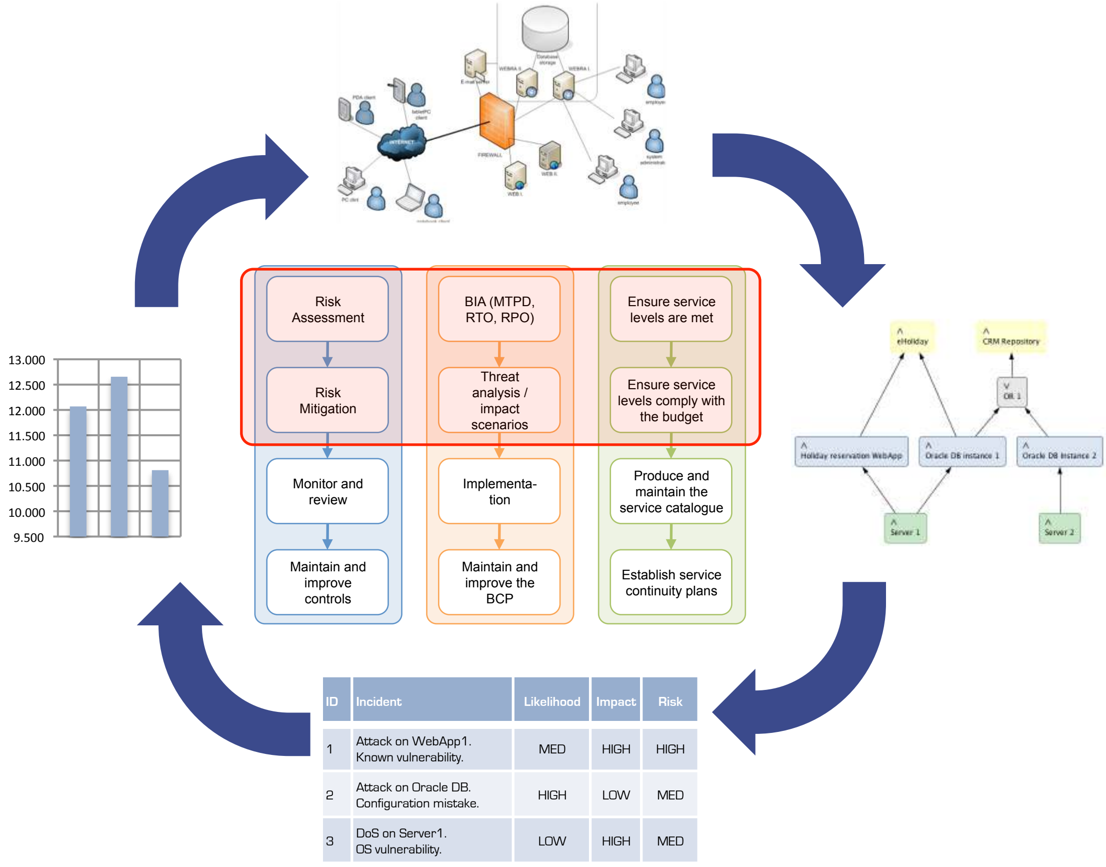
TOWARDS OPTIMAL IT AVAILABILITY PLANNING

METHODS AND TOOLS



Risk Assessment → Risk Mitigation → Monitor and review → Maintain and improve controls

BIA (MTPD, RTO, RPO) → Threat analysis / impact scenarios → Implementation → Maintain and improve the BCP

Ensure service levels are met → Ensure service levels comply with the budget → Produce and maintain the service catalogue → Establish service continuity plans

ID	Incident	Likelihood	Impact	Risk
1	Attack on WebApp1. Known vulnerability.	MED	HIGH	HIGH
2	Attack on Oracle DB. Configuration mistake.	HIGH	LOW	MED
3	DoS on Server1. OS vulnerability.	LOW	HIGH	MED

Emmanuele Zambon

Towards Optimal IT Availability Planning: Methods and Tools

Emmanuele Zambon

Composition of the Graduation Committee:

Chairman and Secretary	
Prof. dr. ir. A.J. Mouthaan	Universiteit Twente
Promotors	
Prof. dr. S. Etalle	Universiteit Twente
Prof. dr. R.J. Wieringa	Universiteit Twente
Members	
Prof. dr. P.H. Hartel	Universiteit Twente
Dr. A. Pras	Universiteit Twente
Prof. dr. F. Massacci	Università di Trento
Prof. dr. E.R. Verheul	Radboud Universiteit Nijmegen
Dr. A. Herrmann	Axivion GmbH

CTIT Ph.D. Thesis Series No. 10-188
Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE
Enschede, The Netherlands

IPA: 2011-03
The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithms).

ISBN: 978-90-365-3102-3
ISSN: 1381-3617
DOI: 10.3990./1.9789036531023
http://dx.doi.org/10.3990/1.9789036531023

Typeset with LaTeX. Printed by Wöhrmann Print Service.
Cover design: Emmanuele Zambon and Nicole Mazzocato.

Towards Optimal IT Availability Planning:
Methods and Tools

DISSERTATION

to obtain
the doctor's degree at the University of Twente
on the authority of the rector magnificus,
prof. dr. H. Brinksma,
on account of the decision of the graduation committee,
to be publicly defended
on Thursday, 20th of January 2011 at 13.15

by

Emmanuele Zambon

born on 27th of November 1980,
in Vicenza, Italy

The dissertation is approved by:

Prof. dr. S. Etalle	Universiteit Twente (promotor)
Prof. dr. R.J. Wieringa	Universiteit Twente (promotor)

To Nicole

Abstract

The availability of an organisation's IT infrastructure is of vital importance for supporting business activities. IT outages are a cause of competitive liability, chipping away at a company financial performance and reputation. To achieve the maximum possible IT availability within the available budget, organisations need to carry out a set of analysis activities to prioritise efforts and take decisions based on the business needs. This set of analysis activities is called *IT availability planning*.

Most (large) organisations address IT availability planning from one or more of the three main angles: information risk management, business continuity and service level management. Information risk management consists of identifying, analysing, evaluating and mitigating the risks that can affect the information processed by an organisation and the information-processing (IT) systems. Business continuity consists of creating a logistic plan, called business continuity plan, which contains the procedures and all the useful information needed to recover an organisations' critical processes after major disruption. Service level management mainly consists of organising, documenting and ensuring a certain quality level (e.g. the availability level) for the services offered by IT systems to the business units of an organisation.

There exist several standard documents that provide the guidelines to set up the processes of risk, business continuity and service level management. However, to be as generally applicable as possible, these standards do not include implementation details. Consequently, to do IT availability planning each organisation needs to develop the concrete techniques that suit its needs. To be of practical use, these techniques must be accurate enough to deal with the increasing complexity of IT infrastructures, but remain feasible within the budget available to organisations. As we argue in this dissertation, basic approaches currently adopted by organisations are feasible but often lack of accuracy.

In this thesis we propose a graph-based framework for modelling the availability dependencies of the components of an IT infrastructure and we develop techniques based on this framework to support availability planning. In more detail we present:

1. the Time Dependency model, which is meant to support IT managers in the selection of a cost-optimal set of countermeasures to mitigate availability-related IT risks;
2. the Qualitative Time Dependency model, which is meant to be used to systematically assess availability-related IT risks in combination with existing risk assessment methods;
3. the Time Dependency and Recovery model, which provides a tool for IT managers to set or validate the recovery time objectives on the components of an IT architecture, which are then used to create the IT-related part of a business continuity plan;
4. A^{2}THOS, to verify if availability SLAs, regulating the provisioning of IT services between business units of the same organisation, can be respected when the implementation of these services is partially outsourced to external companies, and to choose outsourcing offers accordingly.

We run case studies with the data of a primary insurance company and a large multinational company to test the proposed techniques. The results indicate that organisations such as insurance or manufacturing companies, which use IT to support their business can benefit from the optimisation of the availability of their IT infrastructure: it is possible to develop techniques that support IT availability planning while guaranteeing feasibility within budget. The framework we propose shows that the structure of the IT architecture can be practically employed with such techniques to increase their accuracy over current practice.

Acknowledgements

This last four years of my life have been a true adventure. This final achievement would have never been possible without the help of special people I would like to thank.

I first met Sandro in a pub in the UK (we were there for a conference, do not misunderstand!) He begun the conversation – we were in front of a couple of beers – with his usual question: “what do you want to do in your life?”. And within two hours I was convinced to apply for a PhD position. He has become my daily supervisor: I couldn’t have been more lucky! He is a brilliant researcher and an amazing coach, he taught me most of the things I know about scientific research and about writing papers. And he has provided me the best motivations to finish my PhD even when I couldn’t see the end of it. But most importantly, he has become a good friend.

It was not before the second year of my PhD that I got to know Roel, my promotor, he had been ill for a very long time. But even before I met him I had the pleasure to read his monthly reports about the status of his medications and all the interesting things he learnt about being hospitalised: real fun! Then we started working on research. It was challenging, but it was worth it. During our monthly meetings you gave direction to my research (“What is the problem we are solving here?”), helping to put the pieces together. And we also wandered in historical conversations and “meta-questions” which were always interesting. Thanks.

I want to thank Pieter for all the support he gave me during these four years: he has welcome me at the DIES group and kept an eye upon me. Thanks again for reading this thesis so thoroughly, for giving me valuable comments that really helped me improve it and for patiently answering all my questions all the times I was popping up at your door.

The person who hatched to have me sitting at the same table with Sandro in that pub is my friend Damiano. I will never thank him enough for what he did. I have met Damiano at high school, and from then on we have been studying, working and having fun together. During these four years we have also done research together, probably not as much as I would have wanted, though. In fact, when we do research together we are able to create this amazing process of one coming out with a vague idea and the other one taking the idea to the next level and so on until it becomes something very interesting: I still remember the two of us inventing a new anomaly detection engine in only one night! Already from the early days we have been trying to start up our own business (I remember the first attempt was in the field of CDs and DVDs . . .). It has been quite a winding road to come to SecurityMatters, and it would never have been possible to reach this point without the key insertion of Sandro in what I now consider a damn good team: I think we must be proud of it.

Nicole, you are the most important person in my life and I have many things to thank you for. You gave me the freedom of doing the PhD, even if this implied not being together for almost three years. You have turned this into a strength: after all the time apart we are now more tied than ever. You chose to join me in the Netherlands, winning all the hesitations, and now we have the opportunity of living together. You always supported me, even in the darkest moments . . . and you also proofread this thesis twice!

Thanks to the RMC team at "The Company" for the support they provided during the case studies: Jeroen, Coen, Peter, Barry, Leo and Wim.

Thanks to colleagues and friends for the good time spent together. Ayse, my research and journey mate, Stefano, Marco, Anna, Lorenzo, Zlatko, Lianne, Andreas, Daniel, Dulce, Chen, Dina, Julius, Michele, Nienke, Bertine. Thanks to my former colleagues at Valueteam and KPMG (especially Marco for his contributions to this thesis). My great friend and companion in music Claudio, who made me the honour of being my paranymph, and all the other band mates: Luca, Andrea, Sandro. My friends in Italy: Giulio, Damiano, Roby, Paolo, Nicolò, Jacopo, Davide, Stefano, Tommaso, Matteo, Giulio, Mirco, Valentina, Roberto and all the others that I cannot mention since I already communicated the final number of pages to the editor.

Il ringraziamento più grande va ai miei genitori. Questa tesi vi appartiene, perchè io sono il prodotto del vostro amore e di tutti i sacrifici che avete fatto per la mia educazione. È un debito che non potrò mai ripagare, consideratelo un anticipo.

Enschede, December 2010.

Contents

Chapter 1

Introduction

Today, organisations use Information Technology (IT) to support most of their business operations. The global connectivity brought by the Internet has created new business opportunities, such as Business Process Outsourcing or e-commerce, and boosted the business of telco companies. IT is widely used to develop, market and distribute products or services, as well as to support the business management activities (communications, accounting, customer relationship management, etc.). Organisations that could continue to operate without computers before mainframe or even the Internet era are now so heavily dependent on IT that they rely on a near 100% availability of their IT systems to carry out their business.

Therefore, guaranteeing the availability (defined as: ensuring that authorised users have access to information and associated assets when required [46]) of business-supporting IT systems has become important for these organisations [60, 89, 109, 105]. IT outages are a cause of competitive liability, chipping away at a company financial performance and reputation. A report based on a 2007 survey from HP [16] estimates average hourly cost of downtime to the considerable amount of $ 90,000 (per company), with a loss of nearly $ 1M per outage. Disasters involving availability of IT systems are fairly common, since nearly 31% of companies polled in the survey by HP had to carry out their plans in a real disaster. However, most downtime is caused by non-disastrous events. 90% of downtime reported by survey respondents was due to network/telecommunications issues, hardware or software failures or operator errors.

To deal with IT outages, organisations can adopt a wide range of technical solutions that have been refined over the years. For example, a classic solution for availability is redundancy, which consists of duplicating the critical components of a system in such a way that when one of them fails it is replaced by its duplicate and the system continues to operate. However, such measures are expensive and the budget organisations can spend on IT availability is limited. Budget is mainly limited by two factors: first, the spendings for maintaining IT systems must not exceed the benefit these systems provide to the organisation and secondly, there are constraints imposed by the environment the organisation is operating in, such as laws and regulations for government organisations, or market competition for enterprises. The best an organisation can do is to find the optimal balance between the achieved availability and its cost (the cost of the work needed for finding the balance must be taken into account as well). However, achieving such an optimal balance is difficult. It requires knowledge from different domains: business management, IT management and security. For this reason, different people from different fields are usually involved, with communication problems and conflicting goals. Achieving an optimal balance also requires that business and IT are properly aligned and that decisions are made in each case based on the global business objectives, the technological constraints and the security threats.

In this thesis we focus on the analysis activities that organisations carry out to control the availability of business supporting IT systems.

1.1 Availability Planning

We call *IT availability planning* the set of analysis activities by which organisations set the requirements and take decisions regarding the availability of the IT systems supporting their business. Availability planning allows organisations to find the design for the availability of their IT infrastructure that supports their business at best within the budget limitations. Guidelines for planning the availability of IT are given in standard IT management methodologies such as COBIT [90] and ITIL [62]. Most (large) organisations address IT availability planning from one or more of the three main angles: risk management, business continuity and service level management, which we will now introduce.

Information Security Risk Management

Information security risk management is the process of dealing with the risks information and information processing assets (including IT assets) are exposed to.

Risk management is widely considered a key factor for improving an organisation's IT performance. Risk management is also required by regulation, such as the Sarbanes-Oxley Act of 2002 [112] or the international agreement Basel II [86] (International Convergence of Capital Measurement and Capital Standards), to ensure that the organisation is operating properly.

To introduce the risk management process we follow ISO 27005 (former BS 7799-3 [21]), one of the most popular standards: the same general principles are shared by almost all risk management standards.

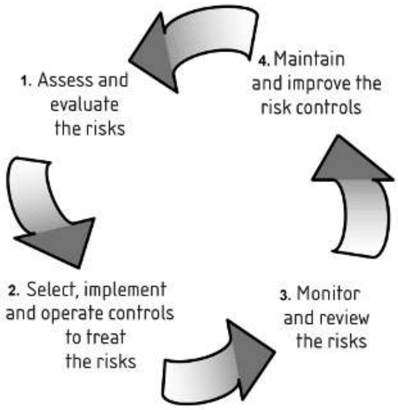

1. Assess and evaluate the risks
4. Maintain and improve the risk controls
2. Select, implement and operate controls to treat the risks
3. Monitor and review the risks

Figure 1.1: The Risk Management process model of ISO 27005

Risk management consists of four main tasks (see Figure 1.1): (1) assessing and evaluating the risks (risk assessment), (2) selecting and implementing controls to treat the risks (risk treatment or risk mitigation), (3) monitoring and reviewing risks and (4) maintaining and improving the risk controls. The whole process is cyclic and it is meant to be repeatedly applied during the life cycle of the IT system(s) under consideration. The two tasks of risk management that are more relevant for availability planning are Risk Assessment (RA) and Risk Mitigation (RM).

Risk management is relevant for optimising IT availability in that it enables the organisation to discover the risks to the business associated to disruptive events on the IT infrastructure, to rank them according to the business objectives and to plan the most effective strategies to deal with them.

A risk assessment identifies potential harmful threats and vulnerabilities of the system target of assessment, determines their likelihood, the harm they can cause and ranks them accordingly. Figure 1.2 shows the interpretation of risk given in NIST SP 800-100 [18], as a function of threat, vulnerability, likelihood and impact. Risk management best practices prescribe that risk assessments should be run periodically, to cope with the evolution of the target system, of the organisation using the system and of the security related issues.

The second main task, risk mitigation, consists of developing and implementing a strategy to manage risks by choosing a proper risk treatment strategy and

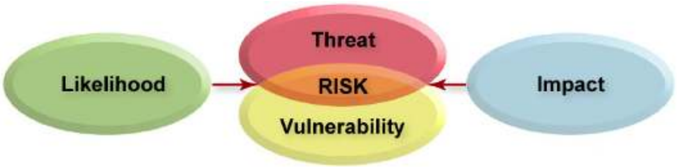


Figure 1.2: The risk function in NIST SP 800-100

by implementing controls. Risk management strategies are risk avoidance (eliminate, withdraw from or not become involved), risk reduction (optimise - mitigate), risk sharing (transfer - outsource or insure) and risk retention (accept and budget). Controls can be technical and organisational (involving people and procedures).

Business continuity

Business continuity management is the process supporting the recovery of interrupted business critical functions after a disruptive incident. Incidents include local incidents (e.g. building fires), regional incidents (e.g. earthquakes), or national incidents (e.g. pandemic illnesses). The outcome of business continuity is a logistic plan called the Business Continuity Plan (BCP).

When an organisation has IT systems supporting its business operations, part of the business continuity plan must address the recovery of the IT infrastructure.

The process of planning, implementing and maintaining a business continuity plan is described in the BS 25999-1 standard [41] released by the British Standard Institute but widely used also outside the United Kingdom. According to this standard, the main activities of business continuity management involving IT can be summarised as (a) an analysis of the (business) continuity requirements for the components of the IT infrastructure, (b) an analysis of threats and their impact scenario, (c) the design and implementation of business continuity strategies (the BCP) satisfying the business requirements with regards to the different impact scenarios, and (d) the maintenance and improvement of the BCP. Figure 1.3 shows the main activities of business continuity management and their relation.

We now describe in more detail the steps involved in activities (1) and (2), which are the ones that have mostly to do with IT availability planning:

1. *Business Impact Analysis (BIA)*: BIA is the study and assessment of effects to the organisation in the event of the loss or degradation of business functions resulting from a destructive event (incident).

2. *Set the Maximum Tolerable Period of Disruption (MTPD)*: based on the results of the BIA, an MTPD has to be set for all the key business activities.

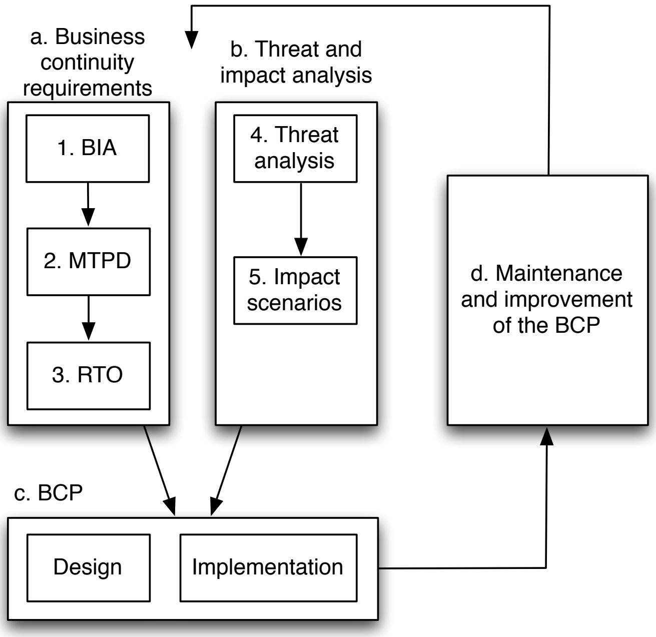

a. Business continuity requirements
1. BIA
2. MTPD
3. RTO
b. Threat and impact analysis
4. Threat analysis
5. Impact scenarios
d. Maintenance and improvement of the BCP
c. BCP
Design
Implementation

Figure 1.3: The main tasks of business continuity management. Blocks are tasks, and edges indicate that information from one block is used in the other

The MTPD expresses the "duration after which an organisation's viability will be irrevocably threatened if product and service delivery cannot be resumed" [41].

3. *Set the Recovery Time Objectives (RTO)*: based on the MTPD, an RTO is determined for all the assets (i.e. people, premises, IT systems) that support a certain activity. The RTO expresses the amount of time to restore an asset. In case a certain business activity is supported by IT, the RTO need to be determined for each component of the IT infrastructure.

4. *Threat analysis*: this step consists of selecting and analysing the threats that could compromise the organisation business. Typical threats taken into account in this analysis include natural disasters (e.g. floods and earthquakes), terrorist attacks or pandemic infections.

5. *Impact scenarios*: based on the results of the threat analysis, several scenarios in which a threat materialises are taken into consideration. For each scenario a (worst-case) estimate is made of the impact on the organisation's

assets (in this case IT components). These scenarios are then grouped together and used to build the BCP.

A BCP specifies the recovery procedures to ensure RTOs can be respected in case of different impact scenarios (e.g. major power failure, loss of a building etc.). For IT, these procedures include the minimal set of IT components that are needed to run the organisation's business functions, and the order on which IT components should be recovered, based on their RTO. Other sections of a BCP include the backup strategies for IT processed information that should make sure that business relevant data is up-to-date with respect to a predefined Recovery Point Objective (RPO). Finally, the BCP needs to be regularly updated as soon as changes happen in the organisation. The plan is tested by simulating recovery scenarios (or when it is actually used in case of an incident) and improved accordingly.

Business continuity contributes to optimise IT availability in case of incidents by limiting the losses they cause to the organisation.

Service level management

An IT service abstracts a functionality provided by the IT infrastructure to its final users (e.g. sending and receiving e-mails). Organisations business units are IT service users. IT services can either be acquired internally from the IT department, or externally from IT outsourcing companies.

The quality of IT services is controlled through Service Level Agreements (SLAs). An SLA is a contract specifying the (measurable) value agreed by the service provider and the service user for a certain quality parameter. For instance, the cost of a service usually depends on the SLA associated to it. Organisations use SLAs to guarantee that the IT services comply with the business requirements. The process of managing SLAs is called Service Level Management.

The ITIL framework provides guidelines on how to do Service Level Management (SLM). The four main tasks involved are: (1) ensuring that agreed service levels are met, (2) ensuring service levels comply with the available budget, (3) producing and maintaining a catalogue of the services and (4) establishing service continuity plans.

Availability is one of the most used quality parameters for SLAs. For example, a typical SLA for availability is to guarantee that a service will have – say – 99% of uptime in a month. Therefore, a successful SLA management is important to optimise IT availability, as it allows the organisation to set a trade-off between the availability level and its associated cost during normal business operations.

Summarising, to successfully plan the availability of an IT infrastructure, IT managers need first to agree with the business units on the required availability levels for the IT systems supporting the organisation's business. They then need to control the IT infrastructure and make sure availability levels can be respected within the available budget. The main control points are based on (1) the management of availability-related IT risks, which need to be identified, evaluated and mitigated when needed, (2) the IT-related section of the BCP, which is meant to ensure that IT systems are recovered after disasters within a predefined time agreed with the business units and (3) the contractual agreements (SLAs) with IT service providers to make sure that the required availability level for IT services is guaranteed to business units during normal circumstances.

These three control points address IT availability from three different angles, which require different techniques. However, these angles target the same IT infrastructure. In any case the analyst has to determine *how* the infrastructure behaves in case one or more of its components fail and how such failures relate to the supported business activities. Based on this observation, the models and techniques we will present in this thesis share the same underlying representation (dependency graphs) of the main availability properties of the IT system under exam.

Figure 1.4 provides an overview of the concepts we just described, and identifies the activities related to availability planning in the three angles of risk, business continuity and service level management.

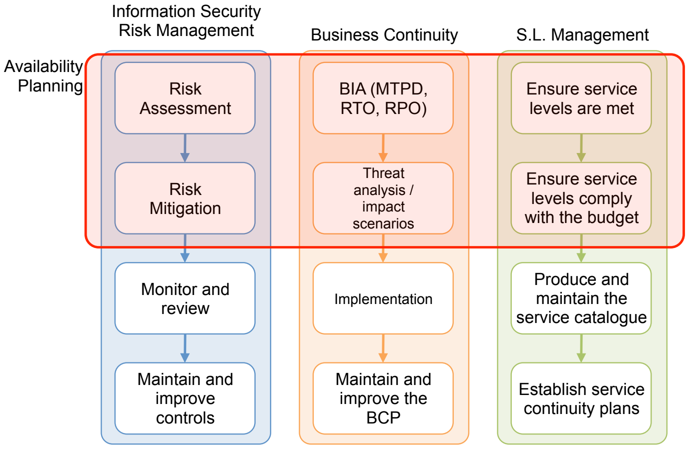


Figure 1.4: Availability planning in relation to risk management, business continuity and service level management

1.2 The Problem

The standards we mentioned so far draw the guidelines for doing the activities described in Section 1.1. However, to be as generally applicable as possible, these standards do not include implementation details. For example, risk assessment standards indicate that the risk assessor should identify threats, but do not specify how this is done in practice. For this reason, each organisation that wants to optimise its IT availability need to find the concrete *techniques* that suit its needs.

To be of practical use, such techniques must comply with at least two requirements:

1. *accuracy*: they must allow the calculation of the different availability figures needed for risk management (e.g. the system outage caused by an incident), business continuity planning (e.g. the system recovery time) and service level management (e.g. the minimal monthly system uptime) as precisely as needed;

2. *feasibility within budget*: their use must require an amount of information and resources that the organisation is able to provide.

There is a group of more advanced approaches proposed by the academic community and another of more basic approaches adopted by the business community. Many of the approaches in the first group consist of statistical models describing in detail the functional aspects of the IT infrastructure to be analysed in relation to the probabilities of the various infrastructure components failures. For each infrastructure component the analyst has to define the relevant internal states of the component, the probability of transition from one state to the others and the connection of each component state to the state of the other components in the infrastructure. With such a model one can in principle deduce any required availability figure of the IT infrastructure. There exist several modelling techniques that can be used for this purpose. For example, Markov models, Bayesian networks and Petri nets have been used in the reliability field for the design and analysis of a number of availability critical systems. Although exact, these techniques are not often applied to plan the availability of business supporting IT infrastructures because of scalability issues. To apply them, an analyst has to model all the internal states of each component and deduce (or estimate) the transition probability for each pair of states. Obtaining this information requires a considerable effort. Therefore, this kind of analysis process is in most cases too slow to comply with the requirement of being feasible within budget.

The second group uses limited or no modelling and mainly relies on the expertise of the personnel devoted to the task for the availability analysis. However, due to the increasing complexity of the IT infrastructure to be managed, even experts can make mistakes. Incidents affecting the availability of a marginal component of an IT system can propagate in unexpected ways to other, more essential components that functionally depend on the presumed marginal component. Mistakes in availability planning can lead to costly IT service disruptions, or to overspending to obtain an availability level which is too high with respect to the organisation's business needs. For example, underestimating the system availability can lead to the adoption of costly countermeasures which are not actually required.

For these reasons, organisations aim at improving the quality of their risk, business continuity and service management processes using methods with a higher degree of accuracy than current practice affords, but that are still feasible within budget. It is the goal of this thesis to propose and validate such a method.

1.3 Technical Research Questions

Based on the analysis of the above mentioned problem, this work focuses on the following practical research aim:

"Design and validate techniques that improve the accuracy and effectiveness of availability planning, while guaranteeing feasibility within budget."

To achieve this aim we focus on the following research questions.

1. *"How can we improve the accuracy of current techniques for assessment and mitigation of availability-related IT risks, while guaranteeing feasibility within budget?"*
   The assessment of availability-related risks requires techniques that can accurately determine the consequences (impact) the disruption of an IT component can have on the IT infrastructure and on the business operations supported by it. Optimisation techniques are also required during the risk mitigation phase to support the decision process of adopting the most cost-effective countermeasures to protect agains risks.

2. *"How can we improve the accuracy of current techniques for creating and maintaining business continuity plans, while guaranteeing feasibility within budget?"*
   Creating and maintaining an effective business continuity plan requires techniques and tools to make sure business continuity requirements set on IT components are aligned with the business needs. In other words, with such techniques analysts can check that RTOs are compliant with the existing

MTPDs. Due to budget limitations, it could be the case that MTPDs cannot be respected in all cases. Therefore, analysts and decision makers need techniques to estimate how often this is expected to happen, and therefore the risk of not complying with MTPDs.

3. *“How can we improve the accuracy of current techniques for managing availability-related SLAs, while guaranteeing feasibility within budget?”* To ensure the availability level of a service is met, techniques are needed to properly calculate the availability of an IT service at design or implementation phase. With this information, it is possible to set availability service levels which can be met during the service life. When planning availability service levels, it is also important to comply with budget limitations. For this reason, techniques are needed to support the cost/benefit decisions IT managers have to take regarding the design choices that influence the availability of IT services.

