.

Figure 10.5 shows three execution timelines of the previous workflow with different network topologies. Figure 10.5 (a) involves a single output link per node, Figure 10.5 b) corresponds to a point-to-point network topology (i.e., dedicated links), and Figure 10.5 c) is the same latter scenario but where data regulation has been undertaken. $Task_i$ represents the duration of each task i (white boxes), and $tx_{i,j}$ represents the time for sending data

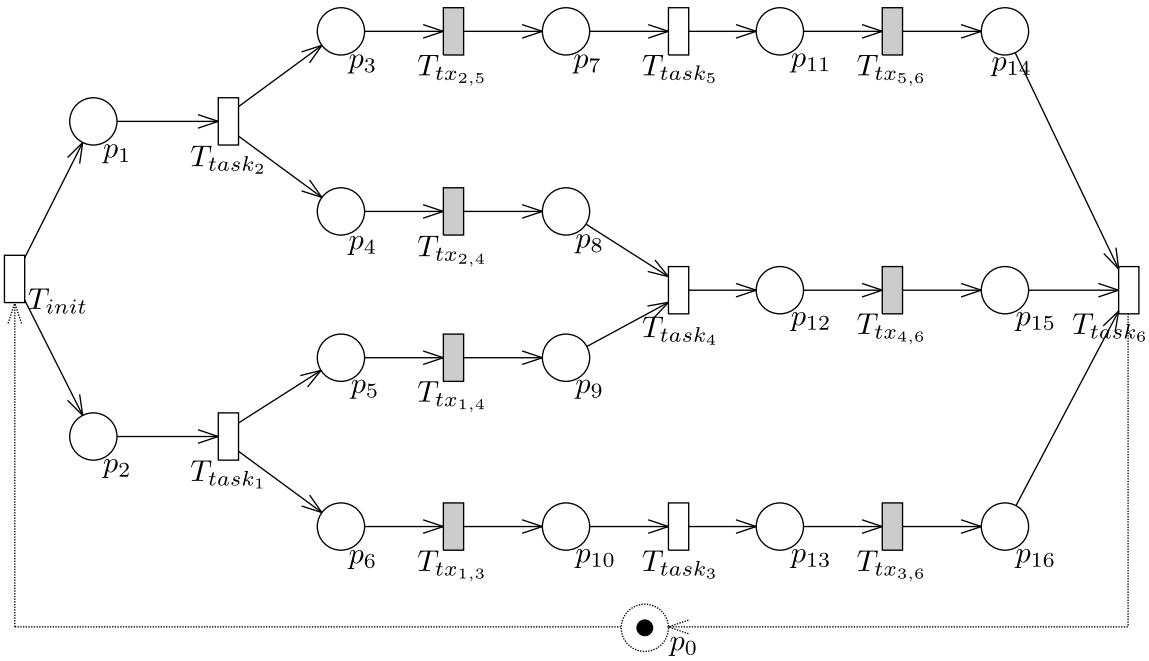


Figure 10.4: PN-based abstract workflow with explicit data-transfer transitions.

from $task_i$ to $task_j$ (grey boxes). The symbol $\Delta_{i,j}$ means the waiting time for the dataset sent from $task_i$ in the input buffer of $task_j$, until $task_j$ begins execution. Although data throttling does not significantly impact makespan (5.6779 seconds versus 5.7750 seconds), it can be noticed that it does impact the input buffer waiting times for $task_4$ and $task_6$.

10.4.1 Experiments and Discussion

We have used the Montage [Berriman et al., 2007] workflow (see Figure 10.6) for evaluation – a real application that has been used to create image mosaics in the astrophysics domain. A description of the abstract workflow and performance information were collected from DAX files generated with the Pegasus workflow system[2]. We have performed two different experiments. The first one determines the impact of bandwidth throttling on the workflow makespan, while the second one shows how buffer and network bandwidth utilisation change.

Our strategy has been implemented in Matlab, whilst simulation of workflow execution has been carried out through SimGrid [Casanova et al., 2008]. SimGrid is a toolkit that provides core functionalities for the simulation of distributed applications in heterogeneous distributed environments. We utilised SimGrid as it provides direct support for DAX files.

[2] `https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator`

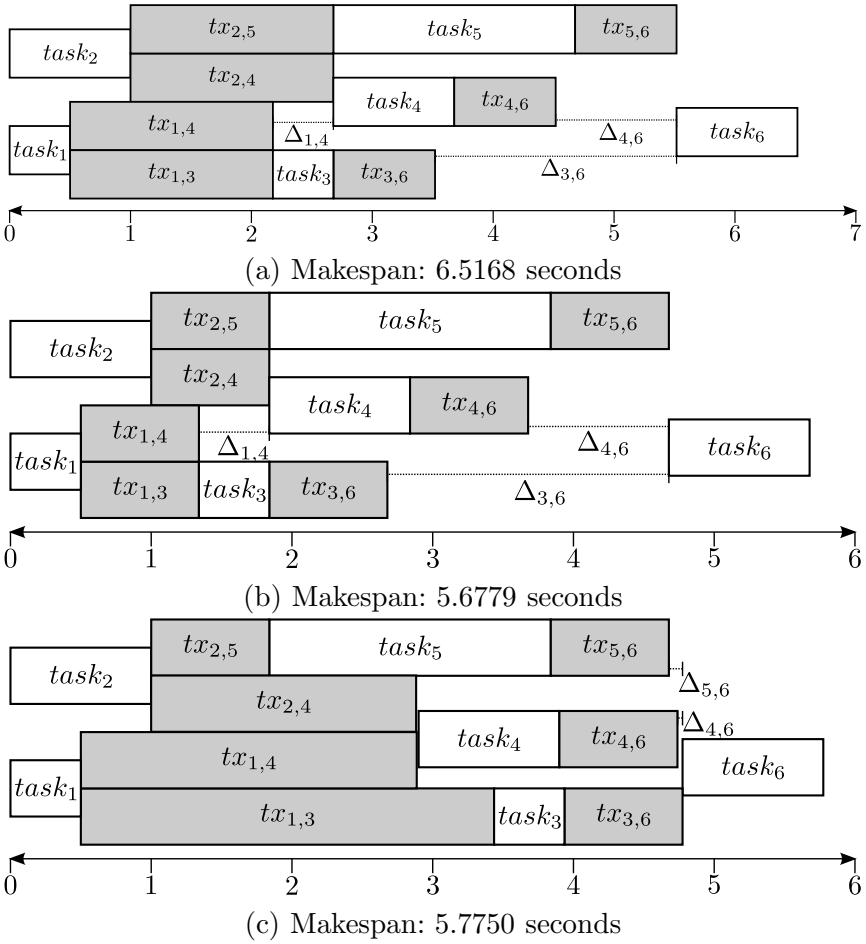


Figure 10.5: Makespan of workflow depicted in 10.3 with different network topologies.

Network topology	Network bandwidth		
	10Mbps	**100Mbps**	**1Gbps**
Single output	193.20	61.18	47.98
PP without BW throttling	153.15	57.17	47.58
PP with BW throttling	153.32	57.21	47.58

Table 10.3: Makespan for Montage workflow with 5 input files.

For the experiments, we have used the Montage workflow for 5 input files, composed of 25 tasks. Each task is considered to be executed on separate hosts. Moreover, we have assumed two different network topologies between hosts on the experiments: (1) each host has just one output data-link; or (2) each host is connected point-to-point (PP) to other hosts. Bandwidth throttling (BW throttling) has been considered only for topology (2). We have considered three different data-link bandwidths: 10Mbps, 100Mbps and 1Gbps. The experiments have been executed on an Intel Pentium IV 3.6GHz with 2GiB RAM DDR2 533MHz host machine.

10.4.2 Impact on the Workflow Makespan

Table 10.3 summarises experimental results showing the impact on makespan. The first column indicates the network topology, the others show the different network bandwidth used in the experiment: 10Mbps, 100Mbps and 1Gbps. The network latency is assumed to be constant and equal to 10^{-4} seconds for all network topologies. The workflow makespan (in seconds) is calculated based on the simulation time from SimGrid.

A point-to-point (PP) connection *without* throttling is 26.15% faster than a single output data-link connection in the first case (10Mbps), and only 0.84% in the last case (1Gbps). Additionally, in all cases a PP connection *with* throttling is slower than without throttling. As the bandwidth increases, the impact on the workflow makespan becomes smaller. Park & Humphrey [Park and Humphrey, 2008] also indicated a correlation between $\frac{data\ transmission}{computation}$ ratio and makespan, which our results confirm.

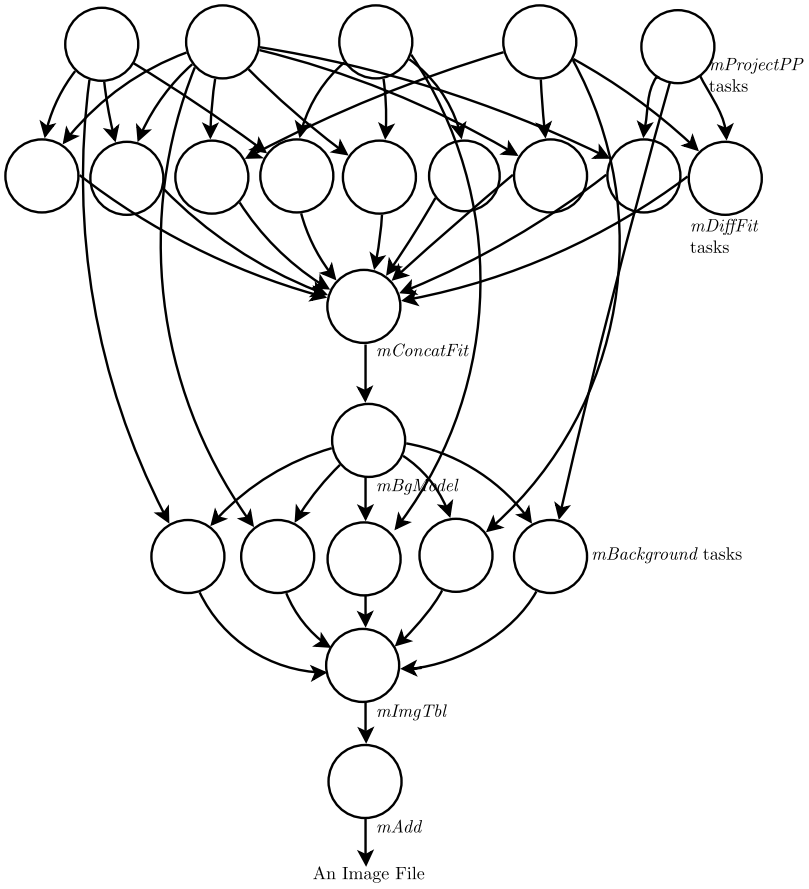


Figure 10.6: Montage workflow for 5 input files.