1.4 Contributions

To address the research questions we have developed a set of architecture-based techniques that support availability planning. Figure 1.5 gives an overview of our suite of techniques.

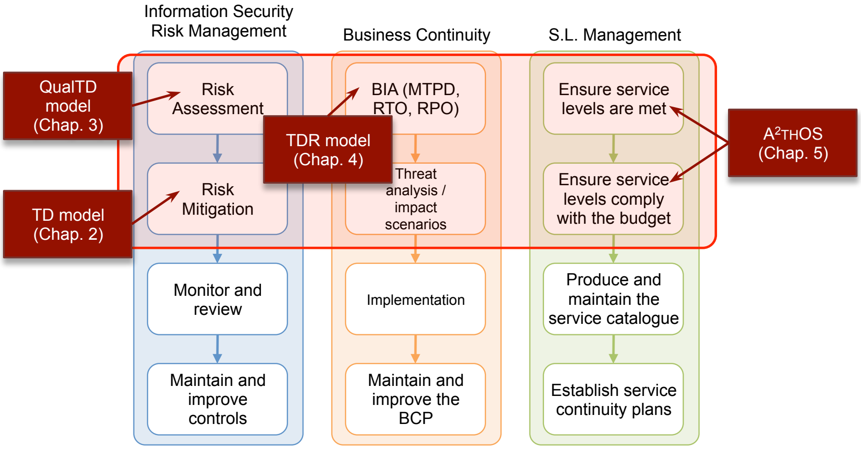

Information Security Risk Management
Business Continuity
S.L. Management
QualTD model (Chap. 3)
TD model (Chap. 2)
TDR model (Chap. 4)
A²THOS (Chap. 5)
Risk Assessment
Risk Mitigation
Monitor and review
Maintain and improve controls
BIA (MTPD, RTO, RPO)
Threat analysis / impact scenarios
Implementation
Maintain and improve the BCP
Ensure service levels are met
Ensure service levels comply with the budget
Produce and maintain the service catalogue
Establish service continuity plans

Figure 1.5: An overview of our suite of techniques in relation on the availability planning activities they support the most

The Time Dependency (TD) model and the associated framework support the assessment and mitigation of availability-related IT risks. The model is based on a graph of the components of the IT architecture and of their dependencies. The framework allows one to determine the impact of the disruption of an IT component to the organisations processes and to optimise the choice of availability-related risk mitigation strategies according to the expected benefit they determine and on their cost. The framework follows the quantitative risk assessment paradigm, in which risks are expressed in a range of magnitudes which can be measured (e.g. expected monetary loss).

The Qualitative Time Dependency (QualTD) model and framework for the assessment of availability-related IT risks is an extension of the TD model with enhanced modelling capabilities (it supports a wider range of dependencies among the IT components). It also allows the risk assessor to relate the identified threats with the vulnerabilities of the IT components to determine the risk caused by availability-related incidents. The QualTD model is meant to be used for risk assessments that follow the qualitative paradigm, in which risks are described by values in an ordinal scale which at most allow value comparison (e.g. high, medium or low). This improves the feasibility of our technique, as it does not require quantitative data about incident likelihood and financial losses, which can be difficult to acquire. Under this aspect, the QualTD model can be also seen as an abstraction of the TD model in which numerical values are replaced by ordered labels.

The Time Dependency and Recovery (TDR) model and tool supports the assessment of a business continuity plan. It is based on the same representation of the IT infrastructure we use in the TD model but it includes the concept of incident repair time. The model allows one to assess a business continuity plan by checking whether the MTPDs set on the critical business activities are met by the RTOs set on the underlying IT infrastructure, and whether RTOs are truly pairwise compatible. The model also allows to evaluate the risk that MTPDs are exceeded.

A^{2}THOS is a framework to calculate the availability of partially outsourced IT services in presence of SLAs. A^{2}THOS consists of a model of an IT system (which provides multiple services), an algorithm to calculate the minimal availability of each service given the minimal availability of the (outsourced) service components, and an algorithm to compute the cost-optimal choice of the availability of the system components based on the target availability of the exported services. There exist techniques, such as fault trees, which allow one to calculate the availability of a system. However, such techniques are not always applicable in case of the outsourcing of system components, as the required information is not available: A^{2}THOS overcomes this limitation.

1.4.1 Thesis Overview and Publications

We now explain the contributions of each chapter of this work.

Quantitative Decision Support for Model-Based Mitigation of Availability Risks (Chapter 2) In this chapter we present the TD model and describe how to use it to determine the risk caused by the disruption of a component of the architecture. We then show how to model risk mitigation strategies and how to determine the set of these strategies which has the best cost/benefit trade-off. Finally, we discuss the feasibility and implementation of our model based on the information about a risk assessment carried out by KPMG-Italy at an insurance company. This work appears in a refereed workshop paper [7], which is joint work with D. Bolzoni, S. Etalle and M. Salvato.

Model-Based Qualitative Risk Assessment for Availability of IT Infrastructures (Chapter 3) In this chapter we introduce the QualTD model. We then show how to apply the QualTD model in a practical case-study we carried out on the authentication and authorisation system of a large multinational company. Based on the case-study we also address the accuracy of our technique in relation to the ones used by the company and then deepen the discussion on its feasibility by presenting a review of risk assessment methodologies and their compatibility with our technique. This work appears in a journal paper [2], which is joint work with S. Etalle, R.J. Wieringa and P.H. Hartel.

A Model Supporting Business Continuity Auditing & Planning in Information Systems (Chapter 4) In this chapter we present the TDR model. We describe how to use the model to assess a business continuity plan and to evaluate the risk that MTPDs are exceeded. Finally, we discuss the feasibility and implementation of our model based on the IT infrastructure of an insurance company provided by KPMG-Italy. This work appears in a refereed conference paper [5], which is joint work with D. Bolzoni, S. Etalle and M. Salvato.

A^{2}THOS: Availability Analysis and Optimisation in SLAs (Chapter 5) In this chapter we present A^{2}THOS. We first introduce the model and provide the theoretical foundations for calculating and optimising the IT system availability based on the model. We then discuss the feasibility and usefulness of our framework based on two case-studies we carried out at a large multinational company. This work appears in a journal submission [1], which is joint work with S. Etalle and R.J. Wieringa.

Chapter 2

Quantitative Decision Support for Model-Based Mitigation of Availability Risks*

We start here with the first research question:

"How can we improve the accuracy of current techniques for assessment and mitigation of availability-related IT risks, while guaranteeing feasibility within budget?"

Risk management is addressed in two separate chapters of this thesis: the present one focuses on the mitigation of availability-related risks, while the second one on their assessment.

Although these two steps of risks management should be logically presented in the reverse order, we prefer this one as the model for risk mitigation was developed before the one for risk assessment, and the latter extends some of the concepts presented in the former.

*This chapter is a minor revision of the paper "Model-Based Mitigation of Availability Risks" [7] published in the Proceedings of the Second IEEE/IFIP International Workshop on Business-Driven IT Management (BDIM '07), pages 144-156, IEEE Computer Society, 2007.

2.1 Introduction

In this chapter we focus on mitigating the risks related to the availability of the IT infrastructure. This is particularly challenging because of the (temporal) dependencies linking the various constituents of an IT infrastructure (machines, processes, assets, etc.) with each other. In complex information systems, a failure in a remote component may propagate across the infrastructure and eventually affect the availability of a good deal of the entire system. Failing to appropriately assess the consequences of such propagations will result in inaccurate RA and RMs.

We argue that current risk management methodologies (e.g. ISO 17799 [44], ISO 13335 [42] and OCTAVE [82]) show accuracy limitations when evaluating and mitigating availability risks. This is due to the fact that they do not fully consider the consequences of the functional dependencies between the constituents of an IT infrastructure: the consideration of these dependencies is mostly left to the judgement of the assessor carrying out the RA phase (although this is not made explicit). Thus, these methodologies are mainly useful to identify and fix individual risks an organisation is exposed to (see also Section 2.2). On the other hand, these dependencies are mentioned in more specific assessment methods such as the Business Continuity Plans, like in the new standard BS25999 [41] (see Section 2.2 for a detailed overview). These methods, however, do not specify how to use this information for RM, making their use unfeasible.

Our thesis is that it is possible to carry out an accurate tool-based RM by using the data collected during RA and BCP activities, under the hypothesis that such data is available and sufficiently accurate. To substantiate this thesis, in this chapter we present a framework and a tool for the assessment and mitigation of availability-related IT risks. The framework is based on the Time-Dependency (TD) model, an extended instance of the IT infrastructure model as it is described in BS25999 (which largely coincides with the data collected by the KARISMA tool developed at KPMG for RA, see Section 2.4). This model allows us to determine how incidents will propagate across the organisation, and therefore what is the actual *impact* of incidents. With this information, we can carry out an optimisation study by comparing the true expected benefit determined by the different *countermeasures* that can be put in place to cope with the various risks.

As we will mention, the computational complexity of the problems posed by our method, make it impossible to carry out the underlying analysis by hand, and this is why the method we propose requires the presence of an appropriate tool. We have implemented the tool using UPPAAL CORA [52] and Prolog.

We consider our solution a concrete enhancement to RM methodologies, providing automatic support to better evaluate the IT relationships and dynamics.

The remainder of this chapter is organised as follows: in Section 2.2 we briefly introduce some of the methodologies describing the current practice in IT and risk management. In Section 2.3 we present the TD model and show with a running example how it can be used to develop a cost-optimal risk mitigation strategy. In Section 2.4 we describe the prototype implementation of these algorithms and their use in combination with a risk management supporting tool developed at KPMG. In Section 2.5 we discuss the feasibility of our model both on the information needed to build it and on the computational complexity of the algorithms to determine the cost-optimal risk mitigation strategy. Finally, in Section 2.6 we present the related work.

2.2 Relevant methodologies for IT availability management

There exists a number of standards and methodologies for IT management as we briefly introduced in Chapter 1. Among them, COBIT (Control Objectives for Information and related Technology) [90] and BS25999 [41] are of particular relevance for this work. COBIT is the *de facto* standard for IT control and management, addressing IT governance and control practices. It provides a reference framework for managers, users and security auditors. COBIT is mostly based on the concept of *control* (be it technical or organisational) which is used to assess, monitor and verify the current state of a certain process (that may refer to procedures, human resources, etc.) involved in the IT system. To implement COBIT, the organisation must benchmark its own processes against the control objectives suggested by the framework, using the so-called *maturity models* (derived from the Software Engineering Institute's Capability Maturity Model [65]). Maturity models basically provide: (1) a measure for expressing the present state of an organisation, (2) an efficient way to decide which is the goal to achieve and, finally, (3) a tool to evaluate progress toward the goal. Maturity modelling enables one to identify gaps and demonstrate them to the management. Key Goal Indicators (KGI) and Key Performance Indicators (KPI) are then used to measure, respectively, when a process has achieved the goal set by management and when a goal is likely to be reached or not. Since COBIT does not suggest any technical solution but only organisational solutions, organisations often combine the control practices of COBIT with the technical security measures described in the *Code of Practice for Information Security Management* part of the ISO 17799 [44] standard.

Although COBIT does not provide any practical solution for mitigating the risks, it requires the organisation to implement a Business Continuity Plan (BCP) to improve the availability of its IT infrastructure and its core processes. Until 2003, no methodology was available to conduct this activity in a precise way. The new standard for managing business continuity BS25999 [41] is mainly focused on providing guidelines to understand, develop and implement a BCP, and aims at providing a standard methodology. This standard requires the organisation to complete different steps when preparing the BCP: (1) identify the activities/processes which carry the core service used by the organisation, (2) identify the relationships/dependencies among themselves, (3) evaluate the impact of the disruption of the core services/processes previously identified (Business Impact Analysis, BIA). The most critical activities/processes are intended to be the ones whose direct/indirect monetary loss is significantly high.

When the risk has been assessed and evaluated, one has to identify the best countermeasures to reduce the risk. Typically, there exists a number of different solutions (technical or organisational) from which business and IT managers must choose the best one(s) matching the required security level and the available budget (or finding the best compromise between the cost of the countermeasures and the benefit they provide). As we mentioned before, current methodologies are not sufficiently taking into account how business processes are linked together and the way a single incident could propagate and affect more of the organisation's IT systems. The fact that COBIT and ISO 17799 do not consider dependencies between processes has even greater impact in the mitigation phase of availability risks: it is standard practice to protect the processes whose availability has a greater *direct* impact on the organisation goals, while a more accurate analysis in many cases reveals that it is more cost effective to protect some of the processes that have an *indirect* impact as well.

2.3 The Time Dependency (TD) model

The framework we propose is based on a timed dependency graph, a directed and acyclic graph modelling the architecture of the organisation's IT-related infrastructure (including a part related to the organisation's business goals). To simplify the exposition, we indicate by $\mathbb{R}^+$ the set of nonnegative real numbers, and we use the following sets to indicate domains: $\mathbb{T}$ is the set of all time intervals (expressed in hours), $\mathbb{E}ur$ is the domain of monetary values (expressed in Euro).

Assumptions We start by providing a brief summary of the data we need to build the model, later we describe this data in more detail and we discuss about the feasibility of obtaining accurate information.

1. A *timed dependency graph*, consisting of: a set of *nodes* (processes, appli-

cations, etc.) and a set of *edges* between these nodes. Edges model which nodes depend on other nodes and must contain an estimate of *how long* a node would be able to survive if another one it depends on becomes unavailable. We express this measure in hours.

2. The *cost* associated to the downtime of those processes *directly* affecting the business objective of the organisation (indirect dependencies are taken care of by the model). We express this measure in Euro per hour.

3. A list of possible incidents affecting the IT infrastructure, together with a conservative estimate of the average downtime each of them cause (per node), given the controls already in place. We also need an estimate of their expected frequency. For the sake of uniformity, in the sequel we express the downtime caused by each incident in hours and their estimated frequency in times per year.

4. A list of *countermeasures*. For each countermeasure we need an estimate of (a) their deployment and maintenance costs (expressed in Euro per year), (b) the effect is has on the estimated frequency of the incidents and/or on the downtime they cause.

In Section 2.5 we address the problem of how and when this data can be collected during the RA and BCP processes.

Timed dependency graph The basic elements of the model are the constituents of the IT infrastructure. We follow notable architecture frameworks such as TOGAF [113], Zachman [114] and ArchiMate [84] as well as IT Governance solutions (IBM [26] and ISACA [90]), to determine those elements which may directly or indirectly be involved in an incident:

- *business processes*: the activities related to the organisations' business e.g. producing a specific product, managing customer orders or invoicing;

- *IT services*: the functionalities provided by IT systems to support business operations, e.g. e-mail service, digital identity management, instant messaging;

- *applications*: the software that provides IT services e.g. production control applications, customer relationship management (CRM) applications or databases;

- *technology*: hardware systems, computer networks and industry-specific technology needed to enable applications;

- *infrastructure or facilities*: physical locations necessary to house IT technology.

Running example - Part 2.1. *We present here an example (intentionally oversimplified) of the business/IT infrastructure of a small bank segment with ten components (see Table 2.1):*

Table 2.1: List of the components of a portion of an enterprise organisation's IT infrastructure and its supported business processes.

Id	Description
p_1	Customer management process
p_2	Financial services process
a_1	Home banking application
a_2	On-line trading application
a_3	Financial founds management application
db_1	Checking account database
db_2	Trading database
m_1	Application server machine
m_2	Oracle machine
m_3	Oracle machine
n_1	Network segment

p_1 *and* p_2 *are two business processes;* a_1*,* a_2 *and* a_3 *are three applications supporting business processes while* db_1 *and* db_2 *are two databases accessed by applications. Finally,* m_1*,* m_2 *and* m_3 *are the three machines running applications and* n_1 *is the network segment connecting the three machines.*

We represent the organisation's IT infrastructure and the business processes it supports by using a graph, where nodes represent the basic components of the infrastructure and labelled edges between nodes represent their dependencies. The presence of an edge from node a to node b indicates that b depends on a, and that if a becomes unavailable for long enough, b will become unavailable as well. In modelling this, we also indicate *how long* b will be able to survive without the presence of a. We do that by annotating each edge with the *survival time*: the time span the dependent node can survive if the other one fails. While for some dependencies, such as the dependency of an application on the machine it runs on, this amount is obviously set to zero, in case of dependencies between applications this can vary between zero and several hours (e.g. in case an application needs to be fed by another one with data at regular time intervals). Sometimes it is possible to extract this information from the functional requirements documentation or from the SLA specification. Although one can argue that these values could change over time, we have empirically verified (by inspecting documentation of several enterprise organisations) that risk management practice does not require such a level of detail yet. A tutorial on how to build dependency graphs can be found in Appendix B of this thesis.

Definition 2.1. *A* timed dependency graph *is a pair* $\langle N, \rightarrow \rangle$ *where* N *is a set of nodes and* $\rightarrow \subseteq N \times N \times \mathbb{T}$.

We write $n_1 \overset{t}{\rightarrow} n_2$ as shorthand for $(n_1, n_2, t) \in \rightarrow$.

A timed dependency graph allows one to express e.g. the dependencies of hardware components on the physical environment they are located in, the dependency of an application on the machines it runs on, and the dependency of a business process on the applications supporting it. We will show in Section 2.5 (as well as in Appendix B) that in certain cases the construction of this graph can be automated.

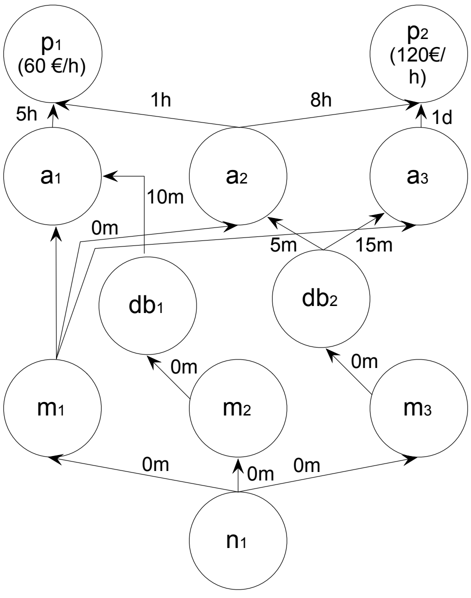


Figure 2.1: A timed dependency graph example

Running example - Part 2.2. *Figure 2.1 shows a* timed dependency graph *built with the components from Table 2.1. The edges connecting* n_1 *to* m_1*,* m_2 *and* m_3 *express the dependency of the machines on the network connection with other machines. The connections from* m_1 *to* a_1*,* a_2 *and* a_3*, from* m_2 *to* db_1 *and from* m_3

to db_2 express the dependency of software processes (applications or databases) on the machines they run on. For all of these connections the survival time is set to zero, since none of the components can survive the disruption of the ones it depends on, not even for a short time. In turn, p_1 depends on both a_1 and a_2, since the customer management is achieved by providing on-line banking and trading, but with different time constraints (five hours for a_1 and only one hour for a_2). Similar reasoning applies to a_1 and p_2.

Notice that these dependencies are *AND* relationships: a node depending on two or more other nodes is disrupted even if just one of these are affected by an incident. For the sake of simplicity, in this chapter we do not consider *OR* relationships, even though it would be possible to include them in our model (as we will see in Chapter 3).

The number of IT components can be very large in a real business environment. However, some of the information needed to build the graph can be available as a result of a RA (the first RA step, according to NIST methodology [73], is system characterisation). For instance, the KARISMA tool developed at KPMG to support RA requires – among other things – the collection of enough data to build an accurate timed dependency graph. Any other similar tool will basically do the same.

Incidents and their propagation Once the model of the IT architecture is defined, it is possible to simulate the availability of the system during and after the occurrence of an incident. We define incidents as events causing the unavailability of a given set of IT components for a given time.

Definition 2.2 (Incident repair time)**.** *Let $g = \langle N, \rightarrow \rangle$ be a timed dependency graph and $i \in I$ be an incident which disrupts a set of nodes $M \subseteq N$. The time needed to repair a node $n \in M$ because of i is a mapping* $\mathrm{rt} : I \times M \rightarrow \mathbb{T}$.

For instance, if we expect that the average occurrence of incident i would bring down machine m_1 for 3 hours, we model this by setting *rt*$(i, m_1) = 3$.

Running example - Part 2.3. *Let us now introduce three different incidents affecting the availability of m_3: Table 2.2 presents them.*

In i_1 one of m_3's hard disks is broken and the repair time is the average time required to replace the broken disk and restore data. i_2 consists of a power disruption in the building hosting m_3, in this case the repair time is the average duration of a power disruption. i_3 consists in an OS failure, due to software bugs, causing the consequent freeze of applications running in m_3 and the repair time is the average time needed to detect the incident and reboot m_3.

Table 2.2: A list of incidents possibly affecting m_3.

Id	Description	Target	Repair time
i_1	Disk failure	m_3	9h
i_2	Power disruption	m_3	3h
i_3	OS failure	m_3	2h

Every incident directly involves one or more nodes, causing them to be unavailable for a certain amount of time. During this time, the incident may propagate to other nodes, following the timed dependency graph.

We say that an incident *propagates* from a node n_1 to n_2, if they have a functional relationship and the unavailability time of n_1, due to the incident, exceeds the survival time of n_2 with respect to n_1, causing it to become unavailable until the incident is resolved.

Incident downtime According to this observation, we can define the downtime caused by an incident to any node of the timed dependency graph (including propagation). This is the crucial information needed in the Risk Evaluation and Mitigation phases to determine the global consequences of an incident, as we will address in Section 2.3.1.

Definition 2.3 (Incident downtime)**.** *Let $g = \langle N, \rightarrow \rangle$ be a timed dependency graph, $i \in I$ be an incident happening on a set of nodes $M \subseteq N$. The incident downtime is a mapping* $\mathrm{dt} : I \times N \rightarrow \mathbb{T}$ *defined as:*

$$\mathrm{dt}(i,n) = \begin{cases} \mathrm{rt}(i,n) & \textit{if } n \in M \\ 0 & \textit{if } n \notin M \textit{ and } D_n = \varnothing \\ 0 & \textit{if } \max_{m \xrightarrow{s} n \in \rightarrow} \mathrm{dt}(i,m) - s < 0 \\ \max_{m \xrightarrow{s} n \in \rightarrow} \mathrm{dt}(i,m) - s & \textit{else.} \end{cases}$$

This definition is well formed because we assumed g to be acyclic.

Running example - Part 2.4. *Figure 2.2 shows how i_1 propagates across our organisation.*

Assume that i_1 occurs at $t = 0$: i_1 brings down m_3; at the same time db_2 becomes unavailable, since its survival time with respect to m_3 is zero. After five minutes a_2 goes down and a_3 follows after fifteen minutes. Accordingly to the timed dependency graph, after one hour from the disruption of a_2, the process p_1 goes down

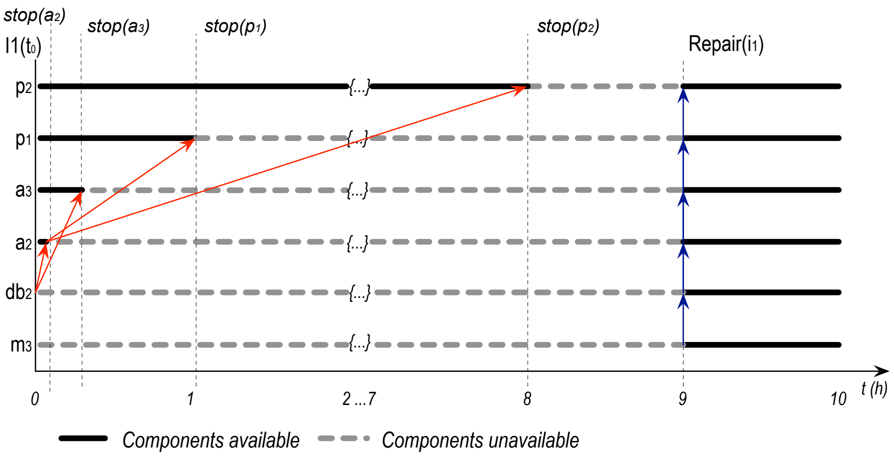


Figure 2.2: Propagation chart of incident i_1.

and after eight hours p_2 goes down as well. Nine hours after t_0, all nodes become available because i_1 has been repaired.

2.3.1 Risk mitigation

The timed dependency graph allows us to model the propagation of incidents. We now show how we can use this information for selecting the best set of countermeasures; technically we aim at finding the set of countermeasures which minimises the cost due to the forecasted downtime of relevant business processes.

2.3.1.1 Evaluating risk

The first step toward risk mitigation is an accurate evaluation of the *cost* (caused by losses) associated to the downtime of each process. In an organisation, there are usually only few processes which – if unavailable – directly cause a real damage (in our running example, only p_1 and p_2). Clearly, this cost depends on the business goals of the company (a one hour downtime of the web server has a different monetary cost at Google than at an insurance company). To model the cost of incidents we now define the *damage evaluation* function, relating the disruption time to the (monetary) loss affecting the organisation.

Definition 2.4 (Damage evaluation)**.** *Let $g = \langle N, \rightarrow \rangle$ be a timed dependency graph.*

*The business-driven damage evaluation function (*dam*) is a mapping from downtime to costs* dam : $N \times \mathbb{T} \rightarrow \mathbb{E}ur$.

Running example - Part 2.5. *In our simplified example, the downtime cost of* p_2 *is 120 Euro per hour (see Figure 2.1), so* $\mathrm{dam}(p_2, t) = 120t$*. This means that the occurrence of incident* i_1 *(which – after propagation – causes a downtime of 55 minutes on* p_2*) would create a damage of 110 Euro.*

In practice, *dam* may not be linear (a downtime of 24 hours may well cause more losses that 24 downtimes of one hour). In general, *dam* should be provided by the organisation's *business* department for the most important business processes and, in general, for all the business-relevant IT components in the organisation. In some cases, obtaining an accurate *dam* function can be a non-trivial task: this is the case of business processes which do not cause any direct financial loss if disrupted. In these cases, the organisation needs to quantify the loss of immaterial goods such as its reputation in the public opinion. Banks and insurance companies are among the organisations that are more prepared to carry out this task.

Frequencies and Global Cost Having determined the damage associated to an incident, we need now just one last factor for an accurate risk evaluation, and that is an assessment of the frequency (likelihood) of an incident.

Definition 2.5 (Incident frequency)**.** *Given a set of incidents* I*, the incident* frequency*, is a mapping* freq : $I \rightarrow \mathbb{R}^+$.

For instance, *freq*$(i) = 0.1$ means that estimates indicate that incident i is likely to happen once in ten years (on average). We should mention that NIST [73, 18] suggests a qualitative approach to assess likelihood (High, Medium, Low), while COBIT [90] promotes both qualitative and quantitative approaches. In this chapter we require a numerical value, which in practice can be derived from the past experiences of the assessment team or from public domain statistics.

Running example - Part 2.6. *For the purpose of our running example we estimate* i_1*,* i_2 *and* i_3 *happen (on average) respectively 5, 12 and 50 times per year. Consequently,* $\mathrm{freq}(i_1) = 5$*,* $\mathrm{freq}(i_2) = 12$ *and* $\mathrm{freq}(i_3) = 50$*.*

Now, the downtime function computed using the timed dependency graph together with the damage and the frequency evaluation allows us to compute an upper bound of the expected cost (per year) due to service downtime.

Definition 2.6 (Estimated downtime cost)**.** *Let* $g = \langle N, \rightarrow \rangle$ *be a timed dependency graph,* I *be a set of incidents,* dt *be the incident downtime mapping,* freq *be the* *frequency mapping and* dam *the damage evaluation for g. The* estimated downtime cost *for the system is defined as*

$$\mathrm{esdc}(I) = \sum_{i \in I, n \in N} \mathrm{dam}(n, \mathrm{dt}(i, n)) \cdot \mathrm{freq}(i) \tag{2.1}$$

Notice that *esdc* delivers precise results only when the following two assumptions hold: (1) incidents will not happen simultaneously, and (2) repetitions of the same incident cause an equal repetition of the same damage. Intuitively, the bigger the number of incidents and their duration, the less likely is that assumption (1) will hold, since the probability of incidents happening simultaneously increases. Should this be the case, the formula given in Definition 2.6 must be adjusted to take into account the consequence of overlapping incidents on the same node. For example, if a node is unavailable because of an incident and in the meantime another incident occurs on the same node, the damage to the organisation does not grow because of the second incident, since the node is already unavailable (because of the first incident). However, when the probability of incidents overlapping is small, the estimated downtime cost calculated with the formula of Definition 2.6 gives an upper bound of the real cost, which complies with the general risk management principle of assuming a realistic worst-case scenario in estimating impact.