Montage task	Network topology (1)	(2)	(3)
$mDiffFit_1$	20.49	0.45	0.22
$mDiffFit_2$	20.49	0.45	0.21
$mDiffFit_3$	20.24	0.22	0.14
$mDiffFit_4$	0.09	0.05	0.23
$mDiffFit_5$	20.59	0.49	0.24
$mDiffFit_7$	20.24	0.23	0.14
$mDiffFit_8$	26.84	0.08	0.09
$mDiffFit_9$	6.60	0.15	0.08
$mConcatFit$	124.76	4.20	1.57
$mBackground_1$	33.84	13.80	7.02
$mBackground_2$	13.35	13.35	6.61
$mBackground_3$	33.94	13.84	6.55
$mBackground_4$	33.59	13.57	6.31
$mBackground_5$	40.19	13.43	6.45
$mImgTbl$	1.63	1.63	0.75

(a) 10Mbps

Montage task	Network topology (1)	(2)	(3)
$mDiffFit_1$	2.44	0.44	0.21
$mDiffFit_2$	2.44	0.44	0.21
$mDiffFit_3$	2.23	0.23	0.12
$mDiffFit_4$	0.04	0.03	0.02
$mDiffFit_5$	2.48	0.47	0.23
$mDiffFit_7$	2.23	0.23	0.11
$mDiffFit_8$	2.73	0.05	0.03
$mDiffFit_9$	0.50	0.18	0.09
$mConcatFit$	15.96	4.06	1.96
$mBackground_1$	15.50	13.49	6.91
$mBackground_2$	13.05	13.05	6.45
$mBackground_3$	15.53	13.52	6.48
$mBackground_4$	15.28	13.28	6.30
$mBackground_5$	15.78	13.11	6.31
$mImgTbl$	1.60	1.60	0.32

(b) 100Mbps

Montage task	Network topology (1)	(2)	(3)
$mDiffFit_1$	0.64	0.44	0.24
$mDiffFit_2$	0.64	0.44	0.24
$mDiffFit_3$	0.43	0.23	0.11
$mDiffFit_4$	0.03	0.03	0.01
$mDiffFit_5$	0.67	0.47	0.25
$mDiffFit_7$	0.43	0.23	0.14
$mDiffFit_8$	0.32	0.05	0.05
$mDiffFit_9$	0.11	0.18	0.11
$mConcatFit$	5.19	4.05	1.95
$mBackground_1$	13.66	13.46	6.73
$mBackground_2$	13.02	13.02	6.42
$mBackground_3$	13.69	13.49	6.61
$mBackground_4$	13.45	13.25	6.31
$mBackground_5$	13.34	13.07	6.44
$mImgTbl$	1.60	1.60	0.21

(c) 1Gbps

Topologies: (1) Single output; (2) PP w/o. BW throttling; (3) PP w. BW throttling.

Table 10.4: Buffer waiting time of Montage tasks under different configurations.

In this section we measure the impact of data throttling on the input buffer occupancy and network bandwidth. In order to fulfill this goal, we consider the inter-arrival time between data items (e.g. files) to the same task. The results of this experiment with three different network bandwidths (10Mbps, 100Mbps and 1Gbps) are shown in Table 10.4((a), (b) and (c), respectively). The first column indicates Montage tasks which need more than one input to execute. The following columns show the waiting time of input data in the buffer until the task begins its execution with different network topologies: one single output data-link per host, a PP connection between hosts without (and with) BW throttling.

Hence, bandwidth throttling has a great impact on the buffer waiting time of the tasks. Bandwidth throttling outperforms both single out and PP in all cases, with $mDiffFit_4$ for 10Mbps network being an anomaly. Note that even the bandwidth throttling is not enhancing the buffer waiting time for $mDiffFit_4$, such buffer waiting time is strongly improved in the case of the task *mConcatFit* (the next task in the workflow to execute after $mDiffFit_4$): it is reduced by 98.74% with respect to the single output data-link topology and 62.61% with respect to PP without bandwidth throttling.

When the network bandwidth is 1Gbps, the results of single output data-link and PP without bandwidth throttling are quite similar, whilst applying the bandwidth throttling still improves the buffer waiting time of the workflow tasks.

We have plotted the results of tasks with the greatest number of input dependencies in the Montage workflow (in this case, *mConcatFit* and *mImgTbl*) in Figure 10.7((a) and (b), respectively). As it can be observed, performing a bandwidth throttling outperforms both other network topologies in the experiment.

We can summarise our experimental findings as:

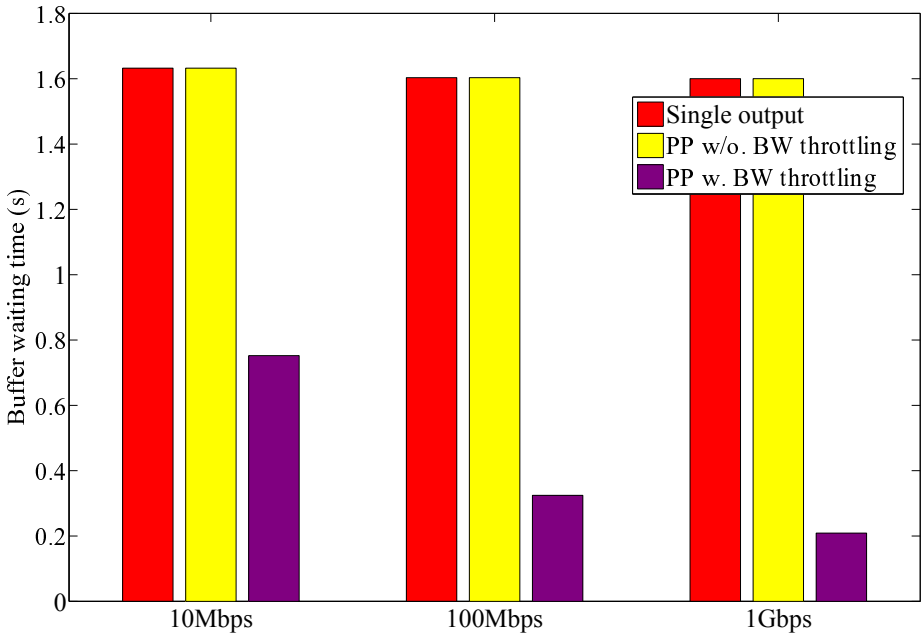
i) there exists an intrinsic relationship between the transmitted data size and workflow task execution time which can be useful for deciding when bandwidth throttling would be most appropriate;

ii) applying data throttling in a PP network topology does not make a significant change to the overall workflow makespan (compared to other topologies); but

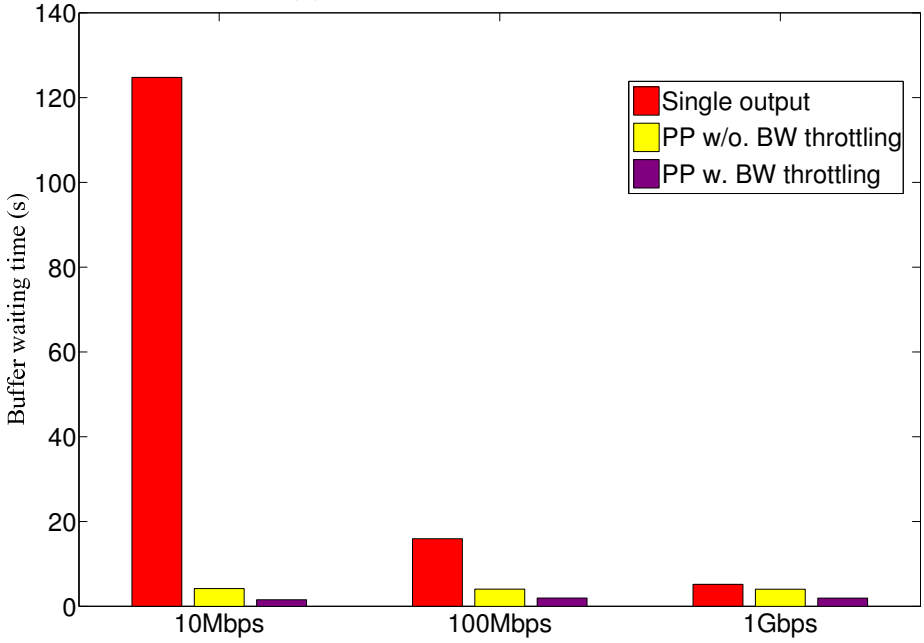
iii) performing data throttling has a great impact on the input buffer and network bandwidth usage, as outlined in our results.

10.5 Concluding Remarks

Current scientific problems usually demand a large amount of data transmissions and complex data analysis. Scientific workflows have become the computational technology of choice for solving such problems. However, scientific workflows can have data dependencies which neglect their intrinsic parallel behaviour. Previous approaches attempt to either avoid data movement completely (i.e. by co-locating tasks or moving them closer to the data source), or move data as fast as possible (based on the network link capacity). This



(a) Workflow task *mImgTbl*



(b) Workflow task *mConcatFit*

Figure 10.7: Buffer waiting time in (a) task *mImgTbl* and (b) *mConcatFit*.

last approach, however, can lead to some workflow tasks waiting for incoming data, while blocking use of shared buffers that could be better utilised.

In this chapter, we have proposed a *quantitative* metric based on the workflow structure, and on performance information derived from past historical executions. We convert the DAG specification into a Petri net model, feed it with such a performance information, and conduct analysis over it to obtain task inputs with *slack*. A slack is a positive value that when computed, appears at a task input that is likely to be idle, waiting for other datasets to arrive. The higher the slack value, the more likely the associated input will have to wait for a longer period. Our metric is proportional to the number of synchronisation points in the workflow and to the sum of all slack values appearing in the workflow. This metric is intended for use at the design-stage, to compare various DAG structures and evaluate their *potential* for optimisation (of network bandwidth and buffer use).

We also study a data-throttling strategy for improving the use of bandwidth and input buffers, so that tasks have all their inputs arriving at the same time (buffer usage minimised). In order to achieved this, some transmissions have their speed modified (either reduced or increased), making a better usage of bandwidth. The strategy takes as an input a Petri net-based model of a workflow. Our approach makes use of linear programming techniques for which polynomial algorithms exist. As an output, the method obtains the throttling data values, but it also analyses the affection that data-throttling can have on workflow performance. The affection depends on the workflow structure, and on the ratio between computational and transmission and tasks. By means of this analysis, the user can establish a trade-off between throttling (and possible degrading the performance) or not. We have tested our approach with Montage, a real scientific workflow from astrophysics, supposing different data-link speeds. The results are very promising, and show us how our approach enables a more effective use of bandwidth, and reduces the waiting time of data in input buffers.

We conducted experiments over 3 Montage workflows, with different sizes. For each synchronisation point in the workflow, we measured the buffer occupancy time and obtained our metric. We observed that the higher the value of the metric, the higher the buffer occupancy time within the Montage workflows. The throttling of data becomes more important with the growing size of the Montage workflow. Due to its characteristics, a Montage workflow with a larger number of tasks suffers from larger buffer occupancy, and this will require more buffering space for the inputs. Although demonstrated through a single workflow our approach is quite general in scope and can be applied to any workflow described using the DAX representation.

Part IV

Tool Support

Chapter 11

The PeabraiN Tool: A PIPE Extension

This chapter introduces PeabraiN, which stands for *"Performance Estimation bAsed (on) Bounds (and) Resource optimisAtIon (for Petri) Nets"*, and is a PIPE extension for performance estimation and resource optimisation. PeabraiN has been presented in an exhibition tool session at the *12th International Conference on Application of Concurrency to System Designs* and also as a paper in the same venue [Rodríguez et al., 2012a]. It has also been used for performing some of the experiments of the previous chapters of this dissertation, and it is currently being promoted in other related research groups for its usage.

11.1 Motivation

Many discrete systems can be modelled in terms of Stochastic Petri Nets (SPNs) [Molloy, 1982]. Such systems may need the use of shared resources. Two studies that are often of interest are: (i) the performance evaluation (or *throughput*, defined as completed jobs per unit of time), and (ii) the resource optimisation, i.e., to have optimally sized the number of shared resources in the system.

Exact performance evaluation may become unachievable, in terms of computation time, due to the need of an exhaustive exploration of the state-space. Normally, the larger the system, the bigger its state-space. An alternative is to estimate the performance by computing performance bounds [Campos and Silva, 1992, Liu, 1995, Rodríguez and Júlvez, 2010].

Resource optimisation is another master key when designing these systems. When resources are not well-dimensioned [Goldratt and Cox, 1986], it may happen that either the throughput is constrained by lack of available resources (then performance is lower than it could be), or there are idle resources (then money has been squandered).

In this chapter, we present PeabraiN, a collection of PIPE [Bonet et al., 2007] tool-compliant modules for performance estimation and resource optimisation based on bounds

computation for SPNs. The algorithms supporting such modules, which have been previously presented in Chapters 6 and 7, intensively use linear programming (LP) techniques, then assuring low computational complexity. Besides, other modules have been added for computing other properties based in LP techniques, such as the computation of structural enabling(marking) bound at a transition(place). Visit ratios computation and SPN simulation analysis modules have been integrated to PIPE as well.

We studied different choices for implementation: implementation of a stand-alone MATLAB application; extension of the GreatSPN [Baarir et al., 2009] tool; extension of the HISim tool; and to develop modules to be integrated in PIPE tool.

The only tool for performance bound computation, to the best of our knowledge, is GreatSPN, which computes lower and upper throughput bounds of transitions. The extension of GreatSPN was finally rejected because of the programming language paradigm used, and its platform dependency. All bound computation algorithms presented in this paper were initially developed for MATLAB. Nevertheless, a final deployment of this solution was ruled out by the dependency of a proprietary software library (namely, MATLAB Component Runtime library). A solution deployed over HISim was rejected when we figured out the easiness of extension through modules directly over PIPE tool. For resource optimisation, as far as our knowledge, there does not exist any tool.

PIPE was chosen because (i) we need only SPNs and not other PN extensions, (ii) it uses the standard PN file format, Petri Net Markup Language (PNML) [Hillah et al., 2009], so it allows an interchange of files between different PNML-compliant tools, (iii) PIPE facilitates a user-friendly GUI editor, (iv) it is multi-platform, and (v) it is open source.

11.2 PeabraiN Framework

This section describes the PeabraiN framework in more detail. Firstly, the implemented features and the PeabraiN design are described. Then, an illustrative example of use of PeabraiN is shown. Lastly, tool availability and installation requirements are introduced.

11.2.1 Implemented Features

PeabraiN provides two main features which implement:

i) an iterative algorithm for performance estimation based on linear bound computation (see Section 6.4); and

ii) a heuristic method to distribute shared resources in order to enhance the system performance as much as possible (see Section 7.2.2).

Lastly, PeabraiN adds other features to PIPE as side effect from the algorithms above mentioned:

- **Lower (upper) throughput bound.** The algorithms given in [Campos and Silva, 1993] for computing the lower (and upper) throughput bound for a SPN, in terms of LP problems, have been implemented. The PN structure needs to fulfil a set of conditions so that the computation of performance bound has some meaning, namely (i) the PN must be structurally live, (ii) structurally bounded, (iii) have a home state and (iv) its vector of visit ratios must have a unique solution. As some of these properties are already fulfilled depending on the PN subclasses, and moreover some of them are NP-decidability problems, we just automatically check the latter property.

- **Slowest p-semiflow.** The LP problem presented in [Campos and Silva, 1993] allows to compute the slowest p-semiflow of a PN, and its throughput which is an upper performance bound for the real system performance. The PN must fulfil the same conditions than in the previous algorithm.

- **Structural marking** and **structural enabling.** The structural marking of a place p, and the structural enabling of a transition t, can be computed by using LP problems [Campos and Silva, 1993]. Such algorithms work for any kind of PN.

- **Visit ratios.** The vector of visit ratios v of a PN, normalised for a transition $t \in T$ can be computed as described in Section 2.1.2.

- **SPN Simulation.** A simulator for SPNs using the Gillespie's stochastic simulation algorithm [Gillespie, 1976] has been implemented. It performs a set of replications of the simulation, and estimates the average throughput with a given confidence interval level and error accuracy.

11.2.2 Framework Design

PeabraiN is made of a set of modules compliant with PIPE-tool modules. As PIPE, PeabraiN has been implemented in Java, and it uses the same libraries as PIPE, and additionally, the Java Interface for LP solvers (Java ILP)[1] library, the Stochastic Simulation in Java (SSJ)[2] library, the Java Matrix (JAMA)[3] library for performing computational operations in matrices and LP solver-specific interface for Java. Hence, such a collection of modules perfectly fits in PIPE tool.

PeabraiN has been designed as a closed architecture by layers, i.e., each layer only calls methods of the immediate lower layer modules (see Figure 11.1). Each of these three layers matches with each component of the Model-View-Controller (MVC) architectural pattern. It has been developed on the top of the Java Runtime Environment (JRE) and some other external libraries as indicated above.

[1] http://javailp.sourceforge.net/
[2] http://www.iro.umontreal.ca/~simardr/ssj/indexe.html
[3] http://math.nist.gov/javanumerics/jama/

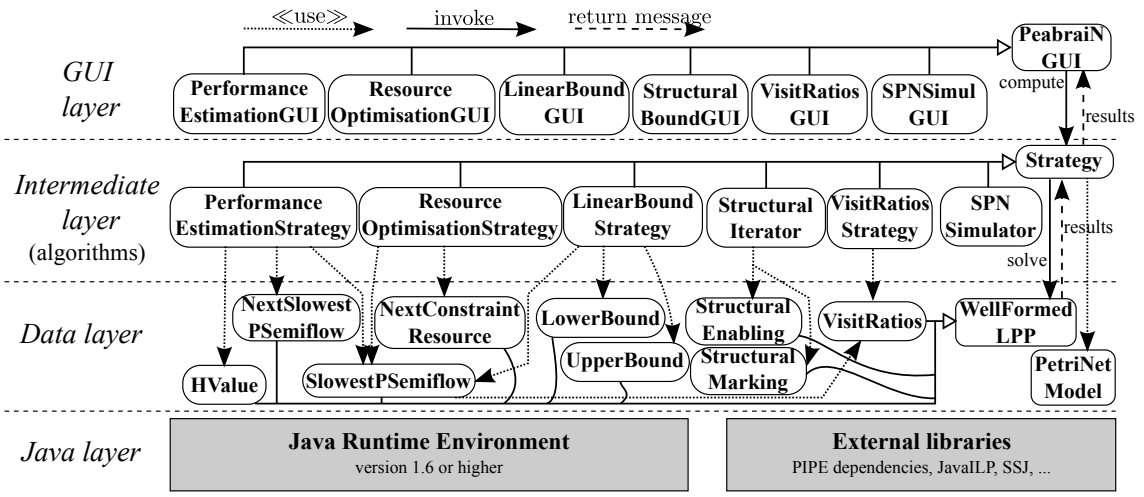


Figure 11.1: PeabraiN software architecture.

The *data layer* contains classes representing the information needed for the algorithms to execute. For instance, the PetriNetModel class represents a PN in its matrix form, and implements several methods related to PN (such as getting the initial marking at a place or getting the rate of some transition). The rest of the classes in this layer represent constraints, they are needed either for the LP problems of the Algorithms 1 and 3 or for the features presented in Section 11.2.1. The WellFormedLPP class is a super-class of the rest of classes in this layer.

The *intermediate layer* encloses the classes that implement the algorithms and features explained in Section 11.2.1. Solid arrows mean that a class invokes methods of another class, while dashed arrows represent the method return messages. For instance, PerformanceEstimationStrategy implements algorithm in Figure 2, and invokes the LP problem for computing the slowest p-semiflow and the LP problem for computing the next slowest p-semiflow. Strategy is a super-class that allows all its child-classes to manage the PetriNetModel in matrix form. The classes in this layer call the solve method of the classes in the data layer through the WellFormedLPP class. The dotted arrows connect a class in this layer with the classes in the data layer that it actually uses.

Finally, the *GUI layer* has classes which create the graphic interfaces for collecting, from the user, information for execution of the algorithms and also to show the results. They invoke the classes in the intermediate layer. For example, Figure 11.4(b) is an instance of the ResourceOptimisationGUI class, it allows to introduce the necessary parameters and after computation shows the results and the execution time.

Figure 11.2 shows the integration of PeabraiN in PIPE. PIPE is extended through modules, and each module must implement the IModule interface. Besides, the open architecture depicted in Figure 11.2 shows how the PIPE-data layer and PeabraiN-data layer are

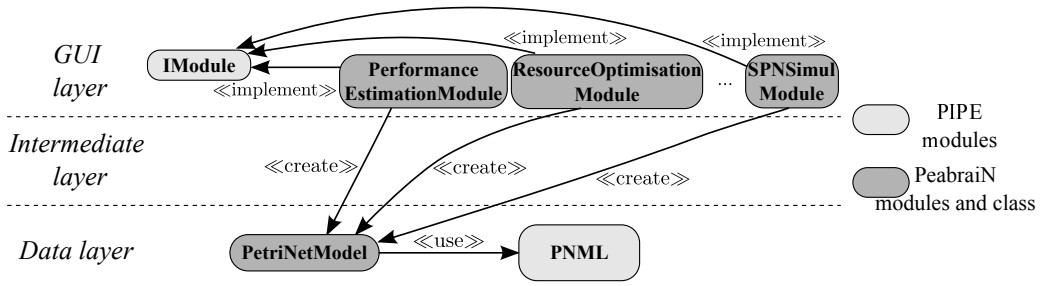


Figure 11.2: Integration of PeabraiN in the PIPE tool.

related. Each PeabraiN module creates a matrix representation of the current PN model, which is in PNML (PIPE format). We do not use PNML in our data layer because the algorithms work with the matrix representation.