Running example - Part 2.7. *Going back to our example, incident i_1 causes an yearly downtime on m_3 of 45 hours (i.e. five times a downtime of 9 hours). Similarly, incidents i_2 and i_3 cause an yearly downtime on m_3 of 36 and 100 hours respectively. Given the total number of hours in an year (8760), the probability that m_3 is unavailable because of incident i_1, i_2 and i_3 is respectively 0.005, 0.004 and 0.011. Assuming these three incidents are independent events, the probability of incidents i_1 i_2 and i_3 happening simultaneously is ~ 0.0002 (less than two hours in one year). We consider this probability to be sufficiently small to use the formula of Definition 2.6 and obtain a reasonable upper bound of the estimated downtime cost. Given the damage evaluation of p_1 and p_2 and the estimated frequency of the set of incidents $I = \{i_1, i_2, i_3\}$, the yearly estimated downtime cost of the system is 7055 Euro.*

2.3.1.2 Developing the risk mitigation strategy

The goal of risk mitigation is to bring down the estimated downtime cost by applying a set of *countermeasures*, which can be technical and organisational. To achieve full generality we define a countermeasure as a function which can modify the timed dependency graph as well as the incident repair time and the incident frequency. Each countermeasure has also a cost per year (summing the amortisation and the maintenance costs).

Definition 2.7 (Countermeasure)**.** *Let* $g = \langle N, \to \rangle$ *be a timed dependency graph,* I *be a set of incidents and* rt *and* freq *be the incident recovery time and frequency functions for* I*. A* countermeasure c*, is a pair* $\langle \text{effect}, \text{cost} \rangle$ *where* effect *maps* $g, \text{rt}, \text{freq}$ *into* $g', \text{rt}', \text{freq}'$*, and* $\text{cost} \in \mathbb{E}ur$ *is the (amortised) cost per time unit (year).*

We note that in practice most countermeasures fall into one of two classes: *frequency countermeasures* and *time countermeasures*, accordingly to the resulting effect. The former reduce the frequency of a given incident, while the latter reduce the downtime due to the incident (e.g. by reducing the incident recovery time or by increasing the survival time). In *frequency countermeasures*, the projection of *effect* on g', rt' is the identity function. It is worth noting that a countermeasure completely preventing an incident can be modelled by setting to zero either the frequency or the downtime relative to the incident.

Running example - Part 2.8. *Table 2.3 reports a list of countermeasures to be applied on* m_3 *to mitigate the negative effects of incidents* i_1*-*i_3 *(disk failure, power disruption and OS failure respectively). Notice that* c_1*-*c_7 *are technical countermeasures while* c_8 *is organisational; moreover* c_1*,* c_4*-*c_7 *are frequency countermeasures since their effect is to reduce the frequency of certain incidents, while* c_2*,* c_3 *and* c_8 *are time countermeasures since their effect is to reduce the recovery time on* m_3*. Figure 2.3 shows the propagation of incident* i_1 *after the application of* c_8*, which reduces the downtime of* m_3 *to five hours. Since the survival time of* p_2 *(eight hours) is longer than the downtime of* a_2*,* p_2 *is never disrupted by this incident, and the component relative to* p_2 *of the cost of* i_1 *is zeroed, reducing the overall estimated downtime cost.*

Table 2.3: A list of countermeasures to be applied on $m3$ to mitigate the negative effects of incidents i_1-i_3. Type F refers to frequency countermeasures, while type T refers to time countermeasures.

Id	Description	Type	Amortised Cost	I	Frequency		Recovery time	
			€/y		aft.	bef.	aft.	bef.
c_1	New disks	F	1000	i_1	3	5	9	9
c_2	UPS	T	3000	i_2	12	12	1	3
c_3	Backup machine	T	4000	I	-	-	2	-
c_4	Service pack	F	900	i_3	20	50	2	2
c_5	New OS version	F	6200	i_3	5	50	2	2
c_6	Patch #143	F	300	i_3	40	50	2	2
c_7	Patch #146	F	300	i_3	42	50	2	2
c_8	Disk backup strategy	T	2000	i_1	5	5	5	9

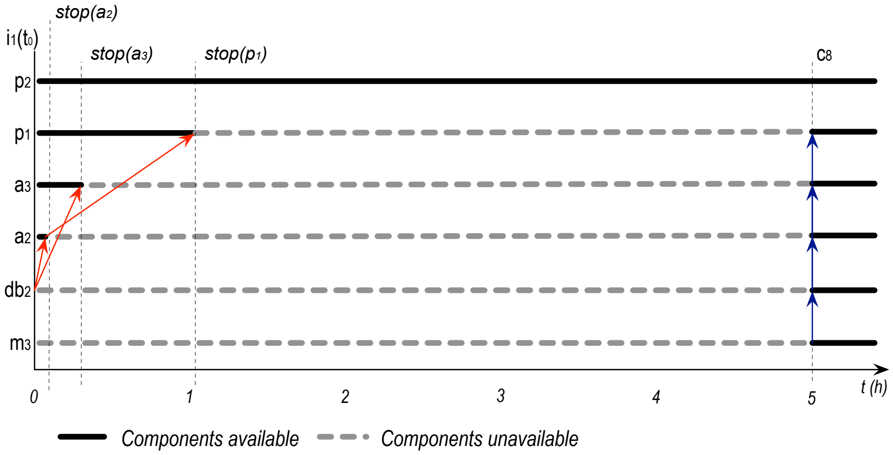


Figure 2.3: Propagation chart of incident i_1 with countermeasure c_8 in place.

It is usually possible to apply more than one countermeasure on the same node, but for this we have to consider that one countermeasure may be *incompatible* with another one. An OS patch, for example, can be incompatible with other patches; moreover, deploying a backup machine can be useless if other backup techniques are already in place.

For instance, in our Running Example, countermeasures c_4-c_7 are mutually incompatible because the service pack can not be installed if single patches are already installed, and because installing patches for the old OS version with the new version already installed would be impossible.

By combining the timed dependency graph, countermeasures and incidents with their cost and frequency, we now give a definition of *best set of countermeasures* as the set of countermeasures that reduces the estimated downtime cost the most (taking into account the cost of the countermeasures). In the definition we extend (2.1) to take into account the selected countermeasures. We also denote by $dt_C(i, n)$ the downtime the incident i causes on node n in presence of a set of countermeasures $C = \{c_1, \ldots, c_n\}$. Likewise, rt_C and $freq_C$ are respectively the recovery time and frequency functions in presence of the set of countermeasures $C = \{c_1, \ldots, c_n\}$.

Definition 2.8. *Let $g = \langle N, \rightarrow \rangle$ be a timed dependency graph, I be a set of incidents, C be a set of countermeasures,* $\text{cost}(C)$ *be the sum of the costs of all the countermeasures in C and* $\text{dt}(i, n, C)$ *and* $\text{freq}(i, C)$ *be the incident downtime and frequency functions for incidents $i \in I$ in the presence of countermeasures in C.*

- *We call* estimated global cost *of incidents I in presence of C the value:* $\mathrm{esdc}(I, C) = \sum_{i,n \in I \times N} \mathrm{dam}(n, \mathrm{dt}(i, n, C))\mathrm{freq}(i, C) + \mathrm{cost}(C)$
- *We say that* BC $\subseteq C$ *is a* best set of countermeasures *(with respect to C) if the countermeasures in* BC *are pairwise compatible, and for every* SC $\subseteq C$ *of pairwise compatible countermeasures,* $\mathrm{esdc}(I, \mathrm{BC}) \leq \mathrm{esdc}(I, \mathrm{SC})$.

Thus, the best set of countermeasures is the one minimising the expected global cost (including the cost of the countermeasures). Similarly, the *expected benefit* of a given set of countermeasures is the difference between the expected downtime cost *esdc*(I) and the expected downtime cost after applying the countermeasures: *esdc*(I, BC).

Two countermeasures are considered *pairwise compatible* if deploying one countermeasure does not make impossible or useless to deploy a second one. For example, in the set of countermeasures of Table 2.3, c_6 (Patch #143) is not pairwise compatible with c_5 (New OS version), since the patch is applicable only to the "old" OS version. Similarly, c_4 (Service pack) is not pairwise compatible with c_6, since by applying the OS service pack one fixes already the vulnerability patched by (Patch #143), thus making its deployment useless.

Running example - Part 2.9. *According to the information of Table 2.3 we can now define the incident downtime and frequency functions in the presence of countermeasures. To define the incident downtime in the presence of countermeasures, we first need to define the incident recovery time in the presence of countermeasures, on which the* dt *function is based upon.*

$$\mathrm{rt}(i, n, C) = \begin{cases} 1 & \textit{if } i = i_2 \textit{ and } n = m_3 \textit{ and } c_2 \in C \\ 2 & \textit{if } i = i_1 \textit{ and } n = m_3 \textit{ and } c_3 \in C \\ 2 & \textit{if } i = i_2 \textit{ and } n = m_3 \textit{ and } c_3 \in C \\ 2 & \textit{if } i = i_3 \textit{ and } n = m_3 \textit{ and } c_3 \in C \\ 5 & \textit{if } i = i_1 \textit{ and } n = m_3 \textit{ and } c_8 \in C \\ \mathrm{rt}(i, n) & \textit{else.} \end{cases}$$

The dt *function in the presence of countermeasures is therefore defined as:*

$$\mathrm{dt}(i, n, C) = \begin{cases} \mathrm{rt}(i, n, C) & \textit{if } n \in M \\ 0 & \textit{if } n \notin M \textit{ and } D_n = \varnothing \\ 0 & \textit{if } \max_{m \xrightarrow{s} n \in \rightarrow} \mathrm{dt}(i, m, C) - s < 0 \\ \max_{m \xrightarrow{s} n \in \rightarrow} \mathrm{dt}(i, m, C) - s & \textit{else.} \end{cases}$$

The freq *function is defined as:*

$$\mathrm{freq}(i, C) = \begin{cases} 3 & \textit{if } i = i_1 \textit{ and } c_1 \in C \\ 20 & \textit{if } i = i_3 \textit{ and } c_4 \in C \\ 5 & \textit{if } i = i_3 \textit{ and } c_5 \in C \\ 40 & \textit{if } i = i_3 \textit{ and } c_6 \in C \\ 42 & \textit{if } i = i_3 \textit{ and } c_7 \in C \\ \mathrm{freq}(i) & \textit{else.} \end{cases}$$

We can now compute esdc *considering the three incidents (i_1-i_3) and each possible combination of countermeasures (c_1-c_8). Recall that only the disruption of p_1 and p_2 involve a loss to the organisation (see Figure 2.1). The result is* BC = $\{c_1, c_4\}$*, i.e. the most cost-effective strategy to mitigate the risk is to install the OS service pack and to update m_3's disks.*

Summarising, our model provides IT managers with an effective way of determining the most cost-effective risk mitigation strategy for availability-related risks by choosing the best set of countermeasures for a given system. For space reasons, we have not addressed other optimisation possibilities which are made possible by this model, but one can use it to find for instance "the least expensive set of countermeasures which bring the expected downtime of service A down to 10 hours per year" or "the best set of countermeasures within a given budget".

2.4 Prototype implementation

A preliminary problem we had to solve when tackling the implementation issue is that of building the timed dependency graph. The information about the IT and business infrastructure is typically spread across a number of free text documents. To build the model it is necessary to report information in a structured form, such as database tables. Fortunately there exist tools for supporting IT risk assessment and business continuity which can deliver this information (previously collected through the tool) in a structured format. The Italian branch of KPMG [102] (a worldwide company delivering also Information Risk Advisory services) has developed a customisable tool, KARISMA (Kpmg Advanced RISk MAnagement), to support their RA (and business continuity) activities. KARISMA supports the risk advisory services of KPMG by providing a set of information collection forms which are filled by the owners of IT assets and business processes assisted by the KPMG personnel. The information collected by KARISMA includes: (1) a map of the organisation's IT infrastructure and the business processes and subprocesses it supports, (2) the value of the business processes to the organisation, (3) the estimate of likelihood and impact of a list of threats and vulnerabilities (derived from the knowledge base of KPMG) that could affect the IT infrastructure and (4) the current coverage and possible applicability of security controls (mainly based on the ones proposed by ISO 27002 [46]). KARISMA delivers a number of reports which summarise the current risk of the IT infrastructure based on the collected information and point out the risk exposure of each business process and subprocess.

We build the timed dependency graph by representing each entry in the database table representing IT components with a node and each entry in the tables representing links between components (e.g. applications with machines or applications with applications through the exchanged data) with an edge between nodes, annotated with the survival time.

After building the timed dependency graph we need to realise an algorithm which (a) explores the timed dependency graph to simulate the consequences of the incidents, (b) evaluates the global cost of a set of incidents, (c) simulates the new behaviour of the TD model in presence of a set of countermeasures and (d) evaluates the new global cost of the set of incidents with different subsets of countermeasures. Figure 2.4 summarises the architecture of the prototype. The task of building the TD model with the information in the KARISMA database is accomplished by the data import component. Tasks (a)-(d) are carried out by the simulation engine, the selection of the best set of countermeasures based on the TD model is carried out by the optimisation algorithm by using the output of the simulation engine.

To realise both the simulation engine and the optimisation algorithm we use in first instance model checking [24], which is a technique to algorithmically analyse concurrent systems, typically used for verifying if (a model of) the system satisfies some given properties, often specified as a model logic formula. The reason of this choice is that model checkers are already devised to explore a graph of several possible system behavioural traces, to find the one realising a given property. Therefore, model checkers provide us with a way of doing fast prototyping without sacrificing performance too much. Among the several model checkers available (e.g. SPIN [38], SMV [57], etc.) we adopt UPPAAL [53], because (1) it allows to specify a time dependent system (such as the one we need to model to accomplish (a) and (c)) and (2) its extension – UPPAAL CORA – provides a cost variable which can be used to implement (b) and (d) and provides an optimisation algorithm that allows us to solve the optimisation problem of finding the best set of countermeasures by minimising the cost.

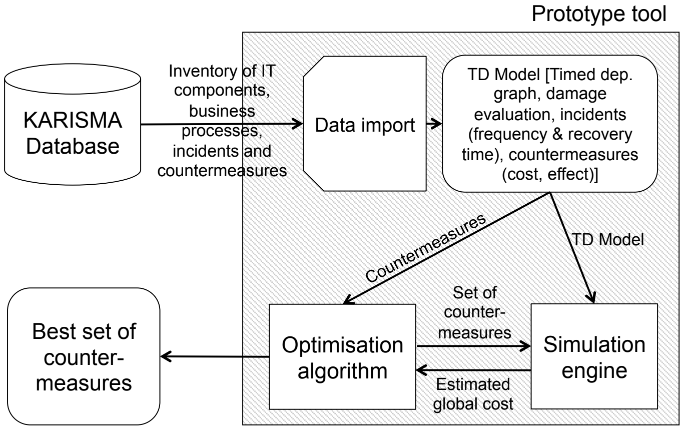

Prototype tool
KARISMA Database
Inventory of IT components, business processes, incidents and countermeasures
Data import
TD Model [Timed dep. graph, damage evaluation, incidents (frequency & recovery time), countermeasures (cost, effect)]
Countermeasures
TD Model
Set of counter-measures
Best set of counter-measures
Optimisation algorithm
Simulation engine
Estimated global cost

Figure 2.4: Architecture of our prototype. The information on the KARISMA database is imported and converted into a TD model. The optimisation algorithm is then run to determine which (sub)set of countermeasures produces the minimum estimated global cost. The estimated global cost is calculated by the simulation engine.

2.4.1 UPPAAL implementation

UPPAAL requires the system to be specified as a timed automaton [24, 14], which is a finite automaton extended with a finite set of real-valued *clocks*. Clock constraints, i.e. guards on edges, are used to restrict the behaviour of the automaton. UPPAAL CORA, is an extension of UPPAAL for cost optimal reachability analysis which applies the theory of Linearly Priced Timed Automata (LPTA) [52]. LPTA extend the model of timed automata with prices on all edges and locations. In these models, the cost of taking an edge is the price associated with it, and the price of a location gives the cost-rate applied when delaying in that location. In UPPAAL CORA prices are defined by means of an implicit monotonically growing variable called *cost*.

UPPAAL has the additional advantage of allowing us to map in a relatively natural way the main elements of our model into a timed automaton with the same behaviour. This one-to-one translation helps avoiding side effects due to the implementation.

We now show how we translate the elements of the TD model into UPPAAL automata. UPPAAL allows one to group the definition of automata with similar behaviour by means of *templates*. We build a separate template for nodes, edges, incidents and countermeasures.

Figure 2.5 shows the template of an UPPAAL automaton implementing a node of the timed dependency graph. A node can be in two states: `Up` and `Down`. The initial state is `Up`. Another automaton can cause the node to transit to state `Down` by sending a message in the shared *channel* variable `take_down`. At the same way, a node transits from state `Down` to state `Up` when a message is passed through the shared channel variable `bring_up`. When a node transits from state `Up` to `Down`, its internal clock `t` is set to zero and starts counting the downtime of the node. For each time unit that a node is disrupted, the global *cost* variable is updated with the value of the *esdc* function, which returns the damage caused on the current node by the downtime (`t`) caused by the current incident multiplied by the current incident frequency.

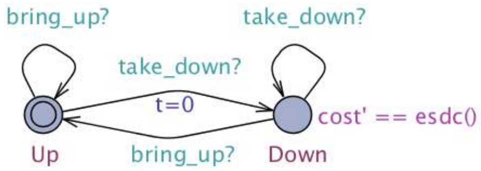


Figure 2.5: Node representation in UPPAAL CORA

Figure 2.6 shows the template of an UPPAAL automaton implementing an edge of the timed dependency graph. An edge can be in three main states: `Up`, `Down` and `Wait`. The initial state is `Up`. There are two additional states, `GoingUp` and `GoingDown`, which are so-called *committed* states (i.e. the automaton cannot delay in that state), and are used to overcome technical limitations of the model checker. The transition between the main states is regulated by means of two channel variables `src_up` and `src_down`, which correspond respectively to the `bring_up` and `take_down` variables of the source node. When the source node of an edge goes down, it causes the edge automaton to transit to the `Wait` state and delay in that state for exactly the *survival time*. Then, the edge transits to the `Down` state and sends a message to bring down the destination node through the `take_down_dst` channel variable, which corresponds to the `take_down` variable of the destination node. When the source node transits back to the `Up` state, the edge transits to the `Up` state in turn and sends a message to the destination node through the channel variable `bring_up_dst`, which corresponds to the `bring_up` channel variable of the destination node.

Figure 2.6 shows the template of an UPPAAL automaton implementing an incident. An incident can be in three main states: `NotHappened`, `Happened` and `Over`, plus an additional committed state `Resolved` introduced for technical reasons. The initial state is `NotHappened`. The boolean variable `incident_taken` is used to implement the assumption that incidents do not

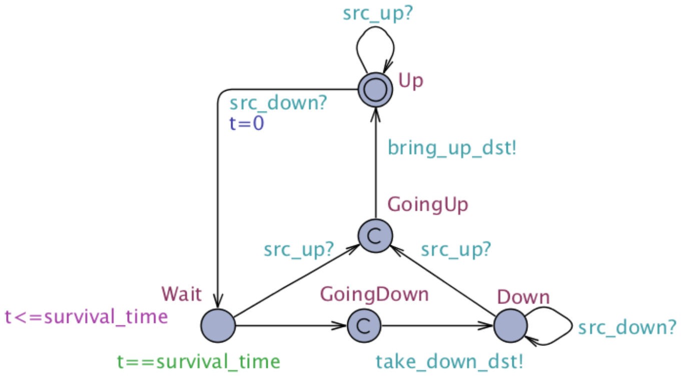


Figure 2.6: Edge representation in UPPAAL CORA

occur simultaneously. When the incident transits to state Happened, a message is sent through the shared channel variable take_down, which corresponds to the channel variable with the same name of the node automata that the incident affects. The global variables incident_freq and cur_incident are also set, with the values relative to the current incident, and will be used in the above-mentioned *esdc* function. The automaton delays in state Happened for exactly the amount of time provided by the rt variable, which corresponds to the result of the *rt* function of the TD model. When the automaton transits to state Over, a message is sent through the shared channel variable bring_up, which corresponds to the variable with the same name of the node automata affected by the incident. The value of the incident_taken and cur_incident variables is also reset.

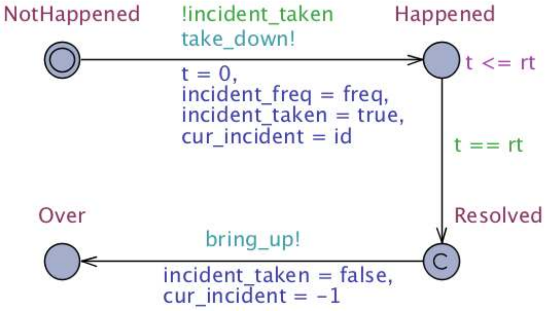


Figure 2.7: Incident representation in UPPAAL CORA

Finally, Figure 2.8 shows the template of an UPPAAL automaton implementing a countermeasure that reduces the frequency of incidents. Similar templates exist for the other types of countermeasures with the same states but different behaviour. A countermeasure can be in two main states: `NotDeployed` and `Deployed`, plus an additional committed state `IncidentFixed` introduced for technical reasons. The initial state is `NotDeployed`. A countermeasure can transit to the `Deployed` state only if the boolean variable `is_applicable` is true. This variable is associated to the countermeasure by the countermeasure `id` and is used to model the compatibility between countermeasures. When the countermeasure transits to the `Deployed` state, the global cost is updated with the yearly cost of the countermeasure, and the function *env_other_counts* is called to set to `false` the `is_applicable` variable of the other countermeasures incompatible with the current one. Once deployed, the countermeasure remains in the deployed state until the incident it is meant to prevent happens. This is modelled through a check on both the `incident_taken` variable, which tells if an incident is currently happening, and on the `i_c` variable, which tells if the current countermeasure mitigates the current incident. If this is the case, the global variable `incident_freq` is updated with the new value given by the countermeasure, and the automaton returns to the `Deployed` state.

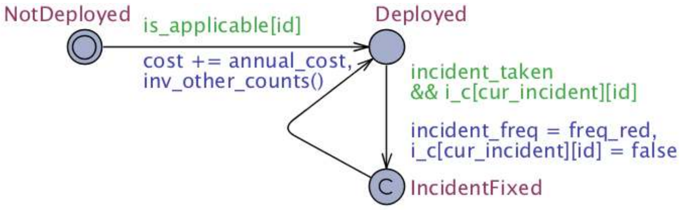


Figure 2.8: Frequency countermeasure representation in UPPAAL CORA

Once the automata templates are built (and initialised), we use the model checker to compute the best set of countermeasures. This is achieved by asking the model checker to verify the existence of a condition in which all the incidents are in state `Over` and all the nodes are in state `Up`. This condition ensures that all incidents have been completely repaired. For example, in the case of our running example, the condition would be as follows:

```
E<> i1.Over and i2.Over and i3.Over and p1.Up and p2.Up and a1.Up
    and a2.Up and a3.Up and db1.Up and db2.Up and m1.Up and m2.Up
    and m3.Up and n1.Up
```

When asked to verify such a condition, the model checker automatically computes a trace that minimises the global *cost* variable. The best set of countermeasures can thus be obtained by checking the state of the countermeasures in the final trace produced by the model checker. The best set of countermeasures is given by all the countermeasures that are in state `Deployed`.

As all model checkers, UPPAAL CORA suffers from the state-space explosion problem, which refers to the fact that the size of the state space grows exponentially in the number of components of the model to be analysed. This results in a performance degradation that makes impossible to analyse arbitrarily large models. In our case, the state space is given by the number of automata (nodes, edges, incidents and countermeasures) in the model. A further problem in our case is given by the fact that the model checker has to compute an optimisation of the global cost by selecting which countermeasures will transit to the `Deployed` state.

To test our implementation we use a dataset related to a real insurance company collected by KPMG auditors using KARISMA during a RA. The dataset contains all the information needed to build the timed dependency graph (19 macro business processes and 122 sub-processes); the remaining information (about incidents, costs and countermeasures) is also provided by the KPMG auditing team who conducted the assessment. In first instance, to avoid the state explosion problem and maintain a reasonable computational time, we perform the analysis on portions of the infrastructure, and then merge results. Each portion of the infrastructure took on average one hour to be processed. In second instance, we realise a translation of the TD model into Prolog.

2.4.2 Prolog implementation

Our Prolog algorithm implements the definitions of Section 2.3 for the timed dependency graph (sets are translated into lists), incidents, countermeasures and for the functions to calculate the estimated global cost of a set of incidents I in the presence of countermeasures C $esdc(I, C)$ (see Definition 2.8).

To compute the best set of countermeasures, we first create a brute-force algorithm which finds an optimal solution by trying all the possible combinations of countermeasures. The pseudo-code of this algorithm is shown in Algorithm 2.1. In the algorithm, I is the set of incidents and C is the set of available countermeasures.

Secondly, we develop a second algorithm which finds a partial optimal solution. The pseudo-code of this algorithm is shown in Algorithm 2.2. In this second algorithm, the best set of countermeasures is computed iteratively by adding at each iteration the countermeasure that reduces *esdc* the most. When all the countermeasures have been evaluated, or no countermeasure is left that can reduce *esdc*, the search for new countermeasures to add is considered finished and the algorithm stops. This algorithm is based on the heuristics of selecting at each it-

Algorithm 2.1 Brute-force algorithm for the selection of the best set of countermeasures

```
$best\_cost = \infty$
$BSC = \emptyset$
for all $SC \in \wp(C)$ do
    $current\_cost \leftarrow esdc(I, SC)$
    if $current\_cost < best\_cost$ then
        $current\_cost \leftarrow best\_cost$
        $BSC \leftarrow SC$
    end if
end for
```

eration the best (i.e. the most cost effective) countermeasure, given the cost and effect of the countermeasures already selected. This heuristics is based on the simple intuition that most of the times the major reduction of downtime cost is achieved by applying a set of very effective countermeasures.

In general, there is no guarantee that the solution produced by this algorithm coincides with the *global* optimum found using the brute-force algorithm. In more detail, the proposed heuristics would fail to select countermeasures which – considered in isolation – do not substantially decrease the estimated downtime cost, but work very well in combination. However, our experiments show that the partial solution is often close to the optimal one. For instance, in the case of our running example the solution found with Algorithm 2.1 coincides with the one found with Algorithm 2.2.

The Prolog implementation allows us to deal with the entire dataset at once, without splitting the IT infrastructure, and tens of incidents while maintaining the computational time in the order of minutes. We carry out optimal analysis for partitions of up to 18 countermeasures and a partial optimal analysis that can deal with thousands of them, on a 3GHz Pentium IV machine with 1Gb RAM.

2.5 Discussion

The technique we propose in this chapter to support IT managers in selecting the best strategy to mitigate availability-related IT risks is based on the TD model, which is an approximation of the real IT infrastructure, its connection to the organisation business, the incidents that could affect it and the possible countermeasures to deploy. The approximation of our TD model is based on some main assumptions which we discussed in Section 2.3 and we now summarise:

Algorithm 2.2 Algorithm for the partial optimal selection of the best set of countermeasures

```
$best\_cost = \infty$
$BSC = \emptyset$
while $C \neq \emptyset$ do
  $best\_c = NIL$
  for all $c \in C$ do
    $SC \leftarrow BSC + c$
    $current\_cost \leftarrow esdc(I, SC)$
    if $current\_cost < best\_cost$ then
      $current\_cost \leftarrow best\_cost$
      $best\_c \leftarrow c$
    end if
  end for
  if $best\_c \neq NIL$ then
    $BSC \leftarrow BSC + best\_c$
    $C \leftarrow C - best\_c$
  else
    break while
  end if
end while
```

1. a failure on an IT component propagates to other components following the functional dependencies among components in the way described by the timed dependency graph;

2. the survival time of one component with respect to the failure of another one can be reasonably approximated to an average value;

3. once the effects of incident on a set of components have been repaired, the components that are unavailable because of incident propagation become available;

4. the damage of the disruption of a business activity (process) can be reasonably approximated as a function of the process disruption time;

5. incidents do not usually overlap (i.e. the chance that two different incidents happen at the same time is negligible);

6. the effect of combined countermeasures can be estimated by IT managers.

Under these assumptions, the accuracy of the results delivered by our technique depends on the accuracy of the input data available. This means that inaccurate information (e.g. regarding the damage of the disruption of a business process, or the frequency of an incident) can mislead the selection of the best set of countermeasures. However, the selection of risk mitigation strategies is anyhow prone to this problem: IT managers are asked to take decision based on incomplete or inaccurate information. The advantage of using our model is that it provides IT managers with a framework that allow them to deal with the complex problem of addressing the dependencies among IT components and of business functions on IT in the selection of countermeasures. In this way IT managers can avoid judgement errors and better document and justify their decisions.

The other interesting point of discussion is under which circumstances it is feasible to adopt our model. We split this discussion in two parts: the first regards the possibility of collecting accurate information, the second regards the complexity of the optimisation algorithms with respect to the size of the IT infrastructure under exam.

Input Data The main concern regarding the feasibility of our approach is whether it is possible to collect accurate information required to build the TD model. The main result of our case study with KPMG is the indication that this information can be available.

Based on the experience we gained during the case study, we note that the required data is typically available after a serious quantitative risk assessment and business continuity plan.

First of all, an accurate map of the IT infrastructure is required by a business continuity plan carried out following the BS25999 [41] standard (and is also available after standard RAs). We have seen in our case-study that such information is enough to build the timed dependency graph. Most of the information required to build the timed dependency graph is also available when the IT infrastructure is modelled by an architectural framework, such as TOGAF [113], Zachman [114] and ArchiMate [84]. Indeed, the layers defined in those frameworks are similar to the ones we adopt for our model, though used for different purposes (e.g. architectural support, new component impact evaluation, etc.). Since those frameworks are widely employed (ArchiMate for instance is used within the Netherlands by ABN Amro and the Dutch Tax Office), and are supported by several tools, they provide us an indirect confirmation of the feasibility of actually obtaining the data we need.

Secondly, an inventory of possible incidents, together with their frequency *has* to be compiled during the quantitative RA (e.g. following the ISAMM standard [93]). To this end, we note that many organisations do not follow the quantitative approach because of the limited amount of quantitative data about security events. The choice of the qualitative approach is also influenced by the general business view of risk in the organisation. For instance, financial institutions (e.g. banks and insurance companies) are used to quantitatively assess financial risks as part of their core business: therefore, IT managers of financial institutions are encouraged to adopt the quantitative approach since it is well understood and used by the top management. It is common practice, and a well-known principle in RM [73], to derive this information from previous observations. We believe that one possible approach to obtain a quantitative estimate about incident frequency is to mix historical data about incidents with the expertise of the risk assessor. It is possible to gather historical data about the availability of IT systems or system components in two main ways: (1) by using publicly available information for recurrent incidents (e.g. disk breakdowns) and (2) by combining the use of so-called configuration management tools to collect the past availability of every IT component with a systematical root-cause analysis to determine the main factors causing the unavailability of (part of) the IT infrastructure after every disruption. After these observations are obtained, the final probability can be obtained by integrating them with the personal belief of the analyst by using the Bayesian update procedure suggested by Reitan in [67].

Finally, a complete evaluation of the effectiveness of chosen incident response strategies (i.e. countermeasures) is required by many standards e.g. NIST SP 800-

30 [73]: thus, the organisation is also required to quantify downtime costs of the different IT components before and after the countermeasures have been applied.

To further substantiate our argument, we note that this data is also collected by tools devised to assist the RA and RM processes. For instance, KARISMA is based on COBIT and ISO 27002, and it is very likely that other tools for RA supporting the quantitative paradigm and based on these standards would collect the same kind of information. Our system can thus be regarded as an additional component for KARISMA or for any other similar tool for RA.

Computational complexity The second concern regarding the feasibility of our approach is whether the algorithms underlying our framework are not too complex to be carried out in reasonable time. It is easy to see that evaluating the optimal set of countermeasures with the brute-force algorithm has complexity in the order of $(x \times c!)$, where x is the cost of calculating the global estimated cost of incidents in the presence of countermeasures (*esdc*). The main problematic factor in the equation is of $c!$, which indicates that the presence of a relatively large set of countermeasures would make it infeasible to carry out a brute-force analysis to find the best set of countermeasures. On the other hand, the algorithm for calculating the partial optimal set of countermeasures has (worst-case) complexity in the order of $(x \times c^2)$. Although it does not computes the optimal solution, this algorithm can be applied to the large sets of countermeasures which can be found in real-world cases.

Other ways to bring down the $c!$ would include splitting the set of countermeasures into various set of independent countermeasures, which would make it possible to apply compositional methods. However, our current implementation does not support this feature.

2.6 Related work

There exist various academic frameworks for carrying out RA, but they all differ from our proposal in that they do not model the propagation of incidents across an organisation as precisely as we do. For instance, Lenstra and Voss [55] present a quantitative approach to IT risk management to determine the optimal RM strategy given a limited budget. Their approach requires performing a risk assessment on all the applications supporting business processes and identifying the (monetary) loss due to each threat on the business process they support, thus the risk is evaluated in terms of the likelihood and the loss. Authors define an action plan (set of countermeasures) as something influencing the likelihood of a threat thus reducing the risk; furthermore they associate a cost to it. The selection of the best set of action plans consists in finding the set that mostly reduces the likelihood of all threats within a given budget. Since this approach is designed to deal with threats to all the three aspects of information security (CIA), to keep it feasible it lacks in a complete representation of the constituents of an IT infrastructure (machines, facilities, etc.) and in modelling the time dependencies between them, which - as we have discussed in the introduction - is essential for properly modelling the availability risks. Our model, on the other hand, being specifically tailored for availability risks, takes into consideration the time dependencies and therefore allows us to simulate how an incident propagates across the organisation.

Furthermore, the authors' choice of allowing a single, atomic, action plan per threat implies that the risk management team should already have found manually the best set of countermeasures to be applied in response to an incident. The proposed framework then, simply decides whether to apply or not this set of countermeasures. On the other hand, our model is able to compute the best set of countermeasures without requiring this pre-processing phase and allowing one to find a more fine-grained solution.

Asnar and Giorgini [10] introduce an extended Tropos [19] goal model to analyse risk at organisation level and to identify and enumerate relevant countermeasures for RM. Their approach is mainly devoted to the enumeration of incidents and countermeasures, while our approach focuses on selecting and prioritising incidents to be mitigated and possible countermeasures to perform the mitigation. Another proposal is that of Aagedal et al. [8], who developed the CORAS framework to produce an improved methodology for precise, unambiguous, and efficient risk analysis of security critical systems. CORAS focuses on the tight integration of viewpoint-oriented visual modelling in the RA process, using an UML-based approach in the context of security and RA. Our approach is orthogonal to CORAS, in the sense that we could use the output of CORAS to feed out tool.

In addition to academic work there exist a number of commercial tools supporting the risk management and RM process. The most closely related to our work are CounterMeasures and GSTool. Alion's CounterMeasures [83] performs risk management based on the NIST 800 series and OMB Circular A-130 USA standards. It provides the ability to perform cost/benefit analysis and ROI on countermeasures. GStool [97] is developed by Federal Office for Information Security (BSI) to assist users of the IT Baseline Protection Manual. GStool supports a qualitative assessment of protection requirements. The main difference between these approaches and ours is that they face the countermeasures selection by an economic prospective (ROI) or a technical prospective only, rather we merge the two aspects in an holistic behavioural model of the whole organisation. For a wider list of risk management supporting tools refer to [96].

Finally, our work has some analogy with some proposal for using model checking to assess the survivability of distributed systems [47, 25]. Jha and Wing [47] use the NuSMV model checker to model the distributed environment and generate a failure scenario graph (sum of counterexamples of survivability properties) by injecting faults into the model. Secondly, they add some additional information about the probability of harmful events to perform reliability analysis and cost/benefit analysis of possible countermeasures. Our approach differs in that we model also time dependencies between IT components: thus we are able to perform a more accurate evaluation of the *global impact*. Furthermore our approach is strictly focused on information risk management. Cloth and Haverkort [25] develop a model checking-based approach to evaluate the survivability of a system. Survivability is defined as the ability of a system to recover in a timely manner predefined service levels after the occurrence of a disaster. They describe the system as a Stochastic Petri net and then automatically convert it into a Continuous Time Markov Chain (CTMC). Finally they use a model checking engine to obtain a time-probability chart that expresses the recovery probability in relation to the recovery time.

2.7 Concluding remarks

In this chapter we focus on supporting the selection of mitigation strategies of risks related to the *availability* of an organisation's IT infrastructure. We argue that the way present methodologies address the time and functional relationships between the constituents of the IT infrastructure is inadequate to properly evaluate the global consequences of an incident. Our contribution consists of a model, a technique and a tool which takes into account the *global impact* of a set of risks in supporting the choice of the best set of countermeasures to cope with them. The selection process we propose complies with standard requirements for risk mitigation, i.e. it supports the selection of countermeasures which are: (1) appropriate to the business needs, (2) commensurate with the business value of assets and with the risk faced and (3) consistent and cost effective. This is achieved by employing the TD model that allows us to represent the actual propagation of an incident across the organisation and to deal with the countermeasures selection process. To this end, the presence of a tool is necessary due to the complexity of the selection process.

After the case-study carried out with data provided by KPMG, we argue that the input required by our approach can be available after a serious quantitative

RA has been carried out; this makes our proposal attractive for organisations in which the qualitative paradigm is mostly used (e.g. some financial organisations). However, we believe a more wide study could help identifying more precisely which kind of organisations could benefit from this approach.

We see two main limitations of our approach. First, since it requires the quantification of damage, incident downtimes and frequencies, it is not readily usable in organisations that carry out risk assessments following the qualitative paradigm. Secondly, although the accuracy of results depends on the accuracy of the input data, our model does not provide an indication on the uncertainty of the results accuracy. Being able to specify the uncertainty about a certain input value and obtaining results that indicate the result uncertainty could be a desirable feature for decision makers which in our opinion deserves future exploration and improvement on the TD model.

Our approach is aimed at finding the set of countermeasures minimising the expected yearly cost due to the unavailability of IT services. Here we note that a related organisation goal is that of achieving a given Recovery Time Objective (RTO), i.e. the latest point in time at which operation must resume after a failure. While this does not reduce the value of our proposal, we believe our model for incident propagation can be extended to analyse the required steps to achieve the given RTO. This topic will be further discussed in Chapter 4.

Finally, our system is particularly suited to support continuous risk management [61]: thanks to its fine granularity, it can be easily reviewed to match situational changes, allowing for early detection of service deterioration, and prompt reaction to changing environments.

Chapter 3

Model-based Qualitative Risk Assessment for Availability of IT Infrastructures*

In the previous chapter we presented the Time Dependency (TD) model, which is meant to be used for the mitigation of availability risks in a quantitative way.

Risk assessments can be carried out following two main paradigms: quantitative and qualitative. In quantitative risk assessments, risks are evaluated by means of numeric values. The magnitude of the difference between risk values is therefore known. In qualitative risk assessments risks are evaluated by means of descriptive "labels" (e.g. high, medium, low) for which only the order is known (i.e. high > medium > low). The qualitative paradigm is the one that is used the most in IT risk assessments, since it does not require numerical data regarding the likelihood of incidents and the associated monetary losses which can be difficult to obtain.

The TD model is devised for risk *mitigation* and it assumes that most of the *assessment* has already been carried out.

In this chapter, we introduce the QualTD model, which is meant to be used for risk assessments. The QualTD model extends the representation power of timed dependency graphs by including the ability to model redundancy of IT components in the infrastructure.

*This chapter is a minor revision of the paper with the same name [2] published in the journal Software and System Modelling (SOSYM), pages 1-28, Springer Berlin/Heidelberg, June 2010.

3.1 Introduction

In this chapter we focus on the following general problem: defining a technique for assessing availability-related IT risks which is simple enough to be included in a real RA, while at the same time providing solid guarantees in terms of accuracy and replicability (i.e. obtaining the same results based on the same input information) of the results it delivers.

The concrete problem that leads to the definition of the above general problem statement regards a large multinational company and the method the company uses to assess availability risks. While it is satisfied with the fact that using the present RA method they can perform RAs in time, the company aims at improving their RAs by assessing risks more accurately, and reducing the dependency of the results on the personnel carrying out the RA (i.e. when determining the impact level of a threat). At the same time, the company wants to keep the method feasible in terms of both the amount and the detail level of the information, and of the time and resources needed to carry out an RA. In other words, any improvement of their current RA method and techniques should not require information that the team carrying out the RA cannot obtain, and should ensure that the results of the RA can still be delivered timely to the requester. The natural choice to achieve these goals is to decompose the risk into its constituting factors so that the following two requirements are met:
(a) the decomposition is accurate, i.e. has a true relationship with the complex risk to be assessed;
(b) data can be collected cost-effectively.

To solve this problem, in this chapter we introduce the Qualitative Time Dependency (QualTD) model and the technique associated with it. The QualTD model and technique allow one to carry out a qualitative assessment of availability risks based on the propagation of availability incidents in an IT architecture. Incident propagation is used to increase the accuracy of incident impact estimation. Likelihood estimation is not specifically addressed by our technique, but can be based on existing likelihood estimation models (see Section 3.2 for details).

To model the assessed system we use a timed *AND* /*OR* dependency graph in which system components are represented by nodes and the functional dependencies (along with time constraints) are represented by edges between nodes. Dependencies are derived from the IT system architecture.

In order to evaluate the technique based on the QualTD model we:

1. carry out an assessment of the availability risks on the global identity and authentication management system of the company (an availability-critical system) by following the company RA method together with the QualTD

model to assess the impact of the threats and vulnerabilities present in the system, from now on we call this assessment RA_2;

2. compare the results on the impact estimation obtained from RA_2 with the results produced during a previous assessment carried out by the company using their internal RA method only, from now on RA_1 (to this end, we used the likelihood estimates from RA_1 to ensure that the results of the two RAs could be comparable);

3. identify some general factors that justify the adoption of our technique also in other cases based on the results of point (2);

4. indicate how to generalise the approach we followed in the present case to other assessments, carried out following other popular (standard) risk management methods;

5. provide a brief review of other RA techniques based on dependency graphs which we found in the literature, and we discuss the results they deliver and their applicability to the present RA case.

Our results indicate that:

1. there is evidence supporting that the technique using the QualTD model satisfies requirement (b), i.e. it is feasible to embed the QualTD model with the company's RA method without requiring too much time or unavailable information;

2. the QualTD model constitutes an improvement towards requirement (a), i.e. according to the RA team of the company, the technique using the QualTD model delivers better results in terms of accuracy (due to a more accurate impact estimation) and helps delivering more inter-subjective results (i.e. less dependent on the personnel carrying out the RA);

3. other RA techniques based on dependency graphs [12, 34, 40, 50] do not satisfy requirement (b), i.e. they could not be applied to the present case, due to the fact that they require information that is unavailable or that requires too much time to be extracted.

4. the QualTD model can be used in combination with other existing standard methods, if the target method is compatible with some key features of the QualTD model (see Section 3.5.1.2).

The last point deserves an additional explanation. A concern one has when introducing a new technique for assessing specific risks is whether this technique fits within more high level RA methods. Intuitively speaking, a general (say company-wide) RA is usually carried out following a (high-level) *method* and a number of specific *techniques*. The high-level method specifies the global lines to set up the RA process and to embed it into the organisation. Examples of RA methods include CRAMM [92], IT-Grundshutz [101], OCTAVE [82] or the NIST SP 800-30 [73]; a more complete list can be found in Section 3.5.1. The RA method usually includes a number of tasks (like evaluating the availability risks), and does not fully specify how to implement them within a specific organisation. This gives organisations the flexibility of choosing an appropriate *technique*. Techniques include Fault and Event Tree Analysis [76], Attack Graphs [71] or HazOP [22]. Our contribution can be seen as a *technique* to assess availability risks. To establish to which extent the QualTD model can be embedded in present popular RA methods, we have made a taxonomy of them and pointed out the conditions that need to be satisfied for this embedding to be successful.

The QualTD model is geared to industrial practice-compliant RAs, since: (1) it allows to link threats and vulnerabilities with the components of the IT system under assessment and derive a list of incidents and (2) it is fully qualitative and does not require numerical information which can be hard to gather. To make the QualTD model qualitative we determine the impact and risk of availability incidents when the estimates about the likelihood of threats and vulnerabilities, the incident duration and the importance of the business functions supported by the analysed IT system are expressed by values in an ordinal scale. The QualTD model also supports *AND* /*OR* dependencies to specify with more flexibility the behaviour of a component on the components it depends on when they fail.

This chapter is structured as follows: in Section 3.2 we formally present the QualTD model and we explain how it works by means of a running example. In Section 3.3 we first introduce the industrial context in which we tested the QualTD model and then we present the technique we used to apply the QualTD model to this industrial case. In Section 3.4 we describe the design, the criteria and the assumptions we made to evaluate the QualTD model and technique, and we present the evaluation results. In Section 3.5 we first discuss the applicability of our technique in combination with standard risk management methods, and then we compare our technique with other dependency-based RA techniques in the literature. Finally, in Section 3.6 we draw the conclusions of the chapter.

3.2 The Qualitative Time Dependency (QualTD) model

We now introduce the model supporting our RA technique. To illustrate the ideas we provide a running example showing how the QualTD model can be employed in practice.

The QualTD model represents the system Target of the Assessment (ToA) by means of a timed *AND* /*OR* dependency graph in which nodes can be system components, services or processes supported by the system, and dependencies among nodes are the edges of the graph. Incidents that can affect the ToA are the results of a combination of threats and vulnerabilities, and affect one or more nodes in the graph. So for example, a threat can be a Denial of Service, a vulnerability can be a buffer overflow, and an incident a Denial of Service on a specific application carried out by exploiting the buffer overflow vulnerability. The effects of an incident can propagate to another system component, service or process following the dependencies in the ToA. The model allows us to compute the global impact and the risk levels of the availability incidents hitting the ToA in the way we are about to explain. Figure 3.1 summarises the main concepts of the QualTD model: for each one of them we will provide a more detailed description in the sequel. Nodes and edges are the constituents of a timed *AND* /*OR* dependency graph. In turn, a node represents an asset constituting the IT architecture, and it is modelled as a generalisation of IT components (e.g. network components, servers, applications) and IT services or processes, which can have a certain criticality for the organisation's business. Threats can materialise on IT components (with a certain likelihood). IT components can (with a certain likelihood) have vulnerabilities. Our definitions of threats and vulnerabilities are similar to the ones given in BS 7799-3 [21]. A combination of a threat, a vulnerability on a specific set of IT components constitutes a security event (see BS 7799-3), which we call incident, and can have a certain duration. Note that BS 7799-3 defines an incident as a security event with good probability of damaging the organisation's business. According to this definition, an incident would be a combination of a threat and a vulnerability on a specific set of IT components which have a *good* likelihood and impact. For the sake of the presentation, we do not report in the diagram the concepts of incident harm and risk, as well as incident risk aggregated by threat/vulnerability, as they are complex concepts which are produced as the output of the model.

We split the presentation of the model according to the three phases of an RA the model supports: (1) definition of the ToA, (2) risk identification and (3) risk evaluation. To simplify the exposition we use the following sets to indicate domains: $\mathbb{M}$ is the set of all the time interval lengths (expressed in minutes), $\mathbb{B}$ is the set of all the possible dependency (edge) types and it is defined as $\mathbb{B} = \{AND\ , OR\ \}$, $\mathbb{D}$ is the set of all the qualitative values expressing duration

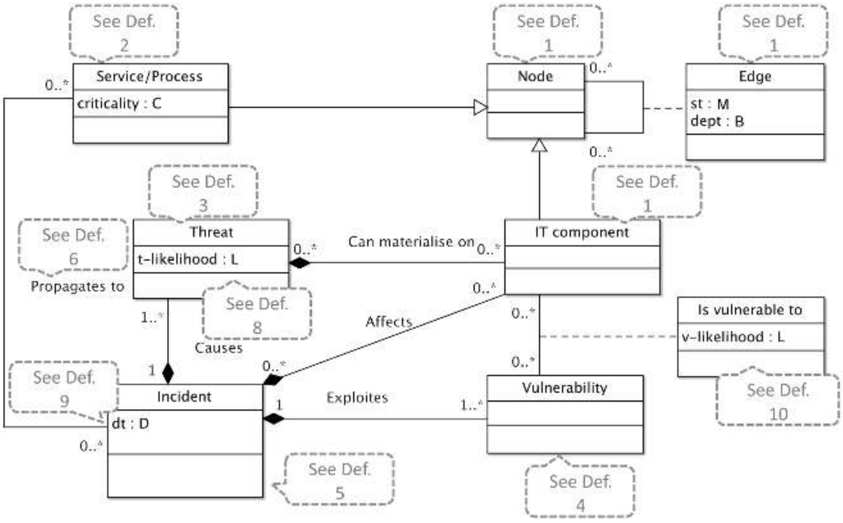


Figure 3.1: UML Class Diagram of the QualTD model. In the diagram, the type name of the attributes (criticality, likelihood, downtime, survival time, dependency type) is referred to by their initial letter only.

(e.g. `Short`, `Long`), $\mathbb{L}$ is the set of all the qualitative values expressing likelihood (e.g. `Likely`, `Unlikely`), $\mathbb{C}$ is the set of all the qualitative values expressing business value/criticality of an asset (e.g. `Critical`, `Unimportant`), $\mathbb{H}$ is the set of all the qualitative values expressing business harm (e.g. `Severe`, `Negligible`) and $\mathcal{R}$ is the set of all the qualitative values expressing the risk (e.g. `High`, `Low`).

3.2.1 Definition of the ToA

We model the ToA by means of an *AND /OR* graph which represents the components of the ToA and their functional/technical and organisational dependencies.

Definition 3.1 (Timed *AND /OR* dependency graph)**.** *A timed* AND /OR *dependency graph is a pair $\langle N, E\rangle$ where N is a set of nodes representing the constituents of the ToA, and E is a set of edges between nodes $E \subseteq \{\langle u, v, \text{dept}, \text{st}\rangle \mid u, v \in N, \text{dept} \in \mathbb{B}$ and $\text{st} \in \mathbb{M}\ \}$.*

Running example - Part 3.1. *The ToA in this example is the portion of the IT infrastructure of an organisation providing two IT services:* `eHoliday`*, the holiday reservation service for the employees of the organisation and* `CRM-Repository`*,*

the organisations Customer Relationship Management (CRM) repository service. These services are implemented by means of three applications: `WS1`*, a web server,* `DB1` *and* `DB2`*, two databases.* `DB1` *and* `DB2` *contain replicas of the CRM data, but only* `DB1` *is used by* `WS1` *as a repository for* `eHoliday`*. Applications are running on two different servers:* `Server1` *and* `Server2`*.* `eHoliday` *is implemented by* `WS1` *and* `DB1` *and, if only one of them is off-line, the service will be off-line as well.* `CRM-Repository` *is implemented by* `DB1` *and* `DB2`*, but both applications must be off-line for the service to be unavailable.* `WS1` *and* `DB1` *run on* `Server1`*, while* `DB2` *runs on* `Server2`*. According to this description, we build the timed* AND /OR *dependency graph* $g = \langle N, E \rangle$ *as follows:*
$N = \{\texttt{eHoliday}, \texttt{CRM-Repository}, \texttt{WS1}, \texttt{DB1}, \texttt{DB2}, \texttt{Server1}, \texttt{Server2}\}$
, and
$E = \{\ \langle \texttt{Server1}, \texttt{WS1}, \text{AND}\ , 0\rangle, \langle \texttt{Server1}, \texttt{DB1}, \text{AND}\ , 0\rangle,$
$\langle \texttt{Server2}, \texttt{DB2}, \text{AND}\ , 0\rangle, \langle \texttt{WS1}, \texttt{eHoliday}, \text{AND}\ , 0\rangle,$
$\langle \texttt{DB1}, \texttt{eHoliday}, \text{AND}\ , 0\rangle, \langle \texttt{DB1}, \texttt{CRM-Repository}, \text{OR}\ , 0\rangle,$
$\langle \texttt{DB2}, \texttt{CRM-Repository}, \text{OR}\ , 0\rangle\ \}$.
Figure 3.2 shows the timed AND /OR *dependency graph of this running example.*

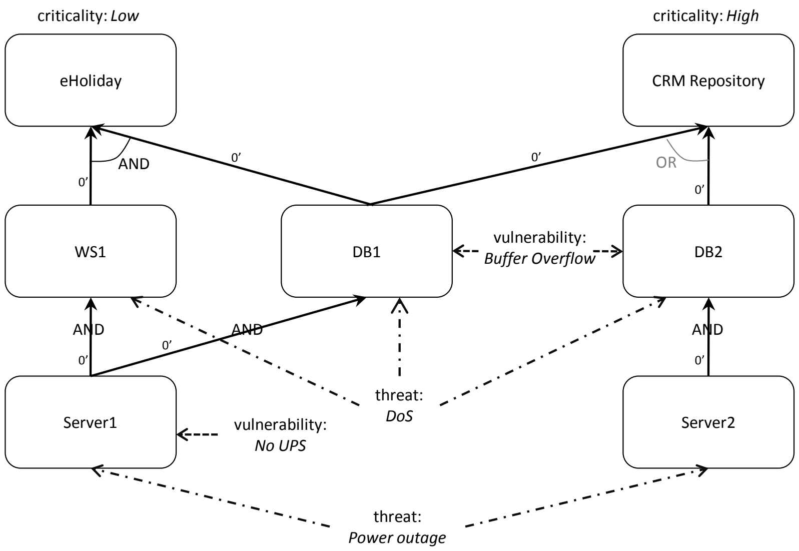


Figure 3.2: The timed *AND /OR* dependency graph representing the ToA in our running example. Nodes are the constituents of the (partial) IT infrastructure under exam. Services are annotated with their criticality level for the organisation. The figure also includes the vulnerabilities and threats which we will formally introduce later in this section and specifically describe in the running example.

The nodes N of the graph are the constituents of an IT architecture together with the business processes the IT supports, i.e. processes/services, applications, technology, infrastructure or facilities (see Section 2.3 for a more complete description). Different IT components can be represented by means of a single node in the graph, according to the abstraction level required by the RA. For example, in a company-wide assessment we could represent an IT service (i.e. a set of servers and all the applications running on them) by means of a single node, while for the assessment of a specific IT system we model each component as an individual node.

An edge from node b to node a indicates that a depends on b. The graph supports both *AND* and *OR* dependencies. In the former case this means that a becomes unavailable when any node it depends on is disrupted. In the latter case a becomes unavailable when all nodes it depends on are disrupted. Each edge is also annotated with the survival time (*st*), which indicates the amount of time v can continue to operate after u is disrupted.

If a node a has an *AND* dependency on nodes b and c and an *OR* dependency on nodes d and e at the same time, we read this as a having an *AND* dependency on nodes b, c and x, with x having an *OR* dependency on nodes d and e. Similarly, the survival time of node a with respect to nodes d and e becomes the survival time of node x with respect to d and e, and the survival time of node a with respect to x is set to zero. This concept is shown in Figure 3.3.

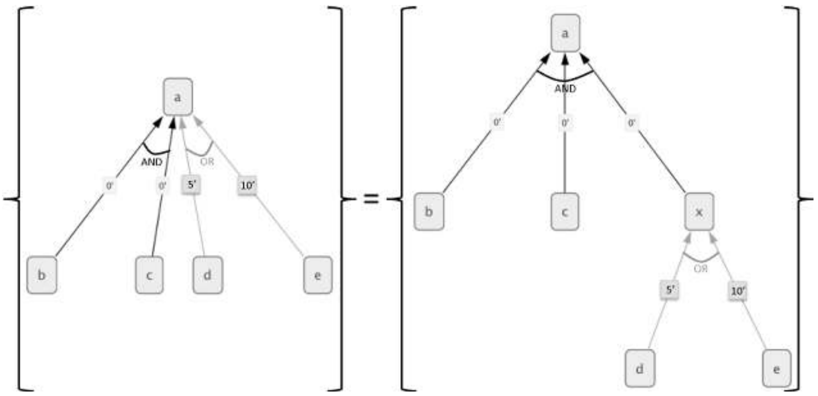


Figure 3.3: Equivalence of a graph with mixed *AND* and *OR* dependencies.

To complete the description of the ToA we include in the model an estimate of the criticality of the business processes and of the IT services in the perspective of the RA requester.

Definition 3.2 (Process/Service criticality)**.** *Given a timed* AND */*OR *dependency* *graph* $g = \langle N, E \rangle$*, the criticality of a process/service is a mapping* criticality : $N \to \mathbb{C}$ *.*

Running example - Part 3.2. *According to the business units of the organisation using the IT system, the criticality level of* `eHoliday` *and* `CRM-Repository` *is respectively* `Low` *and* `High`*.*

criticality is defined only for those nodes which represent IT services or business processes. It expresses the damage the company suffers if the node becomes unavailable. For example, in a production company, an IT service supporting a production line, which is a core business function, has a higher criticality than, e.g. personal e-mail for employees.

3.2.2 Risk identification

After modelling the ToA, we identify the vulnerabilities which are present on it, as well as the threats which could materialise on it, in particular the ones that compromise its availability.

Definition 3.3 (Threat)**.** *Given a timed* AND /OR *dependency graph* $g = \langle N, E \rangle$*, a threat is a potential cause of an incident, that may harm one or more nodes of* g*. We call* T *the set of all the threats to the ToA.*

Running example - Part 3.3. *For the sake of simplicity, here we identify two threats to the ToA: a* `Power outage` *can bring the servers off-line and a Denial of Service (*`DoS`*) attack can cause the unavailability of the applications. Our set of threats is therefore* $T = \{$`Power outage`, `DoS`$\}$*.*

This is a common definition of threat, similar to that given in BS7799-3 [21]; moreover, it is fully compatible with the concept of threat the Company has adopted in its internal RA method. The set of threats T our model addresses are only the ones which have an impact on the availability of the ToA.

Definition 3.4 (Vulnerability)**.** *Given a timed* AND /OR *dependency graph* $g = \langle N, E \rangle$*, and the set of threats* T*, a vulnerability is a weakness of a node (or group of nodes) in* N *that can be exploited by one or more threats in* T*. We call* V *the set of vulnerabilities on the ToA.*

Running example - Part 3.4. *We identify two vulnerabilities which can be present on the nodes of the ToA:* `Server1` *does not have an Uninterruptible Power Supply (UPS) unit for power continuity in case of outage; moreover,* `DB1` *and* `DB2` *may crash after a buffer overflow attack. Our set of vulnerabilities is therefore* $V = \{$`No UPS`, `Buffer overflow`$\}$*.*

Also in this case, our definition of vulnerability is consistent with both the definition given in RA standards, and with the concept of vulnerability the Company has adopted in its internal RA method.