Each PeabraiN module in the GUI layer in Figure 11.2 will create an instance of the class with its same name in Figure 11.1. For example, PerformanceEstimationModule creates PerformanceEstimationGUI to allow the user to introduce ϵ (input parameter in algorithm in Figure 2). Figure 11.3 illustrates the interactions between the user, PIPE, and PeabraiN when executing.

In brief, the new modules added to PIPE are:

- **Performance Estimation.** It needs as input the degree of precision, ϵ, to be achieved. Note that the lower the value of ϵ, the longer it takes to finish. The module reports about the components of the p-semiflow in each iteration step and its throughput, computed by simulation.

- **Resource Optimisation.** This feature enacts an optimal distribution of resources in a shared-resource PN for a given budget and resource costs, trying to optimise the system performance. It needs the process-idle place, that is, the place which represents the workload in the system (i.e., incoming customers or requests), the maximum of budget to be spent and the cost of each resource. Once the input data are validated, it computes and reports about the needed increment of resources, the rest of budget to be assigned and informs if there is more choice of improvement.

- **Linear Bound.** This feature allows to compute the upper and lower performance bound for a given transition and the slowest p-semiflow of the PN.

- **Structural Enabling.** This feature allows the computation of the structural enabling bound for a given transition or for all the transitions.

- **Structural Marking.** This feature allows the computation of the structural marking bound for a given place or for all the places.

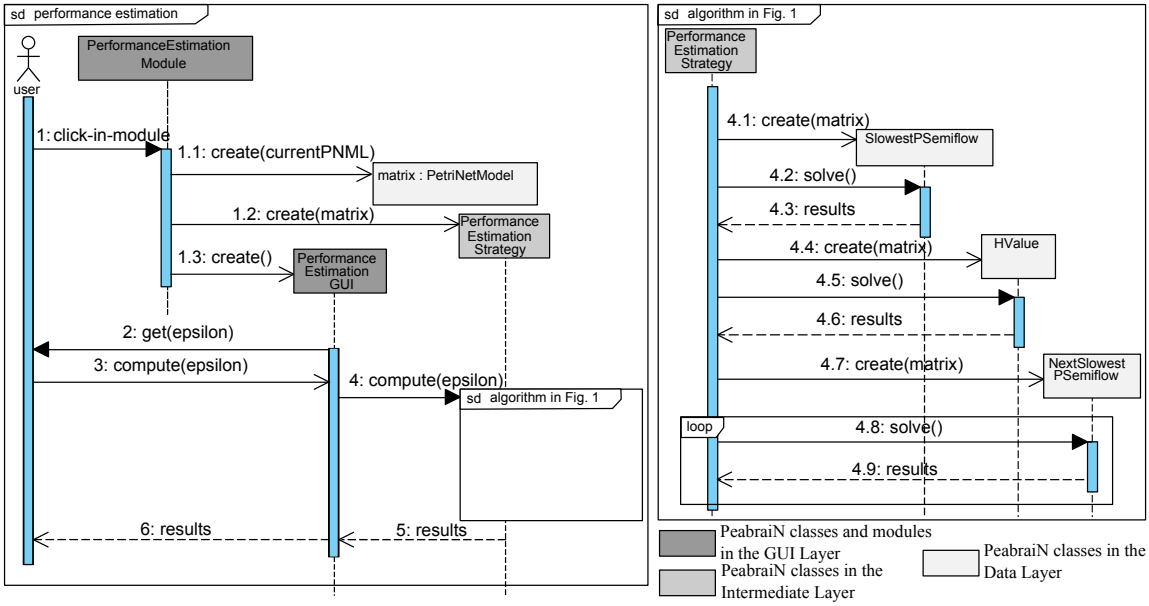


Figure 11.3: UML Sequence Diagram for executing performance estimation module.

- **Visit Ratios Computation.** It needs a transition, used as the normalised transition, for the visit ratios computation. As well as the result, information about the uniqueness of its solution is also given.
- **SPN Simulation Analysis.** Input data are either the maximum simulation time, or the confidence level and error accuracy to be achieved. When the simulation finishes, the module informs about the estimated throughput (computed for transition t_0 by default) of the PN, the confidence level, the error accuracy achieved and the execution time.

11.2.3 Example of Use

Let us illustrate the use of PeabraiN by an example. Recall the Petri net representation of a packet-routing algorithm depicted in Figure 4.5 and used as a running example in Section 4.2.

Let us suppose an expected number of packets $nP = 10$, an initial budget of $\$80,000$, the cost of creating new threads is $\$6,000$, due to reimplementation and deployment issues, and the implementation and reconfiguration of new filtering-threads has a cost of \$250. With this configuration, the resource optimisation procedure gives as result that 3 new threads should be created in order to attend such an incoming number of packets and hence to maximise the performance. Figure 11.4 depicts a snapshot of the results as reported by PeabraiN. It reports about the remaining money to be spent, the number of instances

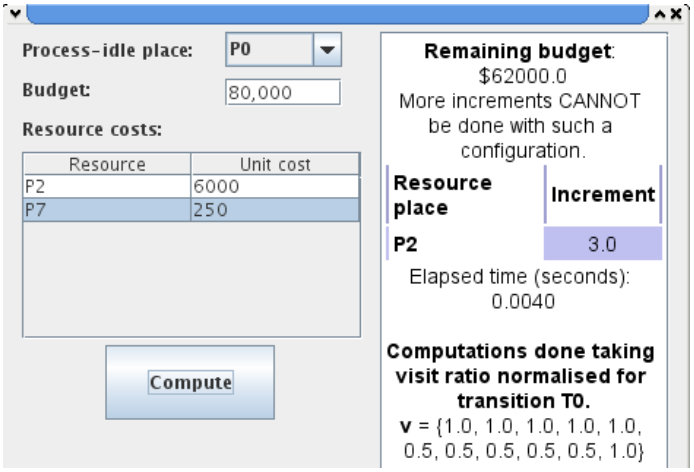


Figure 11.4: PeabraiN: Snapshot of execution results (resource optimisation).

of resources to be incremented and the elapsed time in computation. Besides, for this configuration it reports that no more further improvement can be done with the remaining budget. This means that even if we keep incrementing resources, the bottleneck of the system is the number of packets. Therefore, if resources are incremented they will be idle.

11.2.4 Tool Availability and Installation Requirements

PeabraiN has been developed with the Eclipse IDE under Linux environment, and successfully tested on Linux and Windows environments. PeabraiN needs to have installed in the host machine the following software to execute:

- a JRE version 1.6 (or higher); and
- an LP solver, namely the GNU Linear Programming Kit (GLPK) and its associated library for binding with Java.

Currently, we are working on an automatic detection of installed LP solvers in the host machine, and then automatically configure PeabraiN to work with them. Even PeabraiN is designed for working with several LP solvers, such as CPLEX or lpsolve among others, our initial thought was GLPK as it is free software under GNU General Public License (GNU GPL).

There exists a web page:

http://webdiis.unizar.es/GISED/?q=tool/peabrain

where further information about tool requirements and installation steps, tool binaries and sources can be found. PeabraiN is released under GNU GPL version 3 license.

11.3 Concluding Remarks

We have developed PeabraiN, a collection of PIPE tool-compliant modules for performance estimation and resource optimisation based on bounds computation for Stochastic Petri Nets. Moreover, other features have been added to PIPE, such as structural enabling bound at a transition, structural marking bound at a place, visit ratios computation or SPN simulation analysis.

As future work, we plan to add the choice of LP solver by the user, the automatic detection of installed LP solver for automatic configuration of the tool, and to allow to change the simulation parameters, such as the transient observations to be discarded. Besides, we intend to perform more experiments with larger benchmarks to show its applicability, and to compare the performance of our tool with other broadly used tools, such as GreatSPN or TimeNET.

Part V

Conclusions

Chapter 12

Conclusions and Open Problems

If you think education is expensive,
try ignorance.
(Derek Bok)

This chapter points out the main statements and contributions of this dissertation and opens a discussion about future work and further research lines.

12.1 Thesis Summary

In this work we have addressed the problems of designing and analysing in critical systems that are required to fulfil their mission despite the presence of security issues. These systems can be naturally modelled as Discrete Event Systems (DES) where resources are shared, also called Resource Allocation Systems (RAS). In this dissertation, we have focused on FT systems using shared resources modelled as Petri nets (PNs) as formal model language – more precisely, as process Petri nets. The security issues that can affect this kind of systems may even inevitably consume their resources hence system performance can be influenced, even affecting its full operability.

This dissertation has stepped forward in such a security-performance trade-off. In the first part we have proposed a set of models that allow to bring security into foreground while designing. In Chapter 3 we have proposed a Unified Modelling Language (UML) profile, called `SecAM` (stands for Security Analysis and Modelling), which enhances UML modelling expressiveness by providing security-related concepts. In Chapter 4, we have proposed Fault-Tolerant Techniques (FTTs) models that can be easily added to a system design. The last chapter of the first part, Chapter 5, introduces a model-based methodology for analysis of security-performance trade-off of critical systems that combine FTTs, such as recovery procedures, and/or Security Mechanisms (SMs), such as encryption of data, in order to react to intrusions.

These systems are usually so large that makes the exact computation of their performance a highly complex computational task, due to the well-known state explosion problem. As a result, a task that requires an exhaustive state space exploration becomes unachievable in reasonable time for large systems.

The second part of this work is devoted to performance (also performability – i.e., performance under failure conditions) and resource optimisation analysis in FT systems. Chapter 6 provided some strategies for the upper throughput bound computation on Petri nets thus avoiding the state explosion problem. Recall that we are dealing with critical systems that incorporate FT techniques to deal with any unexpected situations and these additions may have an impact on system performance. In Chapter 7 we have provided a set of algorithms to compensate the throughput degradation in critical systems.

The third part of this dissertation puts the aforementioned developed methods on practice. In Chapter 8 we have introduced a case study where the approaches presented in Chapters 6 and 7 are tested. The system under study is a Secure Database System (SDBS) deployed as a Web Service which stores sensible information. The last chapter of this part, Chapter 10, is devoted to the application of performance analysis methods applied to a more specific domain, namely, scientific workflows.