We model an incident as a security event (as defined in BS7799-3 [21]) caused by a specific threat on a particular component of the IT architecture by exploiting a specific vulnerability. Differently from the definition of incident given in BS7799-3, we consider as incidents all security events, not only events "that have a significant probability of compromising business operations".

Definition 3.5 (Incident)**.** *Given a timed* AND /OR *dependency graph* $g = \langle N, E \rangle$*, a set of threats* T *and a set of vulnerabilities* V*, an incident* i *is a 3-uple* $\langle M, t, v \rangle$ *with* $M \subseteq N$*,* $t \in T$ *and* $v \in V$*, describing the combination of three events:*

1. v *is a vulnerability of each node* $n \in M$
2. t *is the cause of* i *on each node* $n \in M$
3. t *exploits* v

We call I *the set of all incidents generated from* g*,* T *and* V*. Moreover, we say a node* n *is directly affected by an incident* $i = \langle M, t, v \rangle$ *if* $n \in M$*.*

Running example - Part 3.5. *By combining* g*,* T *and* V *we identify four incidents that can hit the ToA: (*i_1*) A power outage causes* `Server1` *to stop because there is no UPS, (*i_2*) a DoS attack is performed on* `DB1` *by exploiting the buffer overflow vulnerability, (*i_3*) a DoS attack is performed on* `DB2` *by exploiting the buffer overflow vulnerability, and (*i_4*) a DoS attack is performed both on* `DB1` *and* `DB2` *by exploiting the buffer overflow vulnerability. Our set of incidents is therefore* $I = \{i_1, i_2, i_3, i_4\}$ *where:*
$i_1 = \langle \{\texttt{Server1}\}, \texttt{Power outage}, \texttt{No UPS} \rangle$, $i_2 = \langle \{\texttt{DB1}\}, \texttt{DoS}, \texttt{Buffer overflow} \rangle$,
$i_3 = \langle \{\texttt{DB2}\}, \texttt{DoS}, \texttt{Buffer overflow} \rangle$, $i_4 = \langle \{\texttt{DB1}, \texttt{DB2}\}, \texttt{DoS}, \texttt{Buffer overflow} \rangle$.

The last concept we introduce for risk identification is incident propagation.

Definition 3.6 (Incident propagation)**.** *Given a timed* AND /OR *dependency graph* $g = \langle N, E \rangle$ *and an incident* $i = \langle M, t, v \rangle$*, we say that* i *can propagate to a node* $n \in N$ *if:*

1. $n \in M$*, or*
2. $\exists e \in E \mid e = \langle m, n, \text{AND }, st \rangle$ *and* i *propagates to* m*, or*
3. $\forall e \in E \mid e = \langle m, n, \text{OR }, st \rangle$*,* i *propagates to* m*.*

Running example - Part 3.6. *We want to know if the incident* $i_1 = \langle\{\texttt{Server1}\}, \texttt{Power outage}, \texttt{No UPS}\rangle$ *propagates to* `eHoliday`*. Although* `eHoliday` *is not directly affected by the incident, it depends on* `WS1` *and* `DB1`*, which in turn depend on* `Server1`*.* `Server1` *is directly affected by the incident, therefore we know that* i_1 *will propagate to* `eHoliday`*.*

Definition 3.7 (Nodes affected by the propagation of an incident)**.** *Given a timed* AND /OR *dependency graph* $g = \langle N, E\rangle$ *and an incident* $i = \langle M, t, v\rangle$*,* $\text{Prop}_i = \{n \in N \mid i \text{ propagates to } n\}$*.*

Running example - Part 3.7. *According to Definition 3.7, the set of nodes affected by the incident* $i_1 = \langle\{\texttt{Server1}\}, \texttt{Power outage}, \texttt{No UPS}\rangle$ *is* $Prop_i = \{\ \texttt{Server1}, \texttt{WS1}, \texttt{DB1}, \texttt{eHoliday}\ \}$*.*

3.2.3 Risk evaluation

The last piece of information we include in the model regards likelihood and duration of incidents. In more detail, a threat is characterised by two indicators: (1) the threat likelihood and (2) the time needed to solve the disruption caused by the threat, e.g. a `Short` or `Long` disruption, or even more than two disruption lengths.

Definition 3.8 (Threat likelihood)**.** *Given the set of threats* T*, the threat likelihood is a mapping* $\text{t-likelihood} : T \to \mathbb{L}$.

Running example - Part 3.8. *Security analysts have assigned a likelihood to the threats in* T *using the following scale:* `Very Likely`*,* `Likely` *and* `Unlikely`*. The likelihood of* `Power outage` *is* `Unlikely` *and the likelihood of* `DoS` *is* `Likely`*.*

The likelihood of a threat is an estimate of the probability of the threat materialising on the ToA. Here we have made the (simplifying) assumption that the likelihood of a threat is a property of the threat itself and it is independent from the IT component the threat occurs on. The assumption holds for most of the threats, but not for targeted attacks (i.e. attacks crafted for and directed to a specific IT component), since the likelihood of the attack is influenced by the value of the targeted component. In this case we split the threat into a number of new threats, each of them representing a specific IT component being targeted.

It is common practice in qualitative RAs to assess the likelihood of threats by means of so-called likelihood models. Each model combines different parameters, e.g. difficulty of the attack, resources needed, etc. to determine the final likelihood of a threat. However, it is out of the scope of this work to specify such a model. In the literature there exist works proposing models for specific contexts (e.g. eTVRA [69] for telco networks).

Definition 3.9 (Incident duration)**.** *Given a timed* AND /OR *dependency graph* $g = \langle N, E\rangle$ *and a set of incidents* I*, the incident duration is a mapping* $\mathrm{dt} : I \times N \rightarrow \mathbb{D}$.

Running example - Part 3.9. *According to the stakeholders of the IT system, an incident is classified as a* `Long` *disruption if it takes more than 3 hours to be repaired, as a* `Short` *one otherwise. The contract signed with the power company guarantees that a power disruption is repaired on average in 6 hours. Therefore,* i_1 *is classified as a* `Long` *disruption. Since restoring* `DB1` *or* `DB2` *after they crashed only requires a restart, incidents* i_2*,* i_3 *and* i_4 *are classified as* `Short` *disruptions.*

dt(i,n) is an estimate of the (average) time a node n is out of service when incident i occurs. If we consider, for example, a buffer overflow attack which causes the stop of an application, the disruption time is the time needed to detect that the application is no longer running and to restart it. We do not take into account the time needed to fix the vulnerability exploited by the threat (e.g. the time to patch the system), unless this activity is needed to restore the functionalities of the system. To keep the model qualitative, and to match the Company method, we apply a discretisation of the disruption time in terms of `short` disruption (i.e. shorter than a given threshold) and `long` disruption (i.e. longer than a given threshold), which constitute our $\mathbb{D}$ set.

We now associate vulnerabilities with their likelihood.

Definition 3.10 (Vulnerability likelihood)**.** *Given a timed* AND /OR *dependency graph* $g = \langle N, E\rangle$*, and the set of vulnerabilities* V*, the vulnerability likelihood is a mapping* v-likelihood : $V \times \wp(N) \rightarrow \mathbb{L}$ *, where* $\wp(N)$ *is the power set of* N*.*

Running example - Part 3.10. *Security analysts have assigned a likelihood to the vulnerabilities in* V *using the following scale:* `Very Likely`*,* `Likely` *and* `Unlikely`*. The likelihood of* `No UPS` *and* `Buffer overflow` *is* `Very Likely`*.*

The *v-likelihood(v,* N_v*)* is an estimate of the probability that the vulnerability v is present in the set of homogeneous nodes N_v, i.e. nodes which can suffer from the same vulnerability with the same likelihood. The simplest and most frequent case is when we determine the likelihood of a vulnerability being present on a single node of g. However, we might also need to consider the likelihood of a vulnerability being present on a set of homogeneous nodes which are involved in a specific incident. For example, consider the case in which some malware causes a number of servers to stop working by exploiting a vulnerability which is present in an application deployed on all of these servers: in this case we need to estimate the likelihood of the vulnerability being present on all of the servers running the application with the vulnerability, since the resulting incident would affect all of them at once.

In case of an accurate RA (e.g. when it is possible to do technical vulnerability verification such as penetration testing), the fact that an application is present on an IT component can be determined without uncertainty; for example by making sure a buffer overflow affects a web server by trying to exploit it. However, in most cases, due to lack of time, the RA team has to rely on indirect (and therefore uncertain) information, for example, by consulting the NIST National Vulnerability Database [108] to check if the web server may suffer from a specific buffer overflow vulnerability. *v-likelihood* is the expression of this uncertainty.

3.2.4 Output of a RA using the QualTD model

We use the information contained in the model to calculate the risk associated with an incident, which is influenced by the likelihood that the threat occurs in the ToA (which is a property of the ToA), the likelihood that a vulnerability is present in a node or a set of nodes (which expresses the uncertainty about whether or not the vulnerability is present in the nodes) and the estimated disruption severity. In more detail, an incident causes (by propagation) a disruption with a certain duration on some nodes of the timed *AND /OR* dependency graph which have a certain criticality. We call this combination the *global impact* of the incident.

We assume that the more critical the processes/services affected and the longer the disruption, the greater the impact of the incident will be, i.e. the global impact of an incident is monotone.

Definition 3.11 (Global impact)**.** *Given a timed* AND /OR *dependency graph* $g = \langle N, E \rangle$*, an incident* $i = \langle M, v, t \rangle$*, a monotone composition function* harm : $\mathbb{C} \times \mathbb{D} \to \mathbb{H}$ *mapping criticality and duration to business harm, and a monotone aggregation function* impact-agg : $\mathbb{H} \times ... \times \mathbb{H} \to \mathbb{H}$ *; the global impact of* i *is defined by* global-impact : $I \to \mathbb{H}$ *, such that:*

$$\text{global-impact}(i) = \text{impact-agg}_{n \in \text{Prop}_i}(harm(criticality(n), \text{dt}(i, n))) \qquad (3.1)$$

Running example - Part 3.11. *The RA team has decided that the global impact of an incident is calculated using the following rules:*
a) the global impact is `Critical` *if the incident causes the disruption of at least one service with* `High` *criticality;*
b) the global impact is `Moderate` *if the incident causes a* `Long` *disruption on*

any service, or a `Short` disruption of at least a service with `Medium` criticality;
c) the impact is `Insignificant` otherwise.
For example, if we take the above definition a), the impact-agg *function is given by the "at least one service" statement, and the* harm *function is given by associating any disruption of a service with `High` criticality to the `Critical` impact. According to these rules the criticality of i_1, i_2, i_3 and i_4 is respectively: `Moderate`, `Insignificant`, `Insignificant`, `Critical`.*

Now that we have defined the incident global impact we can evaluate the incident risk, which is a composition of the likelihood of the threat, the likelihood of the vulnerability and the global impact of the disruption caused by the threat materialising.

Intuitively, this means that the more likely it is that a threat materialises on an IT component (or a set of them), or the more likely it is that the component is vulnerable to that threat, and the more harmful the threat is, the more reasons there will be to protect it against this incident. As for the global impact, also the incident risk is therefore monotone.

Definition 3.12 (Incident risk). *Given an incident $i = \langle M, t, v\rangle$, the incident risk is a monotone composition function* i-risk : $\mathbb{L} \times \mathbb{L} \times \mathbb{H} \rightarrow \mathcal{R}$ *mapping* t-likelihood(t), v-likelihood(v) *and* global-impact(i) *to the risk level of i.*

Running example - Part 3.12. *As for the global impact, the RA team has decided that the risk level of an incident is calculated using the following rules:*
a) the risk level is `High` if either the incident has a `Critical` global impact and at least `Likely` threat and vulnerability likelihood, or if the global impact is `Moderate` and threat and vulnerability likelihood are both `Very Likely`;
b) the risk level is `Medium` if either the incident has a `Critical` global impact and the threat and vulnerability likelihood are both at most `Likely`, or if the global impact is `Moderate` and either threat or vulnerability likelihood is `Very Likely`;
c) the risk level is `Low` otherwise.
In this case, i-risk *is implemented by means of these three rules, which associate the combination of global impact, threat likelihood and vulnerability likelihood to the correspondent risk level. According to these rules, the risk level of i_1, i_2, i_3 and i_4 is respectively: `Medium`, `Low`, `Low` and `High`.*

An additional operation one would like to do is to aggregate the incident risk in terms of threats and vulnerabilities. Evaluating risk in terms of threats and vulnerabilities is important to determine both the risk profile of the ToA, i.e. which threat sources are the most harmful, and to prioritise vulnerabilities to be addressed (i.e. patched) first.

Definition 3.13 (Incident risk aggregated by Threat/Vulnerability)**.** *Given a timed* AND /OR *dependency graph $g = \langle N, E \rangle$, a threat t and the set of incidents $I_t = \{i \mid i = \langle M_t, t, v_t \rangle\}$, a vulnerability v and the set of incidents $I_v = \{i \mid i = \langle M_v, t_v, v \rangle\}$ and a monotone aggregation function* risk-agg : $\mathcal{R} \times ... \times \mathcal{R} \rightarrow \mathcal{R}$ *; the risk of a threat* t *is an aggregation of the risk level of all the possible incidents which can originate from that threat (I_t), i.e. the mapping* t-risk : $\mathcal{R} \times ... \times \mathcal{R} \rightarrow \mathcal{R}$ *such that:*

$$\text{t-risk}(t) = \text{risk-agg}_{i \in I_t}(\text{i-risk}(i)) \tag{3.2}$$

Similarly, the risk of a vulnerability v is the aggregation of the risk level of all the possible incidents in which that vulnerability has been exploited (I_v), i.e. the mapping v-risk : $\mathcal{R} \times ... \times \mathcal{R} \rightarrow \mathcal{R}$ *such that:*

$$\text{v-risk}(v) = \text{risk-agg}_{i \in I_v}(\text{i-risk}(i)) \tag{3.3}$$

Running example - Part 3.13. *If we use* Max *as the aggregation function* risk-agg *to calculate the risk level aggregated by threat/vulnerability, we assign each threat-/vulnerability the maximum risk level of the incidents they are involved in. In this way, the risk level of* `Power outage` *and* `DoS` *is respectively* `Medium` *and* `High`*. Accordingly, the risk level of* `No UPS` *and* `Buffer overflow` *is respectively* `Medium` *and* `High`*.*

The QualTD model supports the traceability of the RA results. For instance, suppose the RA has been carried out, and after some time we want to recall why a DoS is a `High` *risk for our system; we can go through the records of the model and discover that:*

1. *it is* `Likely` *that a* `DoS` *is carried out by exploiting a* `Buffer overflow` *on both* `DB1` *and* `DB2`*;*
2. *both* `DB1` *and* `DB2` *are* `Very Likely` *to be prone to a* `Buffer overflow`*;*
3. *the resulting incident causes a* `Short` *disruption of the* `High` *critical service* `CRM-Repository`*;*
4. *according to points 1–3 and to the impact and risk level definitions, the risk of a* `DoS` *in the system is* `High`*.*

When doing impact and risk evaluation we use the composition and aggregation functions *harm*, *impact-agg*, *i-risk* and *risk-agg*, which operate with qualitative values (e.g. `High` likelihood and `Low` impact): the definition of the composition and aggregation functions is outside the scope of our model and it is left to the choice of the RA team. However, these functions must be monotone and semantically sound with relation to the meaning that the qualitative values involved have for the stakeholders of the RA (i.e. it should make sense given the informal meaning of the words). For example, the definition of `Critical` impact we give in the running example part 11 is semantically sound; whereas it would not have been sound if we defined as `Critical` an incident causing a `Short` disruption on a service with `Low` criticality. In the running example and in Section 3.3.2 we describe two possible implementations of *harm*, *impact-agg*, *i-risk* and *risk-agg*, based on descriptive tables which define all the possible combinations of input and output values.

Rationale for a QualTD model

It is legitimate to argue whether the model is *sound* or not. It is sound iff disruptions in the model propagate in the same way as in the real system. In principle, this could be tested by creating faults in components of the system (or of a twin test system). Regarding soundness, the system we propose has three intrinsic “limitations”: (a) it has only *AND* and *OR* nodes, (b) it does not consider the “recovery time” of the single components, and (c) it works only if the graph is acyclic. In our opinion, the first limitation is not a problem, as it is simple to model even very complicated dependencies with the use of only *AND* and *OR* nodes. The second limitation is a design choice which keeps the model simple, and in our experience does not affect the fidelity of the model. In any case, it is possible to extend our system to also take the individual recovery time into consideration, for example by assigning the recovery time to the nodes and then adding it to the incident downtime during incident propagation. The third limitation is in our opinion the only true limit of the model. Our experience says that acyclic graphs are perfectly suitable to model practical IT architecture. However, it is possible to contrive examples in which this is not the case. For such examples, either one is able to “abstract away” the cycles (for instance by analysing them separately and modelling them with a single node), or our model is simply not applicable. Once one accepts the above three intrinsic limitations, then soundness follows from the soundness of the *AND* and *OR* basic nodes: assuming that (1) the nodes of the timed *AND* /*OR* dependency graph include all the components of the ToA, and that (2) for every component the availability dependency of this component on other components is correctly and completely included in the graph by means of *AND* /*OR* edges, then the fact that an incident on a certain (set of) components will propagate in the ToA as predicted by the QualTD model can be proved by using standard graph theory. We skip the demonstration for space reasons.

It is the task of the risk assessor using the technique based on the QualTD model to make sure that hypotheses (1) and (2) are reasonably verified in a specific case. In Section 3.3 we will show the technique we used to build the timed *AND /OR* dependency graph as completely and correctly as possible.

3.3 Case-study

In this section we show how the QualTD model can be used in a practical RA by describing the case-study we carried out with it. We will also use this case-study to evaluate our technique. Let us start by describing the context in which it was carried out.

3.3.1 The industrial context

The organisation We carried out the case-study at a large multinational company with a global presence in over 50 countries (from now on we call it the Company) counting between 100.000 and 200.000 employees. The Company IT unit supports the business of hundreds of internal departments by offering thousands of applications accessed by approximately 100.000 employee workstations and by many hundreds of business partners. The IT facilities for the European branch are located at one site: our RA was conducted at that site. IT services are planned, designed, developed and managed at the Company's headquarters; those services, such as e-mail or ERP systems, are part of the IT infrastructure which is used by all the different Company's branches all over Europe.

The stakeholders of the IT service are: (1) the Company's Global IT Infrastructure (GIT) management department, (2) the Risk Management and Compliance (RMC) department, (3) users: the Company's units using IT services (including GIT and RMC) and (4) an outsourcing company managing parts of the IT infrastructure on behalf of GIT.

GIT provides basic IT infrastructure services such as desktop management, e-mail and identity management. IT services are designed internally by GIT and then partly outsourced for implementation and management to another company. The outsourced tasks include specialised coding, server management, help-desk and problem solving services.

RMC supports the compliance to internal policies and best practices of the Company IT services; part of the tasks of RMC is to perform on-demand security RAs for the IT services of GIT. An RA is usually requested by the owner of the IT service each time a new service is developed or a new release of an existing one is about to be deployed.

The other business units of the Company rely on these IT services for the continuity of their business. Some of these IT services are developed and managed by the business unit itself (e.g. if they are specific to the competence area of the unit), while global company services (e.g. authentication, e-mail system) are provided by GIT. For efficiency reasons, like in most other large organisations, business units exchange services by means of a "enterprise internal market": one business unit pays another one for the use of a given service and the service provider unit finances its activities by means of these funds. This mechanism increases the efficiency of internal service management.

The implementation and the management of some IT services are outsourced to another company, which we call the Service Provider. Although the servers running the IT services are owned by the Company and physically kept within its data centres, the Service Provider manages the OS and the software running on them. Moreover, for some services, the Company outsources also the development (e.g. coding, deployment) of the custom applications to the Service Provider. The Service Provider has signed contracts with the Company which include Service Level Agreements (SLAs) regarding both the security of the information managed by the outsourcing company and the availability of the outsourced services.

The target of assessment The system on which we focus our case-study is called Oxygen. Oxygen is the global Identity Management for employees and sub-contractors of the Company. The goals of the system are:

1. *Identity Management*: to provide enterprise-wide standard identities for all employees and contractors of the Company, integrate identities with the different identity authoritative sources (e.g. the Human Resources information system) and manage them through a governed process and ensure regulatory and privacy compliance.

2. *Identity/Account Linking and data synchronisation*: to provide a holistic view of the many accounts possessed by a person, enforce account termination when a person leaves the Company, enable data synchronisation among identity provider and identity consuming systems for data accuracy and provide credential mapping, a foundation for Single Sign-On.

3. *Identity Service for authentication and authorisation*: to provide operational directory services for general applications to be used for authentication and authorisation, to provide unique, standard, organisation-wide identifiers for employees and contractors, and to provide a foundation for advanced authentication and authorisation in the future.

Oxygen is designed and implemented by the GIT department, while the management of the servers running it is outsourced to the Service Provider.

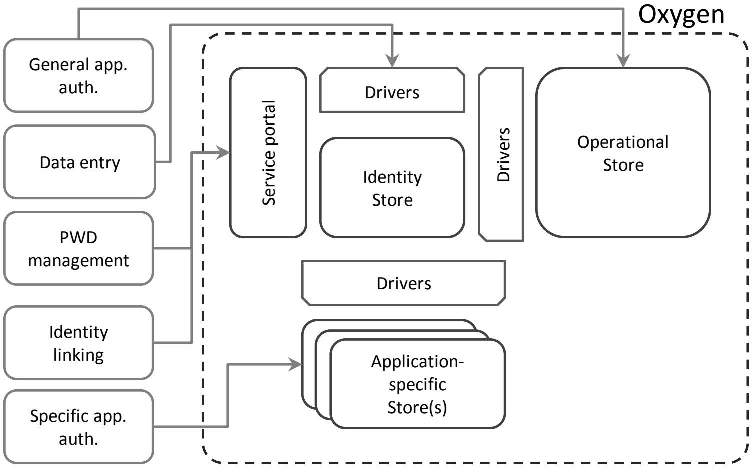


Figure 3.4: An overview of the Oxygen architecture

Figure 3.4 depicts the design of Oxygen: the system is composed of a number of *identity stores*, which are identity databases implemented by means of directory services. The main Identity Store keeps information about all of the identities and their attributes. The Operational and the Application-specific stores contain a (partial) replica of this information and are accessed by the different applications which require identities for authentication and identification. Replication of the identity stores is required for performance reasons.

Oxygen collects identity data from different authoritative sources, such as the information system of the Human Resources department. Data acquisition is performed by means of drivers, which also take care of synchronising data between the different identity stores.

In addition to the identity stores, Oxygen exports also a service portal, which allows employees of the Company to manage part of their identity record (e.g. updating their home address, changing password).

The existing RA method In 2008, the RMC department carried out an RA on the Oxygen system following its internal RA process, which is mainly based on the guidelines provided by BS7799-3 [21], while the official security control policy is compliant with the ISO 27002 [46] standard.

The upper part of Figure 3.5 depicts the process usually followed by RMC. In the following list we describe the 6 tasks composing the RMC process and we link them with the steps of the QualTD technique.

1. *RA intake*: the RA team (composed of people from the RMC department)

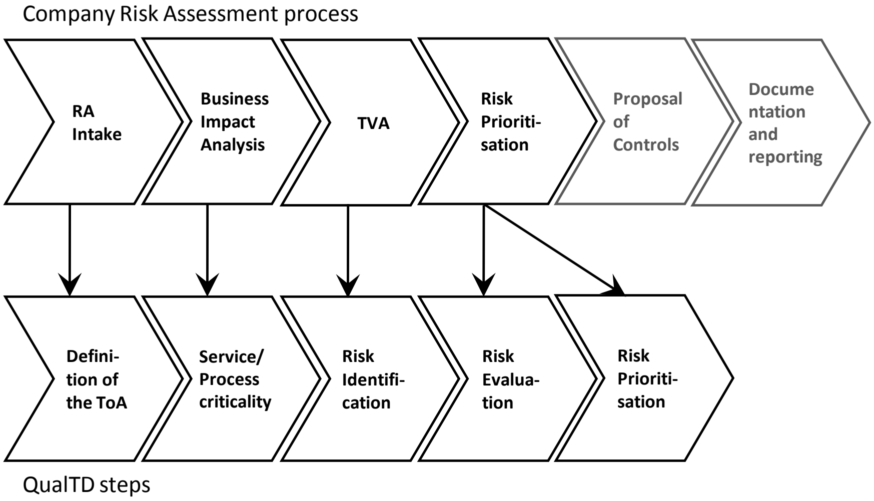


Figure 3.5: The internal RA process (above) linked to the steps of the QualTD technique which complement the process (below)

and the requester project responsible agree on the scope of the RA and the Target of Assessment (ToA). The requester also submits proper documentation about the IT service to the RA team. This task corresponds to the definition of the ToA (see Section 3.2.1) in the QualTD technique.

2. *Business Impact Analysis (BIA)*: the RA team, together with the owner of the ToA, determines the desired levels of Confidentiality, Integrity and Availability for the ToA (e.g. HIGH integrity and availability and LOW confidentiality). They do this by analysing the impact that a breach of one of the three security properties on the information managed by the ToA would have on the business unit in a realistic worst-case scenario. They also determine which legislation or regulation requirements the ToA has to comply with (e.g. SOX [112] compliance). During this task the definition of the service/process criticality in the QualTD technique (see Section 3.2.1) should be made.

3. *Threat/Vulnerability Assessment (TVA)*: the RA team analyses the ToA and determines which threats/vulnerabilities the ToA is exposed to. Risk identification is based on a fixed list of threats/vulnerabilities which has been derived from a number of existing RA standards (e.g. BS7799-3, ISO 17799, BSI IT-Grundshutz [21, 44, 101]) and customised to fit the needs of the Company. The BIA influences the TVA in the sense that the threat list is customised according to the required levels of confidentiality, integrity

and availability of the ToA: the higher the security level, the more detailed the list. The list is then used to check if the main components of the ToA (e.g. network communication, user interface, etc.) are exposed to the threats/vulnerabilities. At this stage, threats/vulnerabilities are flagged as applicable/not applicable to the considered component of the ToA, and as covered/not covered according to the fact that controls that could mitigate them are already deployed. This task corresponds to the risk identification step (see Section 3.2.2) in the QualTD technique.

4. *Risk prioritisation*: it consists in the evaluation of likelihood and impact of the threats/vulnerabilities which have been marked as applicable and not covered during the TVA. The risk assessors estimates the likelihood of a threat/vulnerability based on the company likelihood model, which takes into account several factors, e.g. resources, technical skills and time needed, or attacker motivation. They estimate the impact of a threat/vulnerability, based on the possible incident scenarios that the threat/vulnerability could determine in the ToA. These scenarios are figured out by the RA team based on their personal skills and their knowledge of the ToA. Likelihood and impact are then combined to determine the resulting risk, based on a risk aggregation matrix very similar to the one of Table 3.2. Threats and vulnerabilities are then prioritised based on their risk level: the higher the risk, the higher the priority for controls. This task corresponds to the risk evaluation and to the output of the QualTD technique steps (see Section 3.2.3 and Section 3.2.4).

5. *Proposal of Controls*: the RA team proposes a plan to cope with the identified risks, and identifies controls to mitigate the likelihood of the threats or to protect the ToA from the identified vulnerabilities. Examples of proposed controls include password policies, authentication mechanisms or Intrusion Detection/Prevention Systems.

6. *Documentation and reporting*: the RA team presents the results of the RA to the requester. It is not mandatory for the requester to communicate with the RA team about follow-up actions taken as a consequence of the RA.

The average time needed for an RA is approximately 240 man-hours (2 people for 3 weeks), depending on the size of the ToA (usually, RMC carries out RAs on ToAs which are comparable in size with Oxygen). Roughly, the first 80 man-hours are spent on steps 1 and 2 and for reading all the relevant documentation, another 80 man-hours are spent in steps 3 and 4, and the remaining 80 man-hours are spent in step 5 and to prepare the final report to be exposed during step 6. The

RA team consists of two people performing the same task independently and then peer-reviewing each other's findings to come to a more objective final result.

The RA team uses three main sources of information: (a) documentation provided by the requester, (b) interviews with the requester and (c) vulnerability scans and other forms of direct investigation of security weaknesses.

Documentation includes results from previous assessments (i.e. RAs and security auditing activities), all the design and development documents (i.e. functional specifications, security design, technical architecture design and software design) and SLAs and outsourcing contracts.

Interviews with the requester are carried out after reading the documentation to clarify doubts and to set the boundaries of the RA. Another interview is carried out to address the BIA and, after step 4, to discuss about the main risks identified.

Optionally, the RA includes active forms of investigation of security weakness. The general principle RMC follows is *trust but verify*, which means that documentation about security measures implemented is trusted, but verified in its main aspects by means of, for example, vulnerability scanners.

3.3.2 Availability RA using the QualTD model

In this section we describe how we employed the QualTD model together with the RA method of the Company for the new RA of Oxygen. The main difference of a RA carried out following the Company internal RA process only with one carried out following our technique is that we build a timed *AND* /*OR* dependency graph of the ToA and link threats and vulnerabilities with each other and with the nodes of the graph to better estimate impacts. As we discuss in more detail in Section 3.4, we used likelihood estimates carried out by the Company RMC personnel, since the QualTD model does not specifically address this topic.

We combined the QualTD model with four tasks of the Company internal RA process, as we show in the lower part of Figure 3.5. First, we included in the *RA Intake* the activity of building the timed *AND* /*OR* dependency graph. We spent 80 man-hours to perform this task. We also re-performed part of the BIA: instead of only defining the security requirements for Confidentiality, Integrity and Availability, we also assessed the criticality level of the main IT services of the ToA. We spent one man-hour on this. Finally, we carried out the *Threat Vulnerability Analysis* and *Risk prioritisation* by using the QualTD model as we explained in Section 3.2. We spent 72 man-hours to perform this task.

To build and run the QualTD model for Oxygen we relied on two sources of information: technical documentation and interview sessions. In practice we used the same documentation the RA requester provided for RA_1, as we describe in Section 3.3.1. In more detail, four documents were made available for the RA:

1. *The functional specification document*: this document describes the functionalities provided by Oxygen and how the functional architecture is designed, i.e. software components, what is their task and how they relate to each other.

2. *The security architecture and design document*: this document describes which security measures are implemented, e.g. server redundancy, and how they are implemented, e.g. which services are redundant and where they are located.

3. *The internal SLA document*: this document describes the quality of service parameters which are guaranteed to the users of Oxygen. In the context of availability, this document describes the availability figures for the different services provided by Oxygen, e.g. the authentication service is guaranteed to be available 99% of the times.

4. *The network diagram*: this document describes which are the actual servers running the different components of the Oxygen system, which software they are running and in which datacenter they are being managed.

We now describe in detail the activities we performed. For the sake of exposition we split the description according to the tasks that compose the Company RA process. Each task is further split according to the related step of the QualTD model of Section 3.2, as shown in Figure 3.5.

3.3.2.1 RA Intake

Defining the ToA The first step is building the timed *AND /OR* dependency graph for Oxygen. According to the level of abstraction required for this RA, we modelled the following node types:

1. *Datacenters*: from the security architecture document and the network diagram we extracted the two buildings hosting the datacenters in which the servers are split for redundancy purposes.

2. *Network components*: from the security architecture document and the network diagram we extracted the firewalls protecting the different servers and enabling access to the Oxygen services from the internal network.

3. *Servers*: from the security architecture and the network diagram we extracted which servers are used.

4. *Applications*: from the security architecture, the network diagram and the functional specification documents we extracted the applications running on each server.

5. *IT Services*: from the functional specification and the internal SLA document we extracted the services exported by Oxygen, linking them to the applications implementing them.

The most challenging task in building the timed *AND /OR* dependency graph was determining the dependencies among the nodes. The dependencies among buildings, network components, servers and applications could be inferred from the network diagram and the security architecture. Unfortunately, the functional specification document, which should link software to IT services, only referred to "logical" software components, which are not directly linked to the servers and the applications running on them. For instance, the functional component which acquires identity information from the different authoritative identity sources is actually implemented by three different applications: a Java-based web service, a Directory service and a DBMS; in turn, the DBMS also supports other functional components. To determine these dependencies we proceeded by refinement: whenever in the documentation we found that a certain application runs on a certain server, or that the application implements a certain service, we drew a new dependency among these nodes. Then, we cross checked the information from the functional specification and the network diagram documents to make sure the dependencies we found were consistent throughout all the documents. When we found an inconsistency, we updated the model and iterated the process. We reached a "stable" version of the model after the third iteration of this process.

To support this step we developed a graphical tool. The tool allowed us to draw the timed *AND /OR* dependency graph, show it and modify it quickly during the interview sessions. The resulting graph is made of 65 nodes and 112 edges. Among the nodes we count 13 IT services, 32 applications, 14 servers equally distributed between 2 datacenters and connected simultaneously to 2 different network segments by means of 2 different firewalls. Building the first prototype version of the graph took us approximately 40 man-hours, using only the four documents we described as a source.

After building this prototype version of the timed *AND /OR* dependency graph we checked it with the RMC personnel during an interview session: we showed the graph and explained the reasons motivating each dependency drawn; we then asked for possible missing ones. For example, we showed that a failure in the

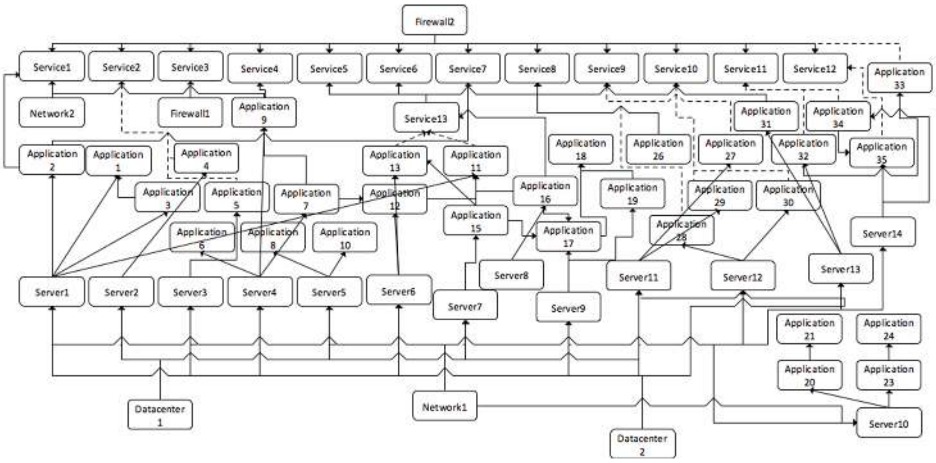


Figure 3.6: This timed *AND /OR* dependency graph resembles the one actually built for Oxygen. We observe from the bottom: datacenters, network components, servers, applications and IT services. Solid edges are *AND* dependencies, while dashed edges are *OR* dependencies.

DBMS would lead to the unavailability of the identity data acquisition service and we asked if this conclusion was consistent with their knowledge of the system. The answer was positive; no inconsistencies were found during this session. Finally, we performed another interview session with the developers of the system to further check for consistency and completeness of the timed *AND /OR* dependency graph. During this session we focused our explanation of the graph on the reasons motivating the choice of modelling a dependency between two nodes. For example, we motivated the choice of drawing a dependency from the DBMS to the application server since the Web Service uses the DBMS to store configuration parameters, and the unavailability of the DBMS would cause the Web Service to be unable to operate in turn. We found some discrepancies between our model and the behaviour of the system which is currently implemented. These discrepancies were due to inaccurate or outdated information in the functional specification document: we decided to keep the graph coherent with the actual implementation of Oxygen, instead of the one present in the documentation. RA_1 did not spot these discrepancies, as the analysis of the ToA required to build the timed *AND /OR* dependency graph is much more detailed than the analysis required for an assessment which does not require to build any formal model.

Figure 3.6 shows an anonymised version of the timed *AND /OR* dependency graph we obtained at the end of this task. During the task, although we did not know anything about Oxygen before our RA, we were able to build the timed

AND /OR dependency graph based on the available documentation. We only relied on interviews to confirm the correctness of the graph, not to build the graph itself. This ensures the method can be used by any risk assessor, who must not be an expert of the ToA.

3.3.2.2 Business Impact Analysis

After we built the timed *AND /OR* dependency graph, we considered the Business Impact Analysis (BIA), which results in determining the required level of availability for the whole Oxygen system and the criticality level of all the IT services exported by Oxygen. We did this by interviewing the GIT department board, together with a member of the RMC department.

Since the required level of availability for Oxygen had already been assessed during RA_1, we only made sure that that part of the BIA was still valid. The GIT personnel confirmed that Oxygen requires a `High` level of availability. We then used this parameter during the risk identification phase for the selection of the threats and vulnerabilities to be used, as we describe in Section 3.3.2.3.

The new step of the BIA required by the QualTD model, which is not part of the RA method of the Company, consists of assessing the criticality of the IT services. For each IT service in the timed *AND /OR* dependency graph we asked the GIT personnel if it had a `High`, `Medium` or `Low` criticality. In this way we defined the *criticality* function (see Definition 3.2).

After this last interview we had a final (approved) version of the timed *AND /OR* dependency graph representing the ToA.

3.3.2.3 Threat/Vulnerability Analysis

Risk identification Recall that the RMC department adopted a threat/vulnerability list for their RAs, which was extracted from a number of standard RA methods and customised to fit the needs of the Company. To be able to compare the results of RA_2 with RA_1 we used the same threats and vulnerabilities. We will describe in more detail the reasons why we chose to do this in Section 3.4.

The list comprises a total of 121 threats and vulnerabilities. Since we only assess availability risks, we selected the subset of this list with an impact on availability, relying on the classification done by the RMC which determines for each entry if it has an impact on confidentiality, integrity or availability. Consequently, the set T was composed of 22 threats and the set V of 39 vulnerabilities. Moreover, according to the Company RA method, threats and vulnerabilities are selected based on the required level of Confidentiality, Integrity or Availability for the ToA. Since the level of availability of Oxygen has not changed in the two RAs we are allowed to use the same availability threats and vulnerabilities.

The next step we carried out was to link threats with vulnerabilities. During RA_1 threats and vulnerabilities were assessed separately, while the QualTD model requires us to link threats with vulnerabilities (thereby making explicit the reasoning that was implicitly done during RA_1). We did this by selecting, for each of the 22 threats, which one of the 39 vulnerabilities the threat can exploit to materialise. To validate our threat-vulnerability mapping we explained our choices to the RMC personnel during an interview session and we integrated our mapping based on their opinion. Although no major inconsistency was found, we had to change a small number of mappings, because of a misinterpretation of the description of some threats.

Subsequently, we determined which nodes of the timed *AND /OR* dependency graph were targeted by threats and in which nodes a certain vulnerability was present. To do this we evaluated which kind of node the threat/vulnerability applies to; for example, a power disruption can only affect a datacenter, a DoS attack can only affect software nodes.

Finally, we enumerated the availability incidents following Definition 3.5. This task was performed automatically by intersecting threats with the nodes they target, vulnerabilities with the nodes they are present in and threats with the vulnerabilities they can exploit. We inserted all this information in a database. Therefore, listing incidents was nothing more than building a view on the existing table schema. We checked our results with the RMC personnel, to detect inconsistencies in our mapping, but we found no discrepancy, as mapping threats and vulnerabilities to asset types was quite an unambiguous task.

3.3.2.4 Risk prioritisation

Risk evaluation We used the estimates of the likelihood of threats and vulnerabilities from RA_1, (for the definition of the *t-likelihood* and *v-likelihood* functions see Definition 3.8 and Definition 3.10). The estimate was done in terms of `High`, `Medium` and `Low` likelihood level, according to the likelihood model adopted by the RMC team, which is based on eight different parameters (e.g. time needed for the attacker, technical skills needed, etc.). The reason why we did not do our own estimate of the likelihood is twofold: first, we needed to ensure that the results of the two RAs could be comparable and, since our model only implies a different way in estimating the impact, likelihood had to be kept fixed. Second, since the results of this second RA are meant to be used by GIT, we wanted the likelihood estimates to be based on the professional judgement of the RMC personnel, instead of ours.

To assess incident duration (i.e. the dt function of Definition 3.9) we first used the Company-internal SLAs to set the threshold between a `Short` and `Long` incident duration. The Company-internal SLAs give an availability figure for the IT services provided by Oxygen. For example, they guarantee that the identity data acquisition service will be available for a certain fraction of time in a month. We set the threshold as the longest amount of time (in hours) the service can be out of service while remaining compliant with its SLA. For example, if the availability figure is 99.5% in a month (i.e. 30 days), we set ~ 4 hours as our threshold. We choose this measure since, in this case, the SLAs were set to give an indication about how long a certain service can be disrupted without causing excessive problems to the Company's business. In this way we distinguished between `Short` incidents (i.e. those shorter than the maximum tolerated disruption time in a month) and `Long` ones (i.e. those which last longer than the maximum tolerated disruption time in a month). Subsequently, we analysed the time needed to solve each of the incidents. We considered both the time needed to detect the disruption and the time needed to fix the problem. The resulting total disruption time, which we compared with the threshold, is the sum of these two parameters. We performed this analysis based on both the information we gained from the SLA document the Company has signed with the outsourcer, and the opinion of the developers of the Oxygen system. The SLA document contains the maximum response time for incidents happening in the portions of the system for which management has been outsourced. For all the remaining parts of the system we relied on the judgement of the GIT developers.

With this we had acquired all the information needed to run the model and obtained the *global impact* of the incidents and their risk. For each incident i we used the timed *AND /OR* dependency graph to determine the set $Prop_i$ of the processes and services which were affected by the incident given the IT components the incident directly targets as we described in Definition 3.7. Subsequently, we used Table 3.1 to determine the global impact level. The definitions we used are based on the requirements for availability the GIT has set on Oxygen during the meeting in which we assessed the criticality of services/processes. These definitions are an implementation of the combination of the composition function *harm* and the aggregation function *impact-agg* of Definition 3.11.

We then used the definitions of Table 3.2 to determine the risk level associated with every incident. The definition of the risk level we give was built on the indications of the RMC personnel and it is an implementation of the function *i-risk* of Definition 3.12.

The choice of using these two tables to evaluate the global impact and the risk level was driven by two main motivations: first, the functions defined by the tables are monotone, therefore they are compliant with the requirements of Def-

Table 3.1: Global impact level determination.

Impact level	*Definition*
`Critical`	At least one service/process with `High` criticality is disrupted for a `Long` period of time.
`Serious`	At least one service/process with `High` criticality is disrupted for a `Short` period of time.
`Significant`	At least one service/process with `Medium` criticality is disrupted for a `Long` period of time.
`Moderate`	At least one service/process with `Medium` criticality is disrupted for a `Short` period of time.
`Marginal`	At least one service/process with `Low` criticality is disrupted for a `Long` period of time.
`Insignificant`	No service/process is disrupted or only service/process with `Low` criticality are disrupted for a `Short` period of time.

Table 3.2: Incident risk level determination.

Risk level	*Definition*
`High`	Impact is `Critical`, both threat and vulnerability likelihood are `Medium`. Impact is `Serious`, both threat and vulnerability likelihood are `High`.
`Med-High`	Impact is `Critical`, either threat or vulnerability likelihood is `Low`. Impact is `Serious`, both threat and vulnerability likelihood are `Medium`. Impact is `Significant`, both threat and vulnerability likelihood are `High`.
`Med`	Impact is `Serious`, either threat or vulnerability likelihood is `Low`. Impact is `Significant`, both threat and vulnerability likelihoods are `Medium`. Impact is `Moderate`, either threat or vulnerability likelihood is `High`.
`Med-Low`	Impact is `Significant`, either threat or vulnerability likelihood is `Low`. Impact is `Moderate`, both threat and vulnerability likelihood are `Medium`. Impact is `Marginal`, both threat and vulnerability likelihoods are `High`.
`Low`	In other cases.

inition 3.11 and Definition 3.12, and they allow one to trace back the reasons causing the assignment of a certain risk level to a certain incident (see Running example 3.13). Secondly, the alternative choice of assigning a numerical value to each qualitative one (e.g. `High` = 3, `Med` = 2 and `Low` = 1) and then perform mathematical operations on them (e.g. sum, multiplication or average) would not work in our case. In fact, although this is a very popular and widely adopted technique

in RAs (e.g. see Cunningham et al. [27]), it only provides meaningful results if we know the exact *ratio* among the qualitative values (e.g. if we knew that `High` is exactly three times `Medium` we could assign 9 to `High` and 3 to `Medium`). Since our RA was carried out in a completely qualitative manner, we only know that `High` is bigger than `Medium`, but we do not have any indication on how big the *ratio* is between them, therefore, we cannot perform any mathematical operation on these values. In other words, we work with values in an *Ordinal scale*, while the other approach would at least require values in an *Interval scale*, as shown by Herrmann [37].

Having determined the risk level, we ranked availability incidents according to their risk. However, to complete the outcome of the threat/vulnerability assessment step, we also needed to rank the most dangerous threats and vulnerabilities for Oxygen. We did this by assigning each threat/vulnerability the risk of the incident they cause, which has the highest level associated. In other words, we used *max* as the aggregation function *risk-agg* of Definition 3.13.

3.4 Case-study evaluation

In this section we make an evaluation of our case-study. To this end, the methodology we follow is the one introduced by Wieringa et al. [77, 78] for technical research, which is based on the following two statements:

1. `solution` *&* `context` *produces* `effects`

2. `effects` *satisfy (to an acceptable extent)* `stakeholder-motivated criteria`

Wieringa et al. observe that each technological solution which is applied in a context produces some effects on it. The effects may (or may not) contribute to satisfy some goals defined by the stakeholders of the research context. The evaluation criteria set by the stakeholders must be in a measurable or comparable form, so that if two different solutions are applied to the same context, they can be evaluated and compared with relation to these criteria. The reasoning scheme can be applied when a solution is specified but not yet implemented [36] or after a solution is implemented [66].

In our case, the technical solutions to be evaluated are the RAs performed on the Oxygen system: the first is done following the RA method of the Company and the second made by integrating the same method with the QualTD model. The context in which we apply these solutions is described in Section 3.3.1.

3.4.1 Stakeholders, goals and criteria

First, we present the stakeholder's goals and the derived evaluation criteria, which we have already briefly introduced in Section 3.3.1. These goals regard both specifically (the security of) Oxygen and the quality of the general RA process. Methodologically, we derive the goals by analysing the description of the activities GIT provided us during the interviews; subsequently we defined the criteria to measure those goals. Finally, we validated the goals and criteria by means of interviews with the stakeholders. For the sake of the presentation we only report the results of this activity in the list below. Although our case-study will not allow us to evaluate all the criteria, we report them all to give an overview of stakeholder's objectives.

Goals and criteria regarding Oxygen:

- *GIT*

  1.1 The goal *Ensure cost/effective mitigation controls and timely mitigation plans* is measured by the quality criterion *Cost for managing High/Medium/Low risks*.

  1.2 The goal *Implement controls with the least possible contractual and financial impact* is measured by the quality criterion *number of controls with contractual and financial impact*.

- *Services depending on Oxygen*

  1.3 The goal *Have the authentication/identity service for their application available when needed* is measured by the quality criteria *number of times authentication was not available in one month* and *number of times identity management was not available in one month*.

- *The Service Provider*

  1.4 The goal *Manage systems with the least possible effort and by remaining compliant with SLAs* is measured by the quality criteria *Euro/resources employed for managing hardware/software and to guarantee SLAs (including consequences for not fulfilling contractual obligations)*.

Goals and criteria regarding the RA process:

- *RMC*

  2.1 The goal *Ensure good quality of the RA Service* is measured by the quality criterion *number of important risks for the RA requester identified during an RA vs. number of unimportant risks*.

2.2 The goal *Make the RA process more efficient* is measured by the quality criterion *number of man-hours employed for an RA by the members of the team.*

2.3 The goal *Make the RA process less subjective* is measured by the quality criterion *number of choices let to the risk assessor.*

- *GIT*

2.4 The goal *Use global (shared) solutions to solve the same problem in different systems* is measured by the quality criteria *number of months to implement controls* and *number of different solutions employed to solve the same problem in different systems.*

3.4.2 Design of the evaluation process

Given the stakeholders goals and criteria, we use them to analyse and compare the results of RA_2 with those of RA_1.

First, we briefly discuss the procedure we followed. In this analysis we assume that, given a method to calculate the risk in an RA, the quality of an RA is only determined by the knowledge of the risk assessor about: (a) the ToA, (b) threats and their likelihood, (c) vulnerabilities and their likelihood and (d) how threats, vulnerabilities relate to each other and impact the ToA. We choose not to include all the social/organisational factors, e.g. the relationships among the stakeholders and their commitment to IT security, the alignment of all the stakeholders with respect to the organisation business goals, etc. These factors are indeed very important for the success of an RA but, for the sake of this evaluation, we assume them to have remained steady in the Company throughout the two RAs, and therefore to have no impact. For more examples of other IT RA social/organisational success factors, please refer to [90, 32]. The experiment we carried out compares the results of two RAs, performed sequentially by different people on the same IT system. For these reasons, to keep the experiment under control, we needed to make sure that: (1) the order in which the RAs were carried out does not influence their results, and (2) the quality of the results does not depend on the security skills of the people carrying out the RAs. To accomplish these conditions we conducted RA_2 before having access to the results of RA_1, but using the same sources of information. We used the same list of threats and vulnerabilities, as well as the same likelihood estimation, in both the RAs and we made sure the technique we employed to relate threats, vulnerabilities and nodes did not depend on the particular security skills of the risk assessor.

Table 3.3 summarises the conditions that we enforced to ensure the two RAs are comparable.

Table 3.3: RA comparison control variables

	(1) RA Order	*(2) Security skills*
(a) ToA	Used the same documentation in the two RAs. RA_2 is blind to the results of RA_1 (see Section 3.3.2.1).	Build the timed *AND /OR* dependency graph does not require to be an expert of the ToA (see Section 3.3.2.1).
(b) Threats & likelihood	The same threat list and likelihood estimation was used for RA_1 and RA_2 without any change (see Section 3.3.2.3).	Only the security skills of the RMC team have been employed in the two RAs for threat identification and likelihood estimation (see Section 3.3.2.3).
(c) Vulnerabilities & likelihood	The same vulnerability list and likelihood estimation was used for RA_1 and RA_2 without any change (see Section 3.3.2.3).	Only the security skills of the RMC team have been employed in the two RAs for vulnerability identification and likelihood estimation (see Section 3.3.2.3).
(d) Combining threats, vulnerabilities and nodes	RA_2 does not use any information of RA_1 about this (see Section 3.3.2.3).	We combined threats with vulnerabilities in accordance with the personnel who carried out RA_1. Linking threats/vulnerabilities with nodes does not depend on particular security skills (see Section 3.3.2.3).

In the next sections we compare the results of RA_2 with relation to RA_1. To do that we use four evaluation criteria from the list of Section 3.4.1. These parameters are: (2.1) the number of important risks for the RA requester vs. the number of unimportant ones (recall that in RA_1 a risk is the combination of the likelihood and impact of a threat/vulnerability), (2.2) the number of man-hours employed to carry out the RA, (2.3) the number of choices that the RMC personnel have to take and (1.1) the cost of managing availability risks. The other criteria of the list are not decidable by a risk assessor but would be observable after the system has been in use for a while. An RA will have an impact on how the system scores on these criteria but based on our evaluation alone we cannot tell what the impact of our technique will be.

For the sake of presentation, we summarise the results: (1) the QualTD model has improved the (perceived) accuracy of RA_2 by increasing the number of identified important risks for the RA requester, (2) it introduced an overhead in the number of hours employed, (3) it helped reducing the subjectivity of impact estimates in RA_2 and (4) thanks to the effects of points (1) and (3), the QualTD model supports a better risk prioritisation, which is one of the requirements for optimising the cost of risk mitigation.

To further substantiate our findings, our technique should be tested by people who did not participate in its development. We plan to have this test done by the RMC personnel of the Company.

3.4.3 Evaluation of the criteria

Evaluation of Criterion 2.1: number of important risks for the RA requester vs. number of unimportant risks

The first evaluation criterion is given by the number of important risks for the RA requester with respect to the less important ones and it expresses the `result quality` of an RA method. With *important risks*, here we mean the threats/vulnerabilities which have a high or medium risk level (i.e. the ones that will be taken into account when deciding the risk profile of the system and the risk mitigation strategy) which are judged to have been assessed accurately. In this case the number of relevant risks identified in the two RAs is not influenced by the number of threats/vulnerabilities identified or by their likelihood, as the list of threats/vulnerabilities remained the same in both assessments as well as their likelihood estimation. On the other hand, the risk of a threat/vulnerability can be overestimated or underestimated in case certain incidents and their impact are not taken into account in the RA. In this case, we would have important risks which are not considered when the risk level of the corresponding threat/vulnerability has been underestimated, or less important risks considered as important, when the risk level of the corresponding threat/vulnerability has been overestimated. We focus our evaluation on this aspect.

To determine the performance in identifying important risks of RA_2 with respect to RA_1, we compared and analysed the results of the two RAs together with the RMC personnel.

First, we made sure that risks were evaluated following the same criteria in both RAs, i.e. given the same threat and vulnerability likelihood and impact levels, the resulting risk level is the same.

Secondly, we analysed the cases in which the two RAs gave different results and we analysed the reasons for the difference. Table 3.4 summarises our findings. The RMC personnel acknowledges that in all cases, the risk estimation made in RA_2 is more accurate than the one previously made in RA_1. For this reason, in Table 3.4 we set the estimation given by RA_2 as a reference for RA_1 and we say RA_1 overestimates the risk level of a threat/vulnerability when the risk level given by RA_1 is higher than the one given by RA_2 for that threat/vulnerability. The same applies when the risk level given by RA_1 is lower than the one given by RA_2, in this case we say RA_2 underestimates the risk level of a threat/vulnerability.

Table 3.4: Summary of the number of differences between the two RAs.

	Threats	*Vulnerabilities*	*Total*
Related to Availability	22	39	61
RA_1 overestimates risk level	1	2	3
RA_1 underestimates risk level	5	13	18
Differences caused by factors not related to the QualTD model	1	6	7
Differences caused by using the QualTD model	5	9	14

In seven cases, the reason of the difference was due to external causes that do not involve the use of the QualTD model. For example, in RA_1 the vulnerabilities regarding the configuration of the Company network were usually underestimated on purpose. This because the final report of the RA carried out without the model was directed to the GIT board, who is not directly managing the Company network. Consequently, the judgement of the RMC team was that it was not useful to point out the obvious in the report, since the RA requester had no way of managing that kind of risk. The remaining 14 differences are due to a better quality of RA_2.

According to our analysis, the success of RA_2 is due to the fact that the QualTD model enables the risk assessor to estimate with more precision the impact of a threat materialising, and also to determine the impact of the vulnerabilities, by explicitly linking them to the incidents they can cause: all these operations are hard to perform without an architecture model that allows one to reason about the availability impact. For example, the impact of malware (e.g. worms) spreading across the Company network and infecting the (few) Windows servers of Oxygen were underestimated due to the lack of awareness of the risk assessors during RA_1 about the connections between these servers and other core components of Oxygen.

Evaluation of Criterion 2.2: number of man-hours employed for an RA

We split the analysis on time consumption of the two RAs according to the four steps of the Company RA process supported by the QualTD model.

1. *RA intake*: the time needed to accomplish this step with the Company method is 80 man-hours (by two people) on average. Building the timed

AND /*OR* dependency graph certainly constitutes an overhead, since it requires to formalise the knowledge acquired from the documentation and it also required at least one additional meeting with the developers of the system. In our case, we spent approximately 80 man hours (by one person) to finish the RA intake step using the QualTD model. About 40 man-hours were needed to gain knowledge of the Company, which would not have been necessary by an experienced RA team in the Company itself. So we think that one person of the RA team of the RMC department, experienced as we are, would have needed about 50 man-hours to build the timed *AND* /*OR* dependency graph. Currently, the RA intake takes 80 man-hours (by two people), so the overhead introduced by our model would be of approximately 10 man-hours. Whether this is worth the investment depends on the benefits to be gained from this in terms of a more accurate RA and in terms of the reusability of this graph for future RAs of this or other (related) systems.

2. *BIA*: including the estimation of the service/process criticality into the Business Impact Analysis is an inexpensive task, since it is already included in the procedure followed by the RMC personnel, only in an informal way. Moreover, we experienced that it was easy for the GIT to rank the services by criticality, since this knowledge is part of their everyday business. Formalising service/process criticality took less than one man-hour.

3. *TVA*: differently to the Company method, the QualTD model explicitly requires to link threats and vulnerabilities to the nodes of the timed *AND* /*OR* dependency graph to evaluate the risk. This task took us approximately 30 man-hours more than the time normally employed by the RMC personnel. However, this is partly due to the fact that we had to "learn" and get used to the definitions of the threats and vulnerabilities of the list provided by the Company. We estimate that, should we have known them better we would have done the same job in half the time. Moreover, another good part of the work was that of manually linking threats and vulnerabilities to nodes; we did this step by hand and it was very time consuming: a proper GUI would have saved us other time.

4. *Risk prioritisation*: using the QualTD model does facilitate this step. In fact, following the Company RA process, the RA team has to perform a (time-expensive) peer review of the risk evaluation performed by each member of the team, i.e. the team members have to go through their personal estimation of likelihood and impact for each threat/vulnerability and, in case they find any discrepancy, determine the reasons motivating each decision and reach a final agreement on the proper likelihood/impact levels. The

QualTD model allows one to automatically prioritise threats and vulnerabilities. Moreover, as risks are evaluated in a more detailed level (i.e. incidents instead of threats/vulnerabilities), the QualTD model facilitates the discussion on the final impact level of threats/vulnerabilities. For example, during the discussion with the RMC personnel on the final results of RA_2, we used the model to explain why a certain threat or vulnerability had a certain risk level by going into detail on the incidents that these threats and vulnerability are involved in. This technique was judged very useful and practical by the RMC personnel. It is also possible to reuse most of the work of linking threats, nodes and vulnerabilities for future RAs on the same ToA, this would reduce to zero the difference with the original method in the time consumption on the TVA step.