The last part of this dissertation introduces `PeabraiN`, which stands for *"Performance Estimation bAsed (on) Bounds (and) Resource optimisAtIon (for Petri) Nets "*. `PeabraiN` is a PIPE tool extension that implements some of the methods presented in Chapters 6 and 7.

12.2 Main Contributions

This section summarises the main contributions that we have identified:

- *Specification of security as a Non-Functional Property into UML designs.* The UML profile that we have developed, called `SecAM`, presents a powerful UML framework for the specification and analysis of security. `SecAM` enables to consider security in the early stages of the software and systems life-cycle, thus promoting the analysis of system security before the deployment stage.

- *UML models representing common Fault-Tolerant (FT) techniques.* We have proposed some models that represent common FT techniques. The rationale behind these models is to combine them with software behavioural designs for dependability assessment. The easy integration of the proposed FT techniques (FTTs) into software designs may allow to test different techniques for the same design to find the ones fitting better.

- *Quantitative estimation of performance-security trade-off in critical systems.* We have provided a model-based methodology able to quantitatively estimate the system performance while introducing FTTs and/or SMs aimed at dealing with 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.

- *Accurate estimation of the steady-state throughput by upper throughput bounds.* We have proposed two methods that can be applied to Stochastic Marked Graphs and Process Petri nets to compute more accurate upper throughput bounds than the ones that can be computed with other methods. Both methods are based on iterative algorithms that make use of linear programming techniques, thus exhibiting a good trade-off between accuracy and efficiency.

- *Compensation of throughput degradation in FT systems.* We have provided a strategy that, for a given initial budget and cost of each resource, gauges the number of instances of each resource so that the system performance is maximised and the budget is not exceeded. We have also presented an iterative algorithm that computes the initial marking needed to maintain a given upper throughput bound in a FT system model and we have presented an Integer-Linear Programming Problem (ILPP) that minimises the cost of compensating throughput degradation caused by the presence of faults and errors.

- *Quantitative metric for evaluating optimisation on scientific workflows.* We have proposed a quantitative metric based on the workflow structure, and on performance information derived from past historical executions that is intended for use at the design-stage. This metric enables to compare different workflow definitions and evaluate their potential for optimisation, focusing on network bandwidth and buffer usage.

- *Data-throttling strategy for improving the use of bandwidth and input buffers in scientific workflow tasks.* In the context of scientific workflows, we have provided a strategy for improving the use of bandwidth and input buffers, so that tasks have all their inputs arriving at the same time (then, the buffer usage is minimised). We have done this improvement by modifying data transmission speed (either reduced or increased), making as a side product a better usage of bandwidth.

- *A tool for performance estimation and resource optimisation in Petri nets.* We have developed a PIPE extension, called `PeabraiN`, which provides a collection of PIPE tool-compliant modules for performance estimation and resource optimisation based on bounds computation for Stochastic Petri Nets. Indeed, `PeabraiN` provides a powerful framework to develop any PIPE tool-compliant module using Linear Programming (LP) techniques for computation in a (very) easy way.

12.3 Future Work and Open Problems

Despite the contributions of this dissertation, there are still several open problems that are planned as future work, such as:

- *Addition of UML* `SecAM` *profile to existing UML case tools.* Although `SecAM` has been integrated into Eclipse through a plug-in, it is still in an early development phase (i.e., it is not fully operative) and not yet released. We plan to extend this Eclipse plug-in doing it fully operative and also to consider other UML case tools, such as ArgoUML or MagicDraw.

- *Usage of UML* `SecAM` *profile for model-checking.* We aim at extending the transformation from UML-profiled diagrams to other interesting dependability/security analysis models, such as Fault Trees or Bayesian Networks. We also plan to define model-to-model (M2M) transformations to automatically compute SecAM derived tagged-values, to verify SecAM UML-profiled model for consistency (i.e., using OCL constraints) as well as to compute vulnerability related metrics.

- *Development and extension of a UML Fault-Tolerant (FT) modelling library.* We plan to provide guidelines to software designers about the best choices of Fault-Tolerant techniques and security mechanisms for the attacks systems may suffer by the usage of a plug-in for common UML design tools, such as ArgoUML, Visual Paradigm or MagicDraw, that incorporates the UML FT models introduced in Chapter 4. We also aim at extending the UML FT modelling library by incorporating other FT techniques, such as *Safe State* or *Software Exception Handlers*.

- *Analysis of the trade-offs concerning security and other non-functional properties.* In this dissertation we have considered on the security-performance trade-off. However, we consider this as a starting point for investigating even more sophisticated trade-offs. For instance, it would be relevant to study the trade-off between security and 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.

- *Quality of upper throughput bounds and extension to more general Petri net classes.* As future work, we plan to research into the quality of the initial upper bound obtained with the strategies presented in Chapter 6 and to extend both strategies to more general Petri net classes.

- *Deeper analysis of the relationship between throughput and marking in FT systems.* We aim at researching the dependence between throughput and other PN-related parameters, such as initial marking or transition timings, in order to provide an algorithm to compensate throughput degradation in systems modelled with a more general class of PNs rather than Process PNs.

- *Development of a tool for workflow design.* We consider to design and implement a tool for workflow design that assists users in their workflow configurations, and helps them determine: i) comparison of different workflows configurations for potential optimisation, ii) sub-optimal throttling values, iii) performance speed-up or degradation when applying data throttling, and iv) buffer storage needs.

- *Deployment of data-throttling strategy in more realistic environments.* We plan to deploy the approach for computing the data-throttling values in scientific workflows presented in Chapter 10 in real scenarios. For instance, we aim at incorporating our approach to Pegasus workflow system to see how it works in industrial scenarios.

- *Improvement of* `PeabraiN` *tool.* As future work, we aimed at extending `PeabraiN` in several ways: firstly, to provide the user the chance of selecting a LP solver for computation. Then, we also plan to develop an approximate stochastic simulation algorithm and lastly, a web service such that `PeabraiN` can be invoked over the Internet.

Besides all the aforementioned future work, we aim at applying all the findings of this dissertation to some other domains, such as Fault-Tolerant in manufacturing systems or scientific workflows.

Relevant Publications Related to this Dissertation

[Rodríguez and Júlvez, 2010] Rodríguez, R. J. and Júlvez, J. (2010). Accurate Performance Estimation for Stochastic Marked Graphs by Bottleneck Regrowing. In *Proceedings of the 7th European Performance Engineering Workshop (EPEW)*, volume 6342 of *LNCS*, pages 175–190. Springer.

[Rodríguez and Merseguer, 2010] Rodríguez, R. J. and Merseguer, J. (2010). Integrating Fault-Tolerant Techniques into the Design of Critical Systems. In Giese, H., 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. Springer.

[Rodríguez et al., 2010] Rodríguez, R. J., Merseguer, J., and Bernardi, S. (2010). Modelling and Analysing Resilience as a Security Issue within UML. In *Proceedings of the 2nd International Workshop on Software Engineering for Resilient Systems (SERENE)*, pages 42–51, London, United Kingdom. ACM.

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

[Rodríguez et al., 2012a] Rodríguez, R. J., Júlvez, J., and Merseguer, J. (2012a). PeabraiN: A PIPE Extension for Performance Estimation and Resource Optimisation. In *Proceedings of the 12th International Conference on Application of Concurrency to System Designs (ACSD)*, pages 142–147. IEEE.

[Rodríguez et al., 2012b] Rodríguez, R. J., Tolosana-Calasanz, R., and Rana, O. F. (2012b). Automating Data-Throttling Analysis for Data-Intensive Workflows. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 310–317. IEEE.

[Rodríguez et al., 2012c] Rodríguez, R. J., Tolosana-Calasanz, R., and Rana, O. F. (2012c). Measuring the Effectiveness of Thottled Data Transfers on Data-Intensive Workflows. In Jezic, G., Kusek, M., Nguyen, N. T., Howlett, R. J., and Jain, L. C., editors, *Proceedings of the 6th International KES Conference on Agents and Multi-agent Systems – Technologies and Applications*, volume 7327 of *Lecture Notes in Computer Science*, pages 144–153. Springer.

[Rodríguez et al., 2012d] Rodríguez, R. J., Trubiani, C., and Merseguer, J. (2012d). Fault-Tolerant Techniques and Security Mechanisms for Model-based Performance Prediction of Critical Systems. In *Proceedings of the 3rd International Symposium on Architecting Critical Systems (ISARCS)*, pages 21–30. ACM.

[Rodríguez et al., 2013] Rodríguez, R. J., Tolosana-Calasanz, R., and Rana, O. F. (2013). Data-Throttling Analysis for Data-Intensive Workflows on Dynamic Environments. Technical report, Universidad de Zaragoza. To be submitted to IEEE Transactions on Parallel and Distributed Systems.

[Rodríguez et al., 2013a] Rodríguez, R. J., Júlvez, J., and Merseguer, J. (2013a). On the Performance Estimation and Resource Optimisation in Process Petri Nets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, PP(99):1–14. doi: 10.1109/TSMC.2013.2245118

[Rodríguez et al., 2013b] Rodríguez, R. J., Júlvez, J., and Merseguer, J. (2013b). Quantification and Compensation of the Impact of Faults in System Throughput. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*. Accepted for publication.

Bibliography

[Aalst et al., 2002] Aalst, W. M. P. v. d., Hirnschall, A., and Verbeek, H. M. W. E. (2002). An Alternative Way to Analyze Workflow Graphs. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, CAiSE '02, pages 535–552, London, UK, UK. Springer-Verlag.

[Abdelmoez et al., 2004] Abdelmoez, W., Nassar, D., Shereshevsky, M., Gradetsky, N., Gunnalan, R., Ammar, H., Yu, B., and Mili, A. (2004). Error Propagation in Software Architectures. In *Proceedings of the 10th International Symposium on Software Metrics (ISSME)*, pages 384–393.

[Abi-Antoun and Barnes, 2010] Abi-Antoun, M. and Barnes, J. M. (2010). Analyzing Security Architectures. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, pages 3–12, New York, NY, USA. ACM.

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

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

[Aversano et al., 2002] Aversano, L., Cimitile, A., Gallucci, P., and Villani, M. (2002). FlowManager: a workflow management system based on Petri nets. In *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, pages 1054–1059.

[Avizienis, 1997] Avizienis, A. (1997). Toward Systematic Design of Fault-Tolerant Systems. *Computer*, 30(4):51–58.

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

[Baarir et al., 2009] Baarir, S., Beccuti, M., Cerotti, D., De Pierro, M., Donatelli, S., and Franceschinis, G. (2009). The GreatSPN tool: recent enhancements. *SIGMETRICS Perform. Eval. Rev.*, 36(4):4–9.

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

[Barnum, 2008] Barnum, S. (2008). Common Attack Pattern Enumeration and Classification (CAPEC) Schema Description. The MITRE Corporation.

[Barnum and McGraw, 2005] Barnum, S. and McGraw, G. (2005). Knowledge for Software Security. *IEEE Security and Privacy*, 3(2):74–78.

[Bernardi et al., 2001] Bernardi, S., Donatelli, S., and Horváth, A. (2001). Implementing Compositionality for Stochastic Petri Nets. *Journal of Software Tools for Technology Transfer*, 3:417–430.

[Bernardi et al., 2011] Bernardi, S., Merseguer, J., and Petriu, D. (2011). A Dependability Profile within MARTE. *Journal of Software and Systems Modeling*, 10(3):313–336.

[Berriman et al., 2007] Berriman, G. B., Deelman, E., Good, J., Jacob, J. C., Katz, D. S., Laity, A. C., Prince, T. A., Singh, G., and Su, M.-H. (2007). Generating Complex Astronomy Workflows. In Taylor, I. J., Deelman, E., Gannon, D. B., and Shields, M., editors, *Workflows for e-Science*, pages 19–38. Springer London.

[Bertino and Crampton, 2007] Bertino, E. and Crampton, J. (2007). Security for Distributed Systems: Foundations of Access Control. In Qian, Y., Tipper, D., Krishnamurthy, P., and Joshi, J., editors, *Information Assurance: Survivability and Security in Networked Systems*, pages 39–80. Morgan Kaufman.

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

[Bobbio, 1989] Bobbio, A. (1989). Petri Nets Generating Markov Reward Models for Performance/Reliability Analysis of Degradable Systems. In Potier, D. and Puigjaner, B., editors, *Proceedings of the Fourth International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, pages 353–365, New York, NY, USA. Plenum.

[Bondavalli et al., 2001] Bondavalli, A., Dal Cin, M., Latella, D., Majzik, I., Pataricza, A., and Savoia, G. (2001). Dependability Analysis in the Early Phases of UML Based System Design. *Journal of Computer Systems Science and Engineering*, 16(5):265–275.

[Bonet et al., 2007] Bonet, P., Llado, C., Puijaner, R., and Knottenbelt, W. (2007). PIPE v2.5: A Petri Net Tool for Performance Modelling. In *Proceedings of the 23rd Latin American Conference on Informatics (CLEI)*, Costa Rica.

[Braga, 2011] Braga, C. (2011). A transformation contract to generate aspects from access control policies. *Software and Systems Modeling*, 10:395–409.

[Braynov, 2003] Braynov, S. (2003). On Future Avenues for Distributed Attacks. In Hutchinson, B., editor, *2nd European Conference on Information Warfare and Security (ECIW)*, pages 51–60, University of Reading, United Kingdom.

[Brglez et al., 1989] Brglez, F., Bryan, D., and Kozminski, K. (1989). Combinational Profiles of Sequential Benchmark Circuits. *IEEE International Symposium on Circuits and Systems*, 3:1929–1934.

[Campos et al., 1992] Campos, J., Chiola, G., Colom, J., and Silva, M. (1992). Properties and Performance Bounds for Timed Marked Graphs. *IEEE T. Circuits-I.*, 39(5):386–401.

[Campos and Silva, 1992] Campos, J. and Silva, M. (1992). Structural Techniques and Performance Bounds of Stochastic Petri Net Models. *Lecture Notes in Computer Science*, 609:352–391.

[Campos and Silva, 1993] Campos, J. and Silva, M. (1993). Embedded Product-Form Queueing Networks and the Improvement of Performance Bounds for Petri Net Systems. *Performance Evaluation*, 18(1):3–19.

[Canetti et al., 1997a] Canetti, R., Gennaro, R., Herzberg, A., and Naor, D. (1997a). Proactive Security: Long-term Protection Against Break-ins. *CryptoBytes*, 3(1):1–8.

[Canetti et al., 1997b] Canetti, R., Halevi, S., and Herzberg, A. (1997b). Maintaining Authenticated Communication in the Presence of Break-ins. In *Proceedings of the 16th annual ACM symposium on Principles Of Distributed Computing (PODC)*, pages 15–24, New York, NY, USA. ACM.

[Carmona et al., 2009] Carmona, J., Júlvez, J., Cortadella, J., and Kishinevsky, M. (2009). Scheduling Synchronous Elastic Designs. In *Proceedings of the 2009 Application of Concurrency to System Design conference (ACSD)*, Augsburg, Germany.

[Casale et al., 2008] Casale, G., Mi, N., and Smirni, E. (2008). Bound Analysis of Closed Queueing Networks with Workload Burstiness. *SIGMETRICS Perform. Eval. Rev.*, 36:13–24.

[Casanova et al., 2008] Casanova, H., Legrand, A., and Quinson, M. (2008). SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *10th IEEE International Conference on Computer Modeling and Simulation*.

[Chen et al., 2008] Chen, Y.-L., Hsu, P.-Y., and Chang, Y.-B. (2008). A Petri Net Approach to Support Resource Assignment in Project Management. *IEEE T. Syst. Man. Cy. A.*, 38(3):564–574.

[Cheswick et al., 2003] Cheswick, W. R., Bellovin, S. M., and Rubin, A. D. (2003). *Firewalls and Internet Security; Repelling the Wily Hacker*. Addison-Wesley, Reading, MA, second edition.

[Chiola et al., 1993] Chiola, G., Anglano, C., Campos, J., Colom, J., and Silva, M. (1993). Operational Analysis of Timed Petri Nets and Application to the Computation of Performance Bounds. In *Proceedings of the 5th International Workshop on Petri Nets and Performance Models (PNPM)*, pages 128–137, Toulouse, France. IEEE Computer Society Press.

[Cilardo et al., 2007] Cilardo, A., Coppolino, L., Mazzeo, A., and Romano, L. (2007). 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. IEEE Computer Society.

[Cirit and Buzluca, 2009] Cirit, c. and Buzluca, F. (2009). A UML Profile for Role-Based Access Control. In *Proceedings of the 2nd international conference on Security of Information and Networks (SIN)*, pages 83–92, New York, NY, USA. ACM.

[Colom, 2003] Colom, J. (2003). The Resource Allocation Problem in Flexible Manufacturing Systems. In van der Aalst, W. and Best, E., editors, *Applications and Theory of Petri Nets*, volume 2679 of *LNCS*, pages 23–35. Springer Berlin / Heidelberg.

[Colom et al., 1990] Colom, J., Campos, J., and Silva, M. (1990). On Liveness Analysis through Linear Algebraic Techniques. In *Proceedings of the Annual General Meeting of ESPRIT Basic Research Action 3148 Design Methods Based on Nets (DEMON)*, Paris, France.

[Cormen et al., 2001] Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. (2001). *Introduction to Algorithms*. McGraw-Hill Higher Education.

[Cortellessa and Grassi, 2007] Cortellessa, V. and Grassi, V. (2007). A Modeling Approach to Analyze the Impact of Error Propagation on Reliability of Component-Based Systems. In Schmidt, H., Crnkovic, I., Heineman, G., and Stafford, J., editors, *Proceedings of the 10th International Conference on Component-Based Software Engineering*, volume 4608 of *Lecture Notes in Computer Science*, pages 140–156, Berlin, Heidelberg. Springer Berlin / Heidelberg.

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

[Cortellessa et al., 2010a] Cortellessa, V., Trubiani, C., Mostarda, L., and Dulay, N. (2010a). An Architectural Framework for Analyzing Tradeoffs between Software Security and Performance. In Giese, H., 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.

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

[Deelman et al., 2007] Deelman, E., Mehta, G., Singh, G., Su, M., and Vahi, K. (2007). *Workflows for eScience*, chapter Pegasus: Mapping Large-Scale Workflows to Distributed Resources, pages 376–394. Springer.

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

[Devanbu and Stubblebine, 2000] Devanbu, P. T. and Stubblebine, S. (2000). Software Engineering for Security: a Roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 227–239, New York, NY, USA. ACM.

[Dierks and Rescorla, 2006] Dierks, T. and Rescorla, E. (2006). The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, Internet Engineering Task Force (IETF).

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

[Donatelli and Franceschinis, 1996] Donatelli, S. and Franceschinis, G. (1996). The PSR Methodology: Integrating Hardware and Software Models. In *Proceedings of the 17th International Conference of Application and Theory of Petri Nets (ICATPN)*, pages 133–152.

[Dworkin, 2001] Dworkin, M. (2001). Recommendation for Block Cipher Modes of Operation: Methods and Techniques. Technical report, The National Institute of Standards and Technology (NIST). Special Publication 800-38A.

[Ezpeleta and Valk, 2006] Ezpeleta, J. and Valk, R. (2006). A Polynomial Deadlock Avoidance Method for a Class of Nonsequential Resource Allocation Systems. *IEEE T. Syst. Man. Cy. A.*, 36(6):1234–1243.

[Fernández, 2004] Fernández, E. B. (2004). A Methodology for Secure Software Design. In Arabnia, H. R. and Reza, H., editors, *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, volume 1, pages 130–136, Las Vegas, Nevada, USA. CSREA Press.

[Florin and Natkin, 1985] Florin, G. and Natkin, S. (1985). Les réseaux de Petri stochastiques. *Technique et Science Informatique*, 4:143–160.

[Florin and Natkin, 1989] Florin, G. and Natkin, S. (1989). Necessary and Sufficient Ergodicity Condition for Open Synchronized Queueing Networks. *IEEE T. Software. Eng.*, 15(4):367–380.

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

[Georg et al., 2010] Georg, G., Anastasakis, K., Bordbar, B., Houmb, S. H., Ray, I., and Toahchoodee, M. (2010). Verification and Trade-Off Analysis of Security Properties in UML System Models. *IEEE Transactions on Software Engineering*, 36(3):338–356.

[Gillespie, 1976] Gillespie, D. T. (1976). A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. *Journal of Computational Physics*, 22(4):403–434.

[Gokhale et al., 2004] Gokhale, S. S., Wong, W. E., Horgan, J., and Trivedi, K. S. (2004). An analytical approach to architecture-based software performance and reliability prediction. *Performance Evaluation*, 58(4):391–412.

[Goldratt and Cox, 1986] Goldratt, E. M. and Cox, J. (1986). *The Goal: A Process of Ongoing Improvement*. North River Press.

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

[Goŝeva-Popstojanova and Trivedi, 2001] Goŝeva-Popstojanova, K. and Trivedi, K. S. (2001). Architecture-based approach to reliability assessment of software systems. *Performance Evaluation*, 45(2–3):179–204.

[Goudalo and Seret, 2008] Goudalo, W. and Seret, D. (2008). Toward the Engineering of Security of Information Systems (ESIS): UML and the IS Confidentiality. In *Proceedings of the 2d International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)*, pages 248–256.