Evaluation of Criterion 2.3: number of choices let to the risk assessor

Making the RA results more inter-subjective (i.e. shared among the RA stakeholders) is one of the original goals of the RMC department, which aims at (a) delivering better quality results by identifying as many potential and relevant risks as possible, and (b) being able to justify the reasons why a certain threat or vulnerability was given a certain risk level.

The QualTD model supports the first objective by "forcing" the risk assessor to systematically explore all the possible combinations of threats and vulnerabilities, thus reducing the risk of mis-estimating the importance of a certain threat or vulnerability.

Regarding the second objective, since the QualTD model requires to enumerate explicitly availability incidents, it is easier for the risk assessor to trace the reasons why a threat/vulnerability was given a certain risk level (recall that we can calculate an aggregated risk level for both threats and vulnerabilities from the incident risks). Moreover, a member of the RMC department has to give (explicitly or implicitly) four subjective estimates to evaluate a single incident: the likelihood of the threat, the likelihood that the vulnerability is present in some nodes, the duration of the incident and the criticality of the services/processes it hits. By applying the QualTD model, the global impact of an incident is based on the criticality of the nodes involved, which is given by the RA requester. In this way we reduce by one fourth the number of choices to be taken by the RMC personnel (alone) for each incident, and increases the inter-subjectivity of the results, as the criticality of the services has to be agreed upon before the RA starts. In other words, even if the subjectivity of the estimates is still present, it depends less on the expertise of the single risk assessor and it is shared with the risk assessment requester, who is the final user of the assessment results.

Evaluation of Criterion 1.1: Cost of risk management

The budget for managing risks is always limited. In this perspective, optimising the costs of risk management means achieving at least the same security level for at most the same price. To achieve this goal it is important to adequately prioritise the risks one wants to manage in terms of: (a) the risk level and, (b) the cost to mitigate that risk. By providing a more precise risk prioritisation based on (a), the QualTD model supports part of the decision process of prioritising risks for mitigation purposes. At present time however, the model does not include any means of prioritising risks with respect to (b). Actually, our model bears similarities with the quantitative TD model [7] which, on the other hand, does include countermeasures and enables one to run an optimisation algorithm which select the best risk mitigation strategy taking into account (a) and (b). We believe that the same approach is applicable also to the QualTD model with few modifications. This is, however, beyond the scope of this chapter, and left as future work.

3.4.4 Applicability to other scenarios

Based on the experience of the case-study, we observe that there are two main factors which determined the success of the QualTD model.

First, the model forces the RA team to follow a more systematic approach, this means that there is less space for human errors and that the model provides an affordable way to deal with the complexity of the ToA. The QualTD model shares this characteristic with many other model-based approaches, as for example model checking techniques. This also means that, as other model-based approaches, it requires a preliminary investment in terms of time and resource to build the model. With this case-study we showed that the time investment does not exceed 50% of the time spent in an RA carried out without the model, and the resources commonly available for an RA are sufficient to build the model. In general, this investment can be very worthwhile (because e.g. it allows one to reuse the information gathered or it allows one to identify problems that would remain undetected with other techniques), or just a waste of resources. In our case, as confirmed by the RMC team, a QualTD model built for an RA can be widely reused in the following RAs of the same ToA; the resource investment can thus be compensated by reusing the model in successive RAs. This makes it particularly suitable when the ToA is periodically subject to RAs. Moreover, we believe our model-based RA approach should only be used when it either allows one to save resources in the long run (as explained above) or when the need for accurate results is worth the effort of using it. In the case analysed here, Oxygen is an availability-critical system for the Company, and therefore the need for accuracy in the assessment justified the time overhead it introduced. Also, the need to optimise the budget for risk mitigation could be a leading factor for choosing the QualTD model and afford its initial time overhead. Another scenario in which using the QualTD model could be convenient is when the timed *AND* /*OR* dependency graph can be built automatically (e.g. when a configuration management database is already present and can be used to build the graph), since in this case there is almost no time overhead.

A second success factor of the QualTD model is that it links the knowledge about security with the components of an IT architecture, their technical and functional dependencies and their importance. With this case-study we showed that the QualTD model structures information in a way that is simple enough to be used and complete enough to cover all the aspects that are important for a security RA. In fact, we did not find any uncovered risk area in RA_1 which was not covered in RA_2. For this reason, we think that the QualTD model is particularly suitable to be used to assess the availability risks of an IT infrastructure or of parts of it.

3.5 Related work

This section is divided in two parts. In the first part we make a taxonomy of standard RA methods, and we single out the methods, or the characteristics of these methods, that are compatible with the technique we presented in this chapter. In the second part we do a literature review of the techniques that use dependency-based models to improve the quality of an RA; we compare them with the technique we presented in this chapter and we discuss their applicability to the chapter's industrial case.

3.5.1 Combining the QualTD model to standard RA Methods

In this part we look at general RA methods, and we discuss under which circumstances the QualTD model can be used in combination with them. To do this, we first make a taxonomy of RA methods.

3.5.1.1 A taxonomy of RA methods

To provide a snapshot of the state-of-the-art within RA methods we follow the survey by the European Network and Information Security Agency (ENISA) [96]. The survey consists of a list of sixteen RA methods currently in use. Among these methods we only consider international standards, i.e. those which are available in English and which are actually in use in more than one country. According to these criteria, we reduce the initial list of sixteen methods to ten: CRAMM [92], EBIOS [94], ISAMM [93], ISO 13335-2 [42], ISO 17799 (now ISO 27002) [46], ISO 27001 [45], IT-Grundshutz [101], MEHARI [106], OCTAVE [82], NIST SP 800-30 [73]. The remaining six methods are dropped because of two reasons: Austrian IT Security Handbook, Dutch A&K Analysis and MARION because only available in a single language (German or Dutch), while ISF, MAGERIT and MIGRA because of lack of relevant documentation. Finally, since the list on the ENISA survey is admittedly not complete, we augment it with another popular method, the Australian/New Zeland standard for risk management AS/NZS4360 [85], and with CORAS [28], the method resulting from the EU-funded project IST-2000-25031. We explicitly choose to exclude Common Criteria [43] from this list as it is not properly an RA method, even if it requires some risk analysis to be performed.

For the sake of the presentation, we classify the twelve methods by means of three parameters: (1) the scale used to evaluate risk and risk factors (quantitative or qualitative), (2) which factors are proposed in the method to evaluate the impact level and (3) the underlying view on how risk is evaluated.

Parameter (1) determines if the risk level measures something that can be (meaningfully) expressed in numbers (e.g. money), or something which can only be expressed with ordered labels (e.g. high, medium, low). In other words, a qualitative method measures the level of a risk factor in an ordinal scale (i.e. only ordering among values are known), while a quantitative method uses measures in *interval* or *ratio* scales (i.e. the magnitude of the difference between two values in the scale is known; *ratio* scales also define an absolute and non arbitrary zero point).

Parameter (2) indicates which factors influence the impact of a security event (i.e. a threat, a vulnerability or an incident), and to which extent the method is constrained by these factors. Some methods only give general guidelines (e.g. the damage to the organisation), while others strictly define a particular set of parameters (e.g. the monetary loss, or the affected business processes).

Parameter (3) investigates what determines the risk level of a security event and how different properties are combined. To this end we elaborated five different profiles (`Type 1` to `Type 5`):

1. `Type 1`:
   Risk(Threat, Asset) = Likelihood(Threat) ⊗ *Vulnerability(Threat, Asset)* ⊗ *Impact(Threat, Asset)*
   In `Type 1` methods, risk is analysed with relation to a threat and an asset, or a group of assets and it is evaluated as the combination of the likelihood of the threat, the vulnerability level of the asset(s) to the threat and the

impact of the threat on the asset(s). For example, a `Type 1` interpretation of risk is: the risk of a burgle entering my house is obtained by combining (1) the chance that a burgle wants to enter my house (likelihood), (2) the fact that windows in my house are sometimes left open (vulnerability) and (3) what the burgle can steal once in my house (impact). We argue that this approach can be applied both to fine-grained assessments (i.e. taking into account single assets and asset-specific threats) and to more high-level assessments (i.e. taking into account only classes of assets and high level threats).

2. `Type 2`:
   Risk(Threat, Asset, Needs) = Impact(Threat, Needs) ⊗ Vulnerability(Threat, Asset)
   In `Type 2` methods, risk is analysed with relation to a threat, an asset and some security needs on the system and it is evaluated as the combination of the vulnerability of the asset and the impact of the threat on the security needs. For example, a `Type 2` interpretation of risk is: the risk of a burgle entering my house given the fact that no unauthorised people shall enter my house since I keep confidential work information there is obtained by combining (1) how bad it is that a burgle steals my confidential work information (impact) and (2) the fact that my house windows are sometimes left open (vulnerability). We argue that this approach is suitable where security requirements are clearly specified, for example for software products developed by following a rigorous software engineering process.

3. `Type 3`:
   Risk(Threat, Asset) = AnnualLossExpectancy(Threat, Asset) = Probability(Threat, Asset) ⊗ AverageLoss(Threat, Asset)
   In `Type 3` methods, risk is analysed w.r.t a threat and an asset, is intended as the annual loss expectancy (in monetary terms) and it is evaluated as the combination of the probability of the threat affecting the asset and the average loss of the resulting incident. For example, a `Type 3` interpretation of risk is: the risk of a burgle entering my house is obtained by combining (1) the probability of a burgle entering my house (probability) and (2) the average value of the goods the burgle can steal in my house (average loss). We argue that this approach is suitable in all the situations in which decisions are taken based on a financial cost/benefit analysis (e.g. insurance companies), and in which quantitative data is available (e.g. for critical infrastructures).

4. `Type 4`:
   Risk(Threat, CriticalAsset) = Impact(Threat, CriticalAsset) ⊗ Vulnerabil-

ity(CriticalAsset)
In Type 4 methods, risk is analysed with relation to a threat and an asset that has previously been identified as critical, and it is assessed as the combination of the impact of the threat on the critical asset and the vulnerability of the asset. For example, a Type 4 interpretation of risk is: given that I consider critical the goods in my house, the risk that a burgle enters my house is obtained by combining (1) the damage due to the effects of my critical goods being stolen (impact) and (2) the fact that my house windows are sometimes left open (vulnerability). We argue that this approach is suitable where there are critical assets to be protected (e.g. for utility network infrastructures).

5. Type 5:
Risk(Incident, Asset) = Likelihood(Incident) ⊗ Consequences(Incident, Asset)
In Type 5 methods, risk is analysed with relation to an incident (i.e. a combination of a threat and some vulnerabilities) and an asset, and it is evaluated as the combination of the likelihood of the incident and the consequences of the incident itself. Unlike for the Type 1 approach, this approach attributes risk levels only to security incidents (i.e. a threat exploiting a vulnerability) to assess their risk. For example, a Type 5 interpretation of risk is: the risk that a burgle enters my house through an open window is obtained by combining (1) the chance that a burgle wants to enter my house and that the window is left open (likelihood) and (2) the consequences due to my goods being stolen (consequences). We argue that this means that it is more suitable to be applied to fine-grained RAs and it is harder to apply to the high-level ones.

Table 3.5 reports the results of the classification. Most of the methods are meant to be used with qualitative measurements, and this confirms the fact that most RAs today are carried out in a qualitative way, mainly due to lack of reliable quantitative data or to time constraints [15].

Regarding impact level evaluation, we observe that ISO 13335-2 and ISO 17799 only specify that the impact of a security event is tied to the business harm suffered from the organisation. Furthermore, AS/NZS 4360 also specify the possibility of a business advantage of undertaking a certain risk, e.g. leaving servers unpatched may lead to a quicker time to market for the organisation. CRAMM, IT-Grundshutz, NIST SP 800-30 and CORAS specify more precisely how the impact level should be assessed, since they introduce the concept of damage scenarios: the RA team should identify different impact scenarios (e.g. from Catastrophic to Marginal) which describe the negative consequences of a

Table 3.5: Classification of the RA methods.

Method	*Evaluation scale*	*Impact evaluation*	*Risk evaluation*
CRAMM	Qualitative	Based on open damage scenarios	Type 1
EBIOS	Qualitative	Based on security needs	Type 2
ISAMM	Quantitative	Based on monetary loss	Type 3
ISO 13335-2	Both	Based on the business harm	N/A
ISO 17799	Qualitative	Based on the business harm	N/A
ISO 27001	Qualitative	N/A	N/A
IT-Grundschutz	Qualitative	Based on open damage scenarios	Type 5
MEHARI	Qualitative	Based on fixed damage scenarios	Type 1
OCTAVE	Qualitative	Based on critical assets	Type 4
NIST SP 800-30	Qualitative	Based on open damage scenarios	Type 1
AS/NZS 4360	Both	Based on a balance between business harm and business advantages	Type 5
CORAS	Both	Based on open damage scenarios	Type 5

risk event on the organisation. We say that these scenarios are "open" as these methods do not specify a particular set of scenarios or they do not require to use the ones they propose. On the other hand, MEHARI is based on a "fixed" impact scenario, i.e. the description of the consequences is fixed, and the risk assessor can only rank them. EBIOS imposes that the impact level of a security event is assessed in terms of the security needs (i.e. a security requirement on the IT assets) that the event violates. Similarly, in OCTAVE the impact level is measured in terms of how "hard" the security event is hitting a mission-critical asset (e.g. a server which has been pre-determined to be critical for the organisation). Finally, ISAMM measures impact by means of the money the organisation can loose because of a security event.

Regarding risk level evaluation, we observe that CRAMM (which mostly implements the principles given in BS7799-3 [21]), MEHARI and NIST SP 800-30 share the same common view on risk, i.e. they all consider risk as a combination of the likelihood and the impact of a threat to hit a group of assets and the vulnerability level of this group of assets. Similarly, IT-Grundshutz, AS/NZS 4360 and CORAS consider risk as the combination of the likelihood of an incident (i.e. a threat exploiting some vulnerabilities) and the consequences (positive or negative) of this incident happening. On the other hand, `Type 2`, `Type 3` and `Type 4` profiles are intrinsically tied to a particular approach to RA, since `Type 2` and `Type 4` rely on qualitative concepts for defining risk (e.g. critical assets, security needs) and `Type 3` relies on the quantitative concepts of probability and average monetary loss. Finally, we observe that the methods of the ISO family do not adopt any risk analysis profile. This is due to the fact that, according to ENISA [96], ISO 13335-2 is a very general guideline to set up a risk management framework, while ISO 17799 and ISO 27001 are actually not real methods for risk management, but rather compliance standards, reporting a list of controls for good security practices and the requisites that an existing method should have to be standard-compliant respectively.

3.5.1.2 Applying the technique based on the QualTD model together with other RA methods

With applying the QualTD model-based technique to an RA method we mean carrying out some specific parts of the RA process (i.e. definition of the ToA, BIA, risk identification, risk evaluation and risk prioritisation) for availability risks by using the QualTD model.

According to our classification scheme, the original RA method followed by the Company is qualitative, based on the business harm and `Type 1` with relation to the risk level evaluation. The new RA method which integrates our technique based on the QualTD model remains qualitative, but it is based on open damage scenarios and has a basic risk level evaluation of `Type 5`. We also define a procedure to aggregate the evaluation of incident risks per threat and vulnerability, making the evaluation scheme compliant to the original `Type 1`.

From the perspective of the risk level scale, the QualTD model can only be used together with a qualitative RA method; (on the other hand the TD model we proposed in Chapter 2 can only be used with quantitative ones).

From the perspective of impact level determination, we showed in the present case how the QualTD model is compatible with methods evaluating the impact in terms of business harm.

On the other hand, for methods adopting damage scenarios, the integration with our technique is only possible if the scenario descriptions used by the organisation undertaking the RA can be associated with the unavailability of a node in the timed *AND* /*OR* dependency graph.

For methods in which the impact level is based on critical assets, e.g. OCTAVE, the QualTD model cannot be applied as it is, since in the current specification we do not give a definition of critical assets. However, one possible way of adapting the model to this purpose consists in first determining the most critical processes/services and then using the timed *AND* /*OR* dependency graph to find the nodes supporting those processes/services.

We also observe that it is hard to integrate our technique with methods based on security needs, such as EBIOS. Innerhofer-Oberperfler and Breu [40] introduced

an approach, which shares some similarities with ours, and is suitable to be used in combination with these methods: we will present this approach in more detail in Section 3.5.2.
Finally, in the present specification of the model, we do not consider the business advantage of a certain risky factor, as required by AS/NZS 4360: this is the only obstacle we see for the integration of the QualTD model with this standard.

Regarding risk level evaluation, the QualTD model can be integrated with any method adopting the `Type 5` approach. For example, our model could be used in combination with CORAS as an additional, availability-specific, technique to determine the consequences of threats, in substitution of the traditional HazOp, FTA and FMECA techniques.
We showed in the present case how we integrated the QualTD model with a `Type 1` method by means of a threat and vulnerability (aggregated) risk level definition table. We believe that this approach is applicable in general if it is time and information-wise feasible for the risk assessor to explicitly enumerate the vulnerabilities present in the ToA.
Integration with a `Type 4` is instead more challenging, as it would require an approach similar to the one we described previously for OCTAVE.
Finally, RAs following `Type 2` and `Type 3` methods cannot be integrated with our model, due to the fact that `Type 2` methods already (implicitly) take into consideration the consequences of incident propagation in the definition of the security needs for each asset in the ToA, while `Type 3` methods are quantitative.

3.5.2 Dependency-based techniques for RA

Some academic researchers propose to use dependencies to improve the quality of security RAs. They have addressed this topic from multiple perspectives, such as information security, business administration and software engineering. In the literature of security RA we find three kinds of dependencies: security dependencies, software dependencies and organisational and technical/functional dependencies. In this section we will examine previous literature on these three fields which matches our work. Moreover, since our method considers the third kind of dependencies, in the final part of this section we also enumerate some techniques to build technical/functional timed *AND* /*OR* dependency graphs.

Security dependencies Baiardi et al. [12] propose a framework for RA of information infrastructures by building a hyper-graph of security dependencies, i.e. dependencies on the security properties of the system: confidentiality, integrity and availability. The timed *AND* /*OR* dependency graph is a form of attack graph in which nodes are the components of the infrastructure, and edges between nodes represent the dependency of a component on some security properties of the component it is linked to. Threats are represented as users of the infrastructures possessing some security properties on some contents, while vulnerabilities are conditions allowing the extension of security rights from one component to another. The framework allows one to rank countermeasures and create risk mitigation plans. A countermeasure can reduce the vulnerability level of a component, update dependencies, update the initial properties of a threat or increase the resources needed for an attack. Attack graph-based approaches are known to have scalability problems (e.g. see Lippmann et al. [56]) in terms of the number of hosts under assessment. This is due to the fact that building such graphs requires a large amount of work which can be only partially automated. Moreover, they require extensive and difficult to obtain attack details: this information was not available in the Oxygen RA and we believe it would not be readily available in most RAs. On the other hand, our approach is in principle less precise, but it also works when attack details are limited, as the propagation of an availability incident is mostly dependent on the architecture of the ToA, and this information is in many cases readily available.

Software dependencies Goseva-Popstojanova et al. [34] present a semi-quantitative approach for assessing reliability and availability related risks at early phases of a software life cycle by using the UML representation of the ToA. In this work, the authors use dependencies between software components to assess the likelihood of a fault propagating from a component to the other. In more detail, they use the following UML constructs: software architecture diagrams, use case diagrams, sequence diagrams and state charts of software components. By means of this information, they estimate the probability of failure of a software component, and the probability of failure of two software components interacting with each other. They consider the complexity of a software component in order to calculate the probability of its failure, and the number of messages exchanged by components to determine the probability of an interaction failure. They give the impact of a failure in a qualitative scale ranging from *Minor* to *Catastrophic*. Then, they calculate the risk level distribution of each UML use case scenario by building a Markov model from the scenario sequence diagram. Finally, they average all the single use case risk distributions to determine the overall system risk. This approach, however, is not readily applicable to all IT RAs. First, it specifically targets the assessment of risks to software components, but it is less suitable to be used for a whole IT system which includes not only software but also hardware, network components and their interaction. Secondly, as threats only software and communication failures are taken into account. In the RA of a whole IT system one is interested in assessing incidents caused by other threats (e.g. DoS attacks) and this approach does not provide a way to do this. Finally, in the Oxygen case we did not have any UML representation of the ToA and no quantitative figures about the likelihood of threats.

Organisational and technical/functional dependencies Innerhofer-Oberperfler and Breu [40] propose a model-driven approach for assessing IT-related risks using an enterprise architecture as the basis of the model. They group entities of the enterprise architecture in four hierarchical layers: business, application, technical and physical layer. They derive – by refinement – business security objectives and requirements from this enterprise architecture and from the dependencies among its constituents. The refinement process follows a top-down approach starting from high-level business units to technical and physical devices. Then, they identify and analyse risks to the security requirements by selecting threats and vulnerabilities from standard security methods, e.g. BSI IT-Grundshutz [101]. Once risks are identified, they do a bottom-up aggregation of risk scenarios to make sure risks become clearly understandable at each level of the organisation (i.e. from technical to business levels). The approach is qualitative and not linked to a specific threat-list, with a risk analysis technique very similar to the one presented in the EBIOS [94] method. In our view, the strong point of this approach is that RA is fully embedded on the organisation at all levels, from the technical level to the business management level. On the other hand, it imposes that the whole organisation is aligned and has agreed on security requirements at all levels, before the assessment can be done. This is a strong assumption for normal enterprise organisations in which such a cooperation among the many business units and the IT department is hardly achieved. For example, in our case we had little information regarding the high-level goals of the organisation and the main difficulty in applying this method would have been deriving the full list of security requirements from (unknown) high level goals. Our approach on the other hand, only requires the business owner(s) to give a relative value to the different IT services involved in the RA, which is much easier to gather from business-oriented people.

Kim et al. [50] propose a model to assess and prioritise security risks and their treatment in the context of a communication infrastructure. They do this by determining the magnitude of damages produced by a threat to the assets of the ToA, also taking into account incident propagation. To model incident propagation they use technical and functional dependencies among the assets of the communication infrastructure: for each threat they create a workflow (graph) of the incident propagation, with the assets as nodes and the relevant dependencies as edges. They annotate each edge with the probability that the destination node is affected by the damage on the source node. Finally, they model the vulnerability level of an asset by considering the “age” of the asset. Starting from the assumption that systems age over time, and because of the increased level of knowledge attackers gain on the weaknesses of the asset, attacks are supposed to have a greater probability of success over time if the system is not timely patched. Using incident propagation graphs and likelihood distribution functions, the authors are able to calculate the risk of an infrastructure over time, and to prioritise the actions to be taken to control those risks. This approach is substantially quantitative, and this makes it harder to apply due to lack of information: the data available for Oxygen was insufficient to estimate the level of weakness of the system over time. Moreover, it only considers the component's age to determine its vulnerability level, which is limiting in many situations. For example, according to the SLAs with the outsourcing company, patching is performed quite regularly on Oxygen: therefore the weakness (vulnerability) level of the assets is almost constant.

Building a dependency graph Every technique using dependencies for RA is based on the possibility of constructing a dependency graph describing the ToA. Building the dependency graph is an "extra" step which is not required by traditional RA methods, as they mainly rely on the same information but in an implicit form. For this reason, building the dependency graph in a time-effective way is essential for the applicability of dependency-based RAs.

A technical/functional dependency graph can be built either manually or automatically. Manual methods involve acquiring information by functional and technical documentation and from interviews, like we did in the Oxygen RA. On the other hand, *Static Dependency Analysis* [49] and *Active Dependency Discovery* [20] are two automatic techniques to automatically create the graph.

The former method is based on using application configuration files to derive dependencies, e.g. the `web.xml` file for Java web applications. The main drawback of this method is that it does not generate a full, cross-domain dependency graph. This is due to the fact that some dependencies are never derivable from a configuration file, and to the high number of different formats configuration files can take.

The latter method consists of measuring the variation of certain QoS parameters (e.g. availability or response time) of the ToA after some of its components are deliberately perturbed. For example, by simulating a network traffic overload it is possible to measure the dependency of the response time of a software component with relation to the network service it relies on.

An example of an Active Dependency Discovery technique was proposed by Bagchi et al. [11] for the availability of e-commerce environments. The authors propose to inject faults on the test/benchmark environment of the ToA and detect availability dependencies; the same dependencies are then also assumed to hold on the production system. This technique allows one to quickly build a dependency graph without the need to know perfectly the implementation details of the ToA. However, to build a reliable dependency graph, the test/benchmark system must be identical to the production system, which is not the case for Oxygen.

3.6 Concluding remarks

In this chapter we introduce the QualTD model and technique for the qualitative assessment of availability risks based on the propagation of availability incidents in an IT architecture. We apply the model and technique to a real-world case by carrying out an RA on the authentication and authorisation system of a large multinational company. We compare the results of this RA with the ones obtained from a previous RA carried out internally by the Company on the same system. We then evaluate the results with respect to the goals of the stakeholders of the system.

Our results show the feasibility of the QualTD model and technique, and indicate that the model provides better results in terms of accuracy, in terms of impact estimates and reduces the number of subjective decisions taken by the risk assessor. The reasons of success are mainly due to the systematic nature of the approach and to the completeness of the information the model includes. These factors help the risk assessor to deal with the complexity of the ToA in such a way that no relevant risk factor is neglected. Our analysis also shows that the QualTD model is particularly suitable to assess the availability risks of IT infrastructures or parts of them, when RAs are carried out regularly on the same target and when the final results of the RA are used to prioritise the risk mitigation strategies. In more detail, we speculate that using the IT architecture helps the risk assessor to better understand the availability-related IT risks. In turn, this can be used to improve the IT architecture with respect to availability issues.

In addition, we analyse 12 RA standard methods, and we discuss which characteristics of the standard methods are compatible with the QualTD model-based technique. Our analysis shows that the QualTD model can be used in combination with many of the most popular RA standard methods. This indicates a wide range of applicability of the technique, also in organisations not using the same RA method we used in this case.

Finally, we make a review of academic works we found in the literature which apply dependency analysis to RA. We show the type of risk analysis these techniques allow and we discuss their applicability to our real-world case. Our analysis shows that none of the techniques examined are directly applicable to our case either because they require information that was not readily available, or because they cannot satisfy the requirements of the stakeholders.

Chapter 4

A Model Supporting Business Continuity Auditing & Planning in Information Systems*

In Chapter 2 and Chapter 3 we presented two models and techniques which can be used to support the assessment and mitigation of availability risks. In this chapter we address the second research question:

"How can we improve the accuracy of current techniques for creating and maintaining business continuity plans, while guaranteeing feasibility within budget?" .

We do this by introducing a new model, which is based on timed dependency graphs as the TD model, but includes a new calculation framework that supports business continuity.

*This chapter is a minor revision of the paper with the same name [5] published in the Proceedings of the First International Conference on Global Defense and Business Continuity (ICGD&BC '07), pages 33-42, IEEE Computer Society, 2007.

4.1 Introduction

Business Continuity (BC) is the process supporting an organisation in coping with the disruptive events that may affect its infrastructure. The goal of BC is to guarantee that – after incidents – the organisation will recover operations within a predefined time. This is achieved by developing a *Business Continuity Plan* (BCP) and then putting it into practice in case of disruptive incidents. In general, a BCP consists in developing and implementing a strategy to manage incidents to the organisation's assets and recover operations. As for risk mitigation, since not all possible incident scenarios can be covered and not all possible incident recovery strategies put in place (because of financial and practical limitations) BCP includes the evaluation and the conscious acceptance of a residual risk.

Today, business activities of most organisations depend on IT systems. Therefore, a portion of every BCP is dedicated to the recovery procedures for IT systems. In this chapter we focus on the IT-related portion of BC.

One of the main goals of any BCP is achieving that crucial business processes should recover from disruption within a predefined *Maximum Tolerable Period of Disruption* (MTPD). The MTPD expresses the maximum acceptable downtime to guarantee the business continuity. As expected, the MTPD depends heavily on the business goals and we assume it is defined in terms of the *business processes*, and determined by the *business unit*.

Since business processes typically depend on a variety of underlying IT assets, the MTPD has a direct and indirect impact on the maximum downtime that these assets may exhibit in practice. Indeed, the standard technical means to realise a given MTPD is to define *Recovery Time Objectives* (RTOs) on all assets supporting business activities for which the Business Impact Analysis (BIA) has determined that it is necessary to ensure continuity; RTOs strongly depend on the technical and organisational measures the IT department implements to deal with incidents.