[Guan et al., 2006] Guan, Z., Hernandez, F., Bangalore, P., Gray, J., Skjellum, A., Velusamy, V., and Liu, Y. (2006). Grid-Flow: A Grid-Enabled Scientific Workflow System with a Petri-Net-Based Interface: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18:1115–1140.

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

[Haddad et al., 2005] Haddad, S., Moreaux, P., Sereno, M., and Silva, M. (2005). Product-Form and Stochastic Petri Nets: a structural approach. *Perform. Eval.*, 59:313–336.

[Haley et al., 2006] Haley, C. B., Moffett, J. D., Laney, R., and Nuseibeh, B. (2006). A Framework for Security Requirements Engineering. In *Proceedings of the 2006 International Workshop on Software Engineering for Secure Systems*, SESS '06, pages 35–42, New York, NY, USA. ACM.

[Halkidis et al., 2008] Halkidis, S. T., Tsantalis, N., Chatzigeorgiou, A., and Stephanides, G. (2008). Architectural Risk Analysis of Software Systems Based on Security Patterns. *IEEE Transactions on Dependable and Secure Computing*, 5(3):129–142.

[Hansman and Hunt, 2005] Hansman, S. and Hunt, R. (2005). A taxonomy of network and computer attacks. *Computers & Security*, 24(1):31–43.

[Harrison and Avgeriou, 2008] Harrison, N. B. and Avgeriou, P. (2008). Incorporating Fault Tolerance Tactics in Software Architecture Patterns. In *Proceedings of the 2008 RISE/EFTS Joint Int. Workshop on Software Engineering for Resilient Systems (SERENE)*, pages 9–18. ACM.

[Heckerman, 1995] Heckerman, D. (1995). A Bayesian Approach to Learning Causal Networks. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, UAI'95, pages 285–295, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Hee et al., 2001] Hee, K. V., Reijers, H., Verbeek, E., and Zerguini, L. (2001). On the Optimal Allocation of Resources In Stochastic Workflow Nets. In Djemame, K. and Kara, M., editors, *Proceedings of the 7th UK Performance Engineering Workshop*, pages 23–34, University of Leeds, Leeds, UK.

[Heyman et al., 2011] Heyman, T., Yskout, K., Scandariato, R., Schmidt, H., and Yu, Y. (2011). The Security Twin Peaks. In Erlingsson, U., Wieringa, R., and Zannone, N., editors, *Engineering Secure Software and Systems*, volume 6542 of *Lecture Notes in Computer Science*, pages 167–180. Springer Berlin / Heidelberg.

[Hillah et al., 2009] Hillah, L. M., Kindler, E., Kordon, F., Petrucci, L., and Tréves, N. (2009). A primer on the Petri Net Markup Language and ISO/IEC 15909-2. *Petri Net Newsletter*, 76:9–28.

[HoneyNet Project, 2004] HoneyNet Project, T., editor (2004). *Know Your Enemy: Learning about Security Threats*. Addison Wesley Publishing, 2nd edition.

[Horvath and Dörges, 2008] Horvath, V. and Dörges, T. (2008). From Security Patterns to Implementation Using Petri Nets. In *Proceedings of the 4th International Workshop on Software Engineering for Secure Systems (SESS)*, SESS '08, pages 17–24, New York, NY, USA. ACM.

[Houmb and Hansen, 2003] Houmb, S. H. and Hansen, K. K. (2003). Towards a UML profile for Security Assessment. In *Proceedings of the Workshop on Critical Systems Development with UML (UML)*.

[Hu et al., 2012] Hu, H., Zhou, M., and Li, Z. (2012). Liveness and Ratio-Enforcing Supervision of Automated Manufacturing Systems Using Petri Nets. *IEEE T. Syst. Man. Cy. A.*, 42(2):392–403.

[Hussain et al., 2003] Hussain, A., Heidemann, J., and Papadopoulos, C. (2003). A Framework for Classifying Denial of Service Attacks-extended. Technical Report ISI-TR-2003-569b, USC/Information Sciences Institute. (Original TR, February 2003, updated June 2003).

[Hussein and Zulkernine, 2006] Hussein, M. and Zulkernine, M. (2006). UMLintr: A UML Profile for Specifying Intrusions. In *Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS)*, pages 279–288, Washington, DC, USA. IEEE Computer Society.

[Islam et al., 2011] Islam, S., Mouratidis, H., and Jürjens, J. (2011). A Framework to Support Alignment of Secure Software engineering with Legal Regulations. *Software and Systems Modeling (SoSym)*, 10(3):369–394.

[Jensen, 1997] Jensen, K. (1997). *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Monographs in Theoretical Computer Science. Springer, 2nd edition.

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

[Jürjens, 2002] Jürjens, J. (2002). UMLsec: Extending UML for Secure Systems Development. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 412–425, London, UK. Springer-Verlag.

[Kalan et al., 2008] Kalan, A. A. E., Baina, A., Beitollahi, H., Bessani, A., Bondavalli, A., Correia, M., Daidone, A., Deconinck, G., Deswarte, Y., Garrone, F., Grandoni, F., Moniz, H., Neves, N., Rigole, T., Sousa, P., and Verissimo, P. (2008). D10: Preliminary Specification of Services and Protocols. Project deliverable, CRUTIAL: Critical Utility Infrastructural Resilience.

[Kant et al., 2000] Kant, K., Iyer, R., and Mohapatra, P. (2000). 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. IEEE Computer Society.

[Kellerer et al., 2004] Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Knapsack Problems*. Springer-Verlag.

[Khan and Zulkernine, 2008] Khan, M. U. A. and Zulkernine, M. (2008). Quantifying Security in Secure Software Development Phases. *Computer Software and Applications Conference, Annual International*, 0:955–960.

[Khan, 2011] Khan, R. (2011). Secure software development: a prescriptive framework. *Computer Fraud & Security*, 2011(8):12–20.

[Lagarde et al., 2007] Lagarde, F., Espinoza, H., Terrier, F., and Gérard, S. (2007). 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. ACM.

[Li et al., 2004] Li, J., Fan, Y., and Zhou, M. (2004). Performance Modeling and Analysis of Workflow. *IEEE T. Syst. Man. Cy. A.*, 34(2):229–242.

[Li et al., 2012] Li, Z., Wu, N., and Zhou, M. (2012). Deadlock Control of Automated Manufacturing Systems Based on Petri Nets – A Literature Review. *IEEE T. Syst. Man. Cy. C.*, 42(4):437–462.

[Little, 1961] Little, J. D. C. (1961). A Proof for the Queuing Formula: L= λ W. *Operations Research*, 9(3):383–387.

[Liu, 1995] Liu, Z. (1995). Performance Bounds for Stochastic Timed Petri Nets. In *Proceedings of the 16th ICATPN*, pages 316–334. Springer-Verlag.

[Lodderstedt et al., 2002] Lodderstedt, T., Basin, D. A., and Doser, J. (2002). SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 426–441, London, UK. Springer-Verlag.

[Lopez-Grao and Colom, 2011] Lopez-Grao, J. and Colom, J. (2011). On the Deadlock Analysis of Multithreaded Control Software. In *IEEE 16th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8.

[López-Grao et al., 2004] López-Grao, J. P., Merseguer, J., and Campos, J. (2004). From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering. In *Proceedings of the 4th International Workshop on Software and Performance (WOSP)*, pages 25–36, New York, NY, USA. ACM.

[Majzik et al., 2003] Majzik, I., Pataricza, A., and Bondavalli, A. (2003). Stochastic Dependability Analysis of System Architecture based on UML Models. In De Lemos, R., Gacek, C., and Romanovsky, A., editors, *Proceedings of the Architecting Dependable Systems*, number 2677, pages 219–244. Springer.

[McGraw, 2004] McGraw, G. (2004). Software Security. *IEEE Security and Privacy*, 2(2):80–83.

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

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

[Menezes et al., 1996] Menezes, A. J., Vanstone, S. A., and Oorschot, P. C. V. (1996). *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.

[Merseguer et al., 2002] Merseguer, J., Campos, J., Bernardi, S., and Donatelli, S. (2002). A Compositional Semantics for UML State Machines Aimed at Performance Evaluation. In *Proceedings of the 6th International Workshop on Discrete Event Systems (WODES)*, WODES '02, pages 295–, Washington, DC, USA. IEEE Computer Society.

[Meyer, 1982] Meyer, J. F. (1982). Closed-Form Solutions of Performability. *IEEE Trans. Comput.*, 31(7):648–657.

[Microsoft, 2010] Microsoft (2010). Microsoft Security Development Lifecycle. `http://www.microsoft.com/security/sdl/default.aspx`. version 5.

[Molloy, 1982] Molloy, M. (1982). Performance Analysis Using Stochastic Petri Nets. *IEEE T. Comput.*, C-31(9):913–917.

[Mouratidis and Giorgini, 2008] Mouratidis, H. and Giorgini, P. (2008). Integrating Security and Software Engineering: An Introduction. In *Information Security and Ethics: Concepts, Methodologies, Tools, and Applications*, pages 200–210. IGI Global.

[Mouratidis et al., 2003] Mouratidis, H., Giorgini, P., and Manson, G. (2003). Integrating Security and Systems Engineering: Towards the Modelling of Secure Information Systems. In Eder, J. and Missikoff, M., editors, *Proceedings of the 15th Conference On Advanced Information Systems Engineering (CAiSE)*, volume 2681 of *Lecture Notes in Computer Science*, pages 63–78. Springer Berlin / Heidelberg.

[Murata, 1989] Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. In *Proceedings of the IEEE*, volume 77, pages 541–580.

[Nerieri et al., 2006] Nerieri, F., Prodan, R., Fahringer, T., and Truong, H.-L. (2006). Overhead Analysis of Grid Workflow Applications. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, GRID'06, pages 17–24, Washington, DC, USA. IEEE Computer Society.

[Nguyen-Tuong and Grimshaw, 1999] Nguyen-Tuong, A. and Grimshaw, A. S. (1999). Using Reflection for Incorporating Fault-Tolerance Techniques into Distributed Applications. *Parallel Processing Letters*, 9(2):291–301.

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

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

[OMG, 2010] OMG (2010). *Object Constraint Language (OCL)*. Object Management Group. v2.2, formal/2010-02-01.

[Osogami and Raymond, 2010] Osogami, T. and Raymond, R. (2010). Semidefinite Optimization for Transient Analysis of Queues. *SIGMETRICS Perform. Eval. Rev.*, 38:363–364.

[Ostrovsky and Yung, 1991] Ostrovsky, R. and Yung, M. (1991). How To Withstand Mobile Virus Attacks. In *Proceedings of the 10th annual ACM symposium on Principles Of Distributed Computing (PODC)*, pages 51–59, New York, NY, USA. ACM.

[Park and Humphrey, 2008] Park, S.-M. and Humphrey, M. (2008). Data Throttling for Data-Intensive Workflows. In *IEEE International Symposium on Parallel and Distributed Processing*, pages 1–11.

[Patzina et al., 2010] Patzina, L., Patzina, S., Piper, T., and Schürr, A. (2010). Monitor Petri Nets for Security Monitoring. In *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems*, S&D4RCES '10, pages 3:1–3:6, New York, NY, USA. ACM.

[Paxson, 2001] Paxson, V. (2001). An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks. *SIGCOMM Computer Communication Review*, 31:38–47.

[Pellegrini et al., 2008] Pellegrini, S., Hoheisel, A., Giacomini, F., and Ghiselli, A. (2008). Using GWorkflowDL for Middleware-Independent Modeling and Enactment of Workflows. In *Proceedings of the CoreGRID Integration Workshop 2008*, Crete, Greece.

[Pfleeger and Pfleeger, 2006] Pfleeger, C. P. and Pfleeger, S. L. (2006). *Security in Computing*. Prentice Hall, 4th edition.

[Ramamoorthy and Ho, 1980] Ramamoorthy, C. V. and Ho, G. S. (1980). Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets. *IEEE T. Software. Eng.*, 6(5):440–449.

[Ramchandani, 1974] Ramchandani, C. (1974). *Analysis of Asynchronous Concurrent Systems by Petri Nets*. PhD thesis, Dept. of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, MA, USA.

[Randimbivololona, 2001] Randimbivololona, F. (2001). Orientations in Verification Engineering of Avionics Software. In Wilhelm, R., editor, *Informatics*, volume 2000 of *Lecture Notes in Computer Science*, pages 131–137. Springer Berlin/Heidelberg.

[Reussner et al., 2003] Reussner, R. H., Schmidt, H. W., and Poernomo, I. H. (2003). Reliability prediction for component-based software architectures. *J. Syst. Softw.*, 66(3):241–252.

[Rodríguez et al., 2006] Rodríguez, A., Fernández-Medina, E., and Piattini, M. (2006). Security Requirement with a UML 2.0 Profile. In *Proceedings of the 1st International Conference on Availability, Reliability and Security (ARES)*, page 8 pp.

[Rodríguez and Júlvez, 2010] Rodríguez, R. J. and Júlvez, J. (2010). Accurate Performance Estimation for Stochastic Marked Graphs by Bottleneck Regrowing. In *Proceedings of the 7th European Performance Engineering Workshop (EPEW)*, volume 6342 of *LNCS*, pages 175–190. Springer.

[Rodríguez et al., 2012a] Rodríguez, R. J., Júlvez, J., and Merseguer, J. (2012a). PeabraiN: A PIPE Extension for Performance Estimation and Resource Optimisation. In *Proceedings of the 12th International Conference on Application of Concurrency to System Designs (ACSD)*, pages 142–147. IEEE.

[Rodríguez and Merseguer, 2010] Rodríguez, R. J. and Merseguer, J. (2010). Integrating Fault-Tolerant Techniques into the Design of Critical Systems. In Giese, H., 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. Springer.

[Rodríguez et al., 2010] Rodríguez, R. J., Merseguer, J., and Bernardi, S. (2010). Modelling and Analysing Resilience as a Security Issue within UML. In *Proceedings of the 2nd International Workshop on Software Engineering for Resilient Systems (SERENE)*, pages 42–51, London, United Kingdom. ACM.

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

[Rodríguez et al., 2012b] Rodríguez, R. J., Tolosana-Calasanz, R., and Rana, O. F. (2012b). Automating Data-Throttling Analysis for Data-Intensive Workflows. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 310–317. IEEE.

[Rodríguez et al., 2012c] Rodríguez, R. J., Tolosana-Calasanz, R., and Rana, O. F. (2012c). Measuring the Effectiveness of Thottled Data Transfers on Data-Intensive Workflows. In Jezic, G., Kusek, M., Nguyen, N. T., Howlett, R. J., and Jain, L. C., editors, *Proceedings of the 6th International KES Conference on Agents and Multi-agent Systems – Technologies and Applications*, volume 7327 of *Lecture Notes in Computer Science*, pages 144–153. Springer.

[Rodríguez et al., 2013] Rodríguez, R. J., Tolosana-Calasanz, R., and Rana, O. F. (2013). Data-Throttling Analysis for Data-Intensive Workflows on Dynamic Environments. Technical report, Universidad de Zaragoza. To be submitted to IEEE Transactions on Parallel and Distributed Systems.

[Rodríguez et al., 2012d] Rodríguez, R. J., Trubiani, C., and Merseguer, J. (2012d). Fault-Tolerant Techniques and Security Mechanisms for Model-based Performance Prediction of Critical Systems. In *Proceedings of the 3rd International Symposium on Architecting Critical Systems (ISARCS)*, pages 21–30. ACM.

[Rodríguez et al., 2013a] Rodríguez, R. J., Júlvez, J., and Merseguer, J. (2013a). On the Performance Estimation and Resource Optimisation in Process Petri Nets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, PP(99):1–14.

[Rodríguez et al., 2013b] Rodríguez, R. J., Júlvez, J., and Merseguer, J. (2013b). Quantification and Compensation of the Impact of Faults in System Throughput. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*. Accepted for publication.

[Rosado et al., 2010a] Rosado, D. G., Fernández-Medina, E., López, J., and Piattini, M. (2010a). Analysis of Secure Mobile Grid Systems: A systematic approach. *Information and Software Technology*, 52(5):517–536.

[Rosado et al., 2010b] Rosado, D. G., Fernández-Medina, E., López, J., and Piattini, M. (2010b). Developing a Secure Mobile Grid System through a UML Extension. *J. UCS*, 16(17):2333–2352.

[Ross, 1983] Ross, S. (1983). *Stochastic Processes*. Wiley series in mathematical statistics. Probability and mathematical statistics. Wiley.

[Rugina et al., 2007] Rugina, A.-E., Kanoun, K., and Kaâniche, M. (2007). A System Dependability Modeling Framework Using AADL and GSPNs. In Lemos, R., Gacek, C., and Romanovsky, A., editors, *Architecting Dependable Systems IV*, volume 4615 of *Lecture Notes in Computer Science*, pages 14–38. Springer Berlin Heidelberg.

[Sanders and Meyer, 1991] Sanders, W. H. and Meyer, J. F. (1991). A Unified Approach for Specifying Measures of Performance, Dependability, and Performability. *Dependable Computing and Fault-Tolerant Systems: Dependable Computing for Critical Applications*, 4:215–237.

[Sandhu et al., 1996] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-Based Access Control Models. *Computer*, 29:38–47.

[Scarfone and Mell, 2007] Scarfone, K. and Mell, P. (2007). Guide to Intrusion Detection and Prevention Systems (IDPS). Technical report, The National Institute of Standards and Technology (NIST). Special Publication 800-94.

[Schmidt and Wentzlaff, 2006] Schmidt, H. and Wentzlaff, I. (2006). Preserving Software Quality Characteristics from Requirements Analysis to Architectural Design. In Gruhn, V. and Oquendo, F., editors, *Software Architecture*, volume 4344 of *Lecture Notes in Computer Science*, pages 189–203. Springer Berlin / Heidelberg.

[Schneider, 2000] Schneider, F. B. (2000). Enforceable Security Policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50.

[Selic, 2007] Selic, B. (2007). 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. IEEE Computer Society.

[Shamir, 1979] Shamir, A. (1979). How to Share a Secret. *Communications of ACM*, 22(11):612–613.

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

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

[Thapa et al., 2010] Thapa, V., Song, E., and Kim, H. (2010). An Approach to Verifying Security and Timing Properties in UML Models. In *15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 193–202.

[Tran, 2006] Tran, T. (2006). Proactive Multicast-Based IPSEC Discovery Protocol and Multicast Extension. In *IEEE Military Communications Conference (MILCOM)*, pages 1 –7.

[Tricas, 2003] Tricas, F. (2003). *Deadlock Analysis, Prevention and Avoidance in Sequential Resource Allocation Systems*. PhD thesis, Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza.

[Tricas et al., 2000] Tricas, F., Vallés, F., Colom, J., and Ezpeleta, J. (2000). An Iterative Method for Deadlock Prevention in FMS. In Boel, R. and Stremersch, G., editors, *Discrete Event Systems. Analysis and Control*, pages 139–148, Boston, USA. Kluwer Academic Publishers, Kluwer Academic Publishers.

[Trujillo et al., 2009] Trujillo, J., Soler, E., Fernández-Medina, E., and Piattini, M. (2009). A UML 2.0 profile to define security requirements for Data Warehouses. *Computer Standards & Interfaces*, 31(5):969–983.

[van der Aalst and van Hee, 2004] van der Aalst, W. and van Hee, K. (2004). *Workflow Management: Models, Methods, and Systems*, volume 1 of *MIT Press Books*. The MIT Press.

[Vossberg et al., 2008] Vossberg, M., Hoheisel, A., Tolxdorff, T., and Krefting, D. (2008). A Reliable DICOM Transfer Grid Service Based on Petri Net Workflows. In *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 441–448, Washington, DC, USA. IEEE Computer Society.

[Wang and Zeng, 2008] Wang, H. and Zeng, Q. (2008). Modeling and Analysis for Workflow Constrained by Resources and Nondetermined Time: An Approach Based on Petri Nets. *IEEE T. Syst. Man. Cy. A.*, 38(4):802–817.

[Wing, 2003] Wing, J. M. (2003). A Call to Action: Look Beyond the Horizon. *IEEE Security and Privacy*, 1:62–67.

[Wolter and Meinel, 2010] Wolter, C. and Meinel, C. (2010). An Approach to Capture Authorisation Requirements in Business Processes. *Requir. Eng.*, 15:359–373.

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

[Wu et al., 2008] Wu, N., Zhou, M., and Li, Z. (2008). Resource-Oriented Petri Net for Deadlock Avoidance in Flexible Assembly Systems. *IEEE T. Syst. Man. Cy. A.*, 38(1):56–69.

[Yskout et al., 2008] Yskout, K., De Win, B., and Joosen, W. (2008). Transforming security audit requirements into a software architecture. In Whittle, J., Jurjens, J., Nuseibeh, B., and Dobson, G., editors, *Proceedings of the Workshop on Modeling Security (MODSEC08)*, pages 1–10. CEUR Workshop Proceedings (CEUR-WS.org).

[Zhou et al., 2002] Zhou, L., Schneider, F. B., and Van Renesse, R. (2002). COCA: a Secure Distributed Online Certification Authority. *ACM Trans. on Computer Systems (TOCS)*, 20(4):329–368.