Problem Nowadays, determining RTOs that apply to the IT assets is done manually, and it is a subjective work which heavily depends on the experience of the IT personnel. This is not only error-prone, but it does not scale well (to the point that often, determining RTOs is not even done for all the components of the IT infrastructure, despite being required by the standard methodology for business continuity BS25999-1 [41]). Moreover, it is inconvenient in case of changes in the IT infrastructure or in the business goals. In particular, new contracts and agreements can have an impact on the quality of service a business process should deliver and ultimately on the MTPD associated to it. Likewise, changes in the IT infrastructure may affect dependencies and therefore the impact of the (RTOs of the) IT assets on the business MTPDs. In both cases, adapting the BCP to these changes, usually requires a costly new analysis involving both the IT and business units of the organisation.

Contribution We present a new model-based tool to support the analysis of temporal dependencies among IT assets and between IT assets and business process. The primary goals of our model and tool are (1) to support the IT department in setting and validating the RTOs of the IT assets of the organisation (2) to evaluate assigned RTOs w.r.t. the given MTPD to find critical points in the IT infrastructure. Ultimately, our model allows one to put down the fine-grained set of premises and assumptions to infer that a given MTPD will be achieved.

While achieving these goals, we argue that our model is particularly useful for *dynamically auditing* the BCP in various ways: first, the tool allows one to visualise immediately how changes in business goals or in the IT infrastructure affect the compliance with given (or modified) MTPDs; in particular, it is possible to compute whether the measures already in place continue giving enough guarantees also after the changes. Secondly, it allows one to validate the actual response of the IT infrastructure w.r.t. the expected behaviour, promoting a continuous refinement of the model which can adapt to new external circumstances, allowing for early detection of new threats to the business continuity targets.

Technically, this model is based on dependency graphs as the ones we presented in Chapter 2 and Chapter 3. However, here the graph is used for different purposes, and there are modelling differences (e.g. the recovery time of incidents) due to the different requirements of BC with respect to RM. However, it is possible to reuse the definition of timed dependency graph and some of the associated algorithms to explore it.

This chapter is organised as follows: in Section 4.2 we introduce the Time Dependency and Recovery model and we show how the model can be used to asses RTOs set on the IT assets and MTPDs set on the business functions (processes) these assets support. In Section 4.3 we present the application of the model in a real-world case and in Section 4.4 we discuss the feasibility of our approach in other cases. Finally, in Section 4.5 we present the related work.

4.2 Time Dependency and Recovery model

We now present the *Time Dependency and Recovery (TDR) model*, which allows us to (1) model the MTPD set on the business processes, (2) model the RTO set on the components of the IT infrastructure and (3) validate MTPDs and RTOs with respect to the effect of incidents on the IT infrastructure.

We start by providing a brief summary of the data we need to build the model.

1. The information needed to build a timed dependency graph of the IT infrastructure (see Section 2.3 for a complete list).
2. A list of possible disruptive incidents affecting the IT infrastructure, together with the time needed to repair them (per node) given the controls already in place. We also need an estimate of their expected frequency, measured in times per year.
3. The *MTPD* value for each business process on the timed dependency graph.
4. Optionally, a first estimate of the *RTO* value for each node (not business process) on the timed dependency graph.

In Section 4.4 we address the problem of how and when this data can be collected during the business continuity management process.

Let us formalise the main notions. For this, we indicate by $\mathbb{R}^{+}$ the set of positive real numbers, and by $\mathbb{T}$ the set of all time intervals (expressed in hours).

We represent the organisation's business processes and the IT infrastructure supporting them by a timed dependency graph. In this chapter we will use the definition of timed dependency graph we first presented in Chapter 2. For the sake of presentation, we will repeat here only the definition of timed dependency graph and we will then introduce the running example that we will use to describe the TDR model. For a more complete description, please refer to Chapter 2.

Definition 2.1. *A* timed dependency graph *is a pair $\langle N, \rightarrow \rangle$ where N is a set of nodes and $\rightarrow \subseteq N \times N \times \mathbb{T}$.*

Running example - Part 4.1. *We present here an example (intentionally oversimplified) of part of the business/IT infrastructure of a small bank (see Table 4.1). The timed dependency graph in this example coincides with the one of the running example in Chapter 2. p_1 and p_2 represent two business processes; a_1, a_2 and a_3 are three applications supporting business processes while db_1 and db_2 are two databases accessed by applications. Finally, m_1, m_2 and m_3 are the three machines running applications and n_1 is the network segment connecting the three machines. Figure 4.1 shows a* TDR model *built with the nodes from Table 4.1. The edges connecting n_1 to m_1, m_2 and m_3 express the dependency of the machines on the network connection with other machines. The connections from m_1 to a_1, a_2 and a_3, from m_2 to db_1 and from m_3 to db_2 express the dependency of software processes (applications or databases) on the machines they run on. For all of these connections the survival time is set to zero, since no component can*

Table 4.1: List of nodes composing part of the business and IT infrastructure of a small bank

Id	Description
p_1	Customer management process
p_2	Financial services process
a_1	Home banking application
a_2	On-line trading application
a_3	Financial founds management application
db_1	Checking account database
db_2	Trading database
m_1	Application server machine
m_2	DBMS machine
m_3	DBMS machine
n_1	Network segment

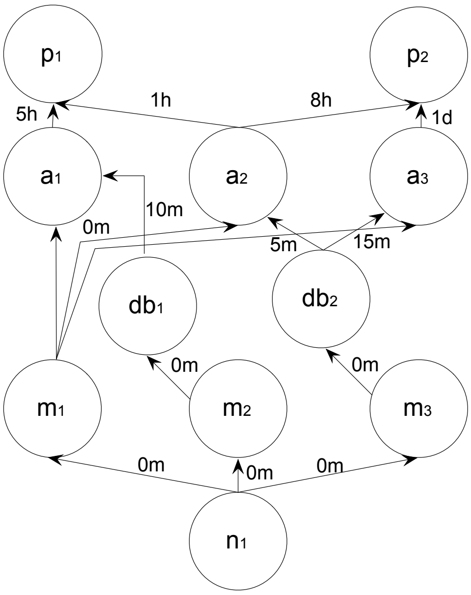


Figure 4.1: A timed dependency graph example

survive the disruption of the ones it depends on, not even for a short time. In turn, p_1 depends on both a_1 and a_2, since the customer management is achieved by providing Internet banking and on-line trading, but with different time constraints (five hours for a1 and only one hour for a_2). A similar reasoning apply to a_1 and p_2.

4.2.1 Incidents and their propagation

We adopt the same definition of incident of Chapter 2: an incident is a disruptive event causing the unavailability of an IT component (or a set of them).

Since different incidents can happen with different frequencies, business continuity should deal with the frequency of incidents to determine how often will the business operations be at risk of disruption and evaluate the associated risk. We adopt the definition of incident frequency given in Chapter 2. For the sake of presentation, we repeat here the definition.

Definition 2.5. *Given a set of incidents I, the* frequency estimate *is a mapping* $\text{freq} : I \to \mathbb{R}^+$.

For example, $\textit{freq}(i) = 3$ indicates that incident i is estimated to happen 3 times per year.

Finally, every disruptive event takes some time to be repaired. Our model encompasses an estimate of the repair time *rt* that is required by the affected nodes to become operational again. Here it is important to notice that, in many cases, it is difficult to guarantee an uniform repair time: repairing a disk could take up to two hours in most cases (say, 90% of the cases), but up to four hours in the remaining (exceptional) 10% of the cases. For instance, a software bug affecting a new application can be repaired in eight hours if it is discovered during the week, or within 24 hours during the week-end, due to lack of personnel. To be sufficiently accurate, our model requires an estimate of the recovery time for both the general and the exceptional cases. For this reason *rt* is expressed as a frequency distribution.

Definition 4.1 (Time partition)**.** *Let $\tau \in \mathbb{T}$ be a time interval, a* time partition *for τ is a set* TP *of pairs* $\langle \text{part-desc}, \text{part-freq} \rangle$ *where:*

- part-desc *is a partition of time in τ;*
- part-freq *is the frequency of* part-desc *in τ;*
- $\sum\{\text{part-freq} \mid \langle \text{part-desc}, \text{part-freq} \rangle \in \text{TP}\} = 1$.

In the above example, if we choose τ to be one week, a partition for τ would be $\textit{TP} = \{\langle \texttt{Weekdays}, 0.71 \rangle, \langle \texttt{Weekend}, 0.29 \rangle\}$.

The definition of *rt* is a refinement of Definition 2.2.

Definition 4.2 (Incident repair time)**.** *Let $g = \langle N, \to \rangle$ be a timed dependency graph, $i \in I$ be an incident happening on a set of nodes $M \subseteq N$, τ be a time interval and* TP *be a partition for τ. The amount of time needed to repair a node*

$n \in M$ from the incident i happening at a time described by an element tp *of* TP *is the mapping* incident repair time $\mathrm{rt} : I \times N \times \mathrm{TP} \to \mathbb{T}$.

In our previous example about the software bug, the *rt* function is:
$rt(\mathtt{SoftwareBug}, \mathtt{Application}, \langle \textit{part-desc}, \textit{part-freq} \rangle) =$

$$\begin{cases} 8 & \text{if } \textit{part-desc} = \mathtt{Weekdays} \\ 24 & \text{if } \textit{part-desc} = \mathtt{Weekend} \end{cases}$$

Every incident directly involves one or more nodes, causing them to be unavailable for a certain amount of time. During this time the incident may propagate to other nodes following the timed dependency graph.

We say that an incident *propagates* from a node n_1 to n_2, if they have a functional relationship (i.e. $n_1 \overset{s}{\to} n_2$) and the unavailability time of n_1, due to the incident, exceeds the survival time (s) of n_2 with respect to n_1. Node n_2 will then become unavailable until the incident is resolved.

According to this observation, we define the downtime caused by an incident to any node of the timed dependency graph (including propagation). The definition of incident downtime is a refinement of Definition 2.3.

Definition 4.3 (Incident downtime)**.** *Let $g = \langle N, \to \rangle$ be a timed dependency graph, I be a set of incidents happening on a set of nodes $M \subseteq N$, let $n \in N$ be a node and D_n be the set of nodes n depends on (i.e. $D_n = \{m | m \overset{s}{\to} n\}$), τ be a time interval and* TP *be a time partition for τ. The incident downtime is a mapping* $\mathrm{dt} : I \times N \times \mathrm{TP} \to \mathbb{T}$ *defined as:*

$$\mathrm{dt}(i, n, tp) = \begin{cases} \mathrm{rt}(i, n, tp) & \textit{if } n \in M \\ 0 & \textit{if } n \notin M \textit{ and } D_n = \varnothing \\ 0 & \textit{if } \max_{m \in D_n, m \overset{s}{\longrightarrow} n \in \to} \mathrm{dt}(i, m, tp) - s < 0 \\ \max_{m \in D_n, m \overset{s}{\longrightarrow} n \in \to} \mathrm{dt}(i, m, tp) - s & \textit{otherwise.} \end{cases}$$

Running example - Part 4.2. *Figure 4.2 shows how an incident happening at a certain time on m_3 propagates across our organisation's IT infrastructure. Assume that incident i occurs at $t = 0$ and, according to the incident repair time, it is repaired within nine hours after t. It brings down m_3; at the same time db_2 becomes unavailable, since its survival time w.r.t. m_3 is zero. After five minutes a_2 goes down and a_3 follows after fifteen minutes. Accordingly to the information in the timed dependency graph, after one hour from the disruption of a_2, process p_1 goes down and after eight hours p_2 goes down as well. After i_1 has been repaired, nine hours after t, all nodes are repaired in turn.*

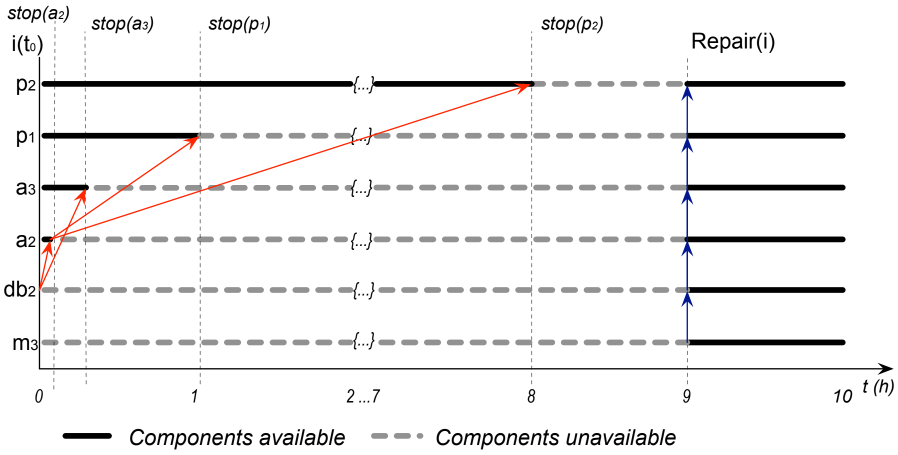


Figure 4.2: Propagation chart of an incident

4.2.2 Assessing the RTO

Recall that our goal is assessing whether, during the normal operation, business process will comply with the Maximum Tolerable Period of Disruption that has been determined (by the business unit) for the business relevant processes. The formal definition is the following.

Definition 4.4 (MTPD). *Let $g = \langle N, \rightarrow \rangle$ be a timed dependency graph and $P \subset N$ be the set of business processes. The Maximum Tolerable Period of Disruption is a mapping* $\mathrm{mtpd} : P \rightarrow \mathbb{T}$.

According to BS 25999-1, the MTPD is determined based on the impact that the disruption of the business process would have on the organisation. The impact is expected to increase over the disruption time, and to vary depending on the day, month or point in the business lifecycle. The MTPD subsumes the impact evaluation and expresses a single time value after which the impact of the process disruption would become unacceptable for the organisation.

Running example - Part 4.3. *The two business processes in our example are noticeably time-dependent, because they both require customer interaction and, in the case of p_2, the operational disruption causes a direct financial loss to the bank. Because of this, it is reasonable to assume that the MTPD is very short, as reported on Table 4.2.*

Table 4.2: MTPD values for the processes

Id	Description	MTPD
p_1	Customer management process	3h
p_2	Financial services process	0.5h

4.2.2.1 Complying with the MTPD

One of our goals is to check under which circumstances we can expect to be able to comply with the MTPD (i.e. we can expect all the business processes to recover from disruptions within the maximum time given by the MTPD). To this end, our model allows us to determine, given the MTPD for the business critical processes, what is the *maximum recovery time* that each node in the TDR model has to respect. Assuming that the timed dependency graph is acyclic this can be defined as follows.

Definition 4.5 (mrt). *Let $g = \langle N, \rightarrow \rangle$ be a timed dependency graph, $P \subseteq N$ be the set of nodes in g representing business processes, then for each $n \in N$ we define the* maximum recovery time *of n* mrt *: $N \rightarrow \mathbb{T}$ as:*

$$\mathrm{mrt}(n) = \begin{cases} \mathrm{mtpd}(n) & \textit{if } n \in P \\ \min\{\mathrm{mrt}(m) + s \mid (n \overset{s}{\rightarrow} m) \in \rightarrow\} & \textit{else.} \end{cases}$$

The definition of *mrt* is well formed under the assumption that every node in the graph is (directly or indirectly) connected to a business process node, i.e. $\forall n \in N$, $\exists p \in N, \{n1, \ldots, n_x \mid n_i \in N\}$ such that p is a business process and $(n \xrightarrow{s_0} n_1) \in \rightarrow$, $(n_1 \xrightarrow{s_1} n_2) \in \rightarrow$, $\ldots$, $(n_x \xrightarrow{s_x} p) \in \rightarrow$.

Assuming that the TDR model is faithful, i.e. that it reflects well how incidents propagate across the infrastructure, the relevance of the maximum recovery time is given by the following result.

Proposition 4.1. *Let $g = \langle N, \rightarrow \rangle$ be a timed dependency graph, let i be an incident affecting a (set of) nodes and happening in a certain time partition* tp *and $P \subset N$ be the set of nodes representing business processes. If $\exists n \in N \setminus P \mid \mathrm{dt}(i, n, \mathrm{tp}) > \mathrm{mrt}(n)$ then $\exists\ p \in P$ for which $\mathrm{dt}(i, p, \mathrm{tp}) > \mathrm{mtpd}(p)$. On the other hand, if $\forall n \in N \setminus P\ \mathrm{dt}(i, n, \mathrm{tp}) \leq \mathrm{mrt}(n)$, then $\nexists\ p \in P \mid \mathrm{dt}(i, p, \mathrm{tp}) > \mathrm{mtpd}(p)$.*

This proposition states that if an incident on a node is not repaired within its MRT, then at least one business process will be disrupted for longer than its MTPD. On the other hand, if an incident is always repaired within the nodes MRT, then no business process will be disrupted for longer than its MTPD. It is possible to verify the intuition behind this proposition by comparing the definitions of incident downtime (Definition 4.3) and the definition of maximum recovery time (Definition 4.5). The maximum recovery time is calculated by taking into account the MTPD set on business processes and to make sure that a the downtime of a node does not cause (by propagation) a downtime on the business processes longer than their MTPD.

Therefore, the *mrt*(n) we have defined is actually the *maximum downtime* we can tolerate on n to ensure that the MTPD is respected for each business process depending (directly or indirectly) on it. Of course, the validity of this result depends on the accuracy of the TDR model, but it is worth mentioning here that (a) as we discuss later, the data needed to build the TDR model is in most cases available and (b) the model can be refined over time by using statistics on incidents and their recovery.

Recovery Time Objectives To comply with the MTPD, which is a high-level measure imposed on the critical business processes, one sets a *Recovery Time Objective* (RTO) on all the assets of the organisation that can be directly involved in incidents (for IT this implies machines, applications, infrastructure, etc.). Our tool can be used to do this in an automatic, fairly user-friendly way. This already represent an improvement on everyday practices, which lack standard procedures to set RTOs with the consequence that sometimes RTOs are not set at all.

Definition 4.6 (RTO)**.** *Let* $g = \langle N, \rightarrow \rangle$ *be a timed dependency graph and* $P \subset N$ *be the set of nodes in* g *representing business processes, the Recovery Time Objective is a mapping* $\mathrm{rto} : N \setminus P \rightarrow \mathbb{T}$.

Proposition 4.1 implies that, if for each node n in which *mrt*(n) is defined *rto*$(n) \leq$ *mrt*$_{mt}(n)$, then the compliance with respect to the RTO implies compliance w.r.t. MTPD. Our model allows us to *validate* the RTO as follows.

Proposition 4.2. *Let* $g = \langle N, \rightarrow \rangle$ *be a timed dependency graph, and* rto *be an RTO for it. Assume that for each* $n, m \in N$ *such that* $n \xrightarrow{s} m$ *the following holds:*

$$\mathrm{rto}(n) \leq \mathrm{rto}(m) - s \tag{4.1}$$

Then, for any two nodes n *and* m*, we have that an incident on the node* $n \in N$ *that causes on* n *a disruption shorter than* $\mathrm{rto}(n)$ *will never cause by propagation on* m *a disruption longer than* $\mathrm{rto}(m)$.

If (4.1) is not satisfied for some n, m, then an incident on n which causes on n a downtime shorter than *rto*(n) would cause on m by propagation a downtime *longer* than *rto*(m). In other words, if (4.1) is not satisfied then one could witness the paradoxical situation that the RTO on m is not satisfied because of an incident on another node n, while this incident remained within the RTO of n in the first place. RTOs are meant to define a local standard that guarantees a global continuity level; because of this we believe that an RTO not respecting (4.1) would be of no practical use.

Running example - Part 4.4. *By applying Proposition 4.1 to the TDR model in Figure 4.1, we evaluate the* mrt *for each node w.r.t. the MTPD expressed in the previous example. Table 4.3 reports the original RTO value assigned in the traditional way (i.e. manually) by the IT-BCP group on the IT assets of the TDR model as well as the automatically evaluated* mrt*. The RTO assigned on the IT assets is in some cases too short and in other cases too long, i.e. insufficient to ensure the business continuity. By applying Proposition 4.2, we compare the original RTO w.r.t. the* mrt *and find four critical points (outlined by a bold circle in Figure 4.3), where the original RTO value exceeds the* mrt*.*

Table 4.3: Manually assigned RTO vs. *mrt* values evaluated by means of the model

Id	RTO	mrt
a_1	6h	8h
a_2	6h	4h
a_3	6h	24h 30’
db_1	5h	8h 10’
db_2	5h	4h 5’
m_1	5h	4h
m_2	7h	8h 10’
m_3	3h	4h 5’
n_1	8h	4h

4.2.2.2 Exceeding the MTPD

As it is impossible to achieve total security, it is often difficult to comply at all times with the given MTPD. Disasters happen and it is normal to accept a residual risk, implying that the given MTPD may be exceeded in truly exceptional situations. For instance, there could be some particularly serious incidents that cannot be recovered in time. On the other hand, the IT department may be unprepared to handle some disruptive events due to lack of personnel or resources. To deal with that, two solutions are possible: (1) the organisation’s management decides

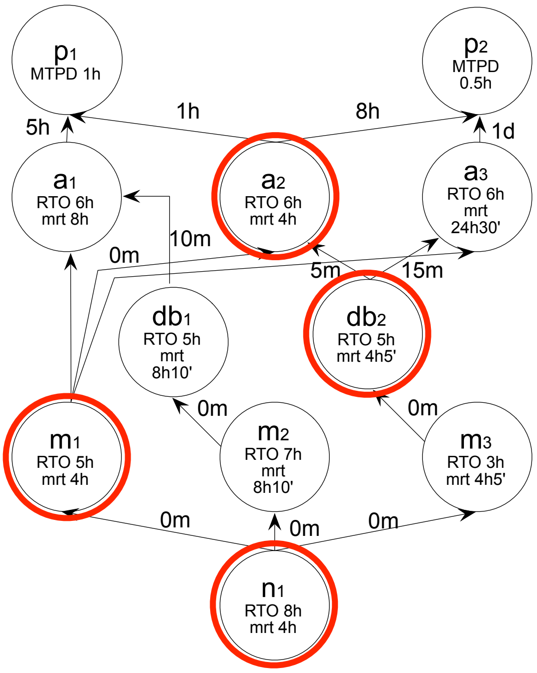


Figure 4.3: Critical points in which the manually assigned RTOs exceed the maximum recovery time.

to employ more resources and deploy new control measures allowing to shorten the disruption time, or (2) the risk of exceeding the MTPD is accepted within a given probability.

However, to be able to accept the risk of exceeding the MTPD, an organisation needs to have a reasonable estimate of *how often* this is going to happen in a given time period (which could be one year or ten years, for instance).

To make this estimate we can use our TDR model. We know from Proposition 4.1 that every time an incident occurring on an node is not repaired within its mrt one or more business processes depending on the node will become unavailable for longer than their MTPD.

Therefore, to evaluate the frequency a business process p exceeds its MTPD we need to make an estimate of how many incidents affecting a node on which p (directly or indirectly) depends on, cause the *mrt* of the node to be exceeded. To this end we use the recovery time distribution that is evaluated during the RA phase of the business continuity management process.

Definition 4.7 (Frequency *mrt* is exceeded)**.** *Let* $g = \langle N, \rightarrow \rangle$ *be a timed depen-*

dency graph, $n \in N$ be a node, $p \in P \subseteq N$ be a business process and $\mathrm{mtpd}(p)$ *be its MTPD. If I is a set of incidents affecting n, τ a time interval and* TP_i *the time partition for τ on which the incident recovery time for $i \in I$ is assessed, then the frequency n exceeds its* mrt *because of incidents in I is given by the mapping $\Phi : N \times I \to \mathbb{R}^+$:*

$$\Phi(n, I) = \sum_{i \in I} \left[\mathrm{freq}(i) \sum_{\mathrm{tp}=\langle \text{part-desc},\text{part-freq}\rangle \in \mathrm{TP}_i \text{ \textit{and} } \mathrm{dt}(i,n,\mathrm{tp}) > \mathrm{mrt}(n)} \text{part-freq} \right]$$

Intuitively, $\Phi(n, I)$ expresses an upper bound on the number of times a node n exceeds its *mrt* because of a set of incidents I (i.e. when the incident occurrences are not overlapping). The following proposition expresses how we use Φ to evaluate *freq-ex*(p), which expresses (the upper bound of) how many times the MTPD is exceeded, given a business process p.

Proposition 4.3. *Let $g = \langle N, \to \rangle$ be a timed dependency graph, $p \in N$ be a business process and I be a set of incidents, then:*

$$\text{freq-ex}(p) = \Phi(p, I) \tag{4.2}$$

In other words, how often a process exceeds its MTPD is determined by the sum of the frequencies the nodes it depends on exceed their mrt. With such an information, the business unit is able to verify if the residual risk it is willing to accept is not further exceeded by the IT department. Such a condition would require the development of more effective strategies to reduce the recovery time to incidents.

Running example - Part 4.5. *Let us introduce two incidents i_1 and i_2, the first affecting a_2, the second affecting db_2. i_1 is estimated to happen five times a year and it is repaired within three hours in the 80% of the cases and within eight hours in the remaining 20%. i_2 is estimated to happen seven times a year and it is repaired within four hours in the 90% of the cases and within six hours in the remaining 10%. If we consider the MTPD of p_1 (three hours), then the* mrt *is 4h for a_2 and 4h5' for db_2. The (upper bound of the) frequency db_2 exceeds its* mrt *is 0.7 times a year, while the (upper bound of the) frequency a_2 exceeds its* mrt *is 1 time a year. Consequently, assuming that the only incidents affecting the nodes in the timed dependency graph are i_1 and i_2, our tool allows us to compute that p_1 is expected to exceed its MTPD at most 1.7 times a year (once a year by 3h, equivalent to the 200% of the MTPD, and 0.7 times a year by 1h55', equivalent to the 164% of the MTPD).*

4.3 The Practice

Our experience on BCP auditing is based on the general approach used by KPMG Italy. We implemented a prototype tool from our model as an additional component of KARISMA (which is the tool developed at KPMG to support RA, see Section 4.4): this enables us to repeat and validate the assessment previously done by the KPMG auditing team.

The tool is composed of two main modules: the *data import module* and the *algorithm module*. The goal of the data import module is to obtain the data needed to build the TDR model from the KARISMA database. It reuses some of the code written for the prototype we built to test the TD model (see Section 2.4). The data import module allows us to obtain sufficient information to build the timed dependency graph, plus a list of (business continuity-related) incidents, their frequency and recovery time. In addition, we obtain the MTPD value for all business processes and subprocesses and the value of the previously assessed RTO for some nodes of the timed dependency graph (only the applications).

The algorithm module implements the algorithms to compute the incident downtime (see Definition 4.3), the maximum recovery time (see Definition 4.5), the frequency MTPDs are violated (see Definition 4.7) and to find critical points in the previously assessed RTOs (according to Propositions 4.1 and 4.2).

The pseudo code for the algorithm to compute the maximum recovery time is shown in Algorithm 4.1, while the algorithm to find critical points is shown in Algorithm 4.2.

Algorithm 4.1 Algorithm to compute the maximum recovery time of a node

```
function mrt($n$)
if $n$ is process then
    return mtpd($n$)
else
    return min $\{mrt(m) + s \mid (n \xrightarrow{s} m) \in \rightarrow\}$
end if
end function
```

We tested our model with the information of an Italian primary insurance company obtained from the KARISMA database. This data was collected during an auditing activity carried out by KPMG to set-up a BCP, and contains information regarding the business and IT infrastructure of the primary insurance company (19 macro business processes and 122 sub-processes) and the results of the BIA analysis carried out by the KPMG personnel, which provides the MTPD value for each business process and subprocess. The remaining information required by

Algorithm 4.2 Algorithm to find critical points

```
function critical($n$, $\epsilon$)
if $n$ is process then
    return false
end if
if rto($n$) - mrt($n$) > $\epsilon$ then
    return true
end if
for all $(n \xrightarrow{s} m) \in\rightarrow$ do
    if rto($n$) ≤ rto($m$) - $s$ then
        return true
    end if
end for
return false
end function
```

our model (about incidents, repair time and frequencies) was also provided by the KPMG auditing team who conducted the assessment.

Regarding the BIA, the procedure used by KPMG analysts to establish the MTPD for the business processes is based on the qualitative analysis of the impact, as perceived by the process owner (business unit), of the consequences of a disruption on the process itself. On the other hand, regarding RTOs, only certain nodes (most of the applications) are taken into consideration and are labelled with an RTO, since it is difficult to properly evaluate the relationships between the different nodes manually. The approach followed by the KPMG analysts to assign an RTO to application nodes is to set a value smaller or equal to the minimum MTPD value of the processes (or subprocesses) that the application supports. Figure 4.4 shows an example of such an assignment with the (anonymised) data of the primary insurance company. Application *A2* supports two business processes: *P2* and *P3*. *P2* has an MTPD of 7 hours, while *P3* has an MTPD of 20 hours. The RTO on *A2* is set to 7 hours, which is equal to the minimum MTPD value of the two processes *A2* supports.

The first important contribution of our model is that all the relationships are evaluated, thus enabling the IT department to extract the RTO values for each involved node (even machines, network and infrastructure components). We used the *mrt* value automatically calculated by the tool as RTO for all the nodes of the TDR model. In this way, we are sure that RTOs are compatible with the MTPD of the business processes (see Proposition 4.1), and that RTOs are pairwise compatible (see Proposition 4.2). Having an RTO set for all the components of the IT infrastructure allows the IT department to systematically assess its ability to com-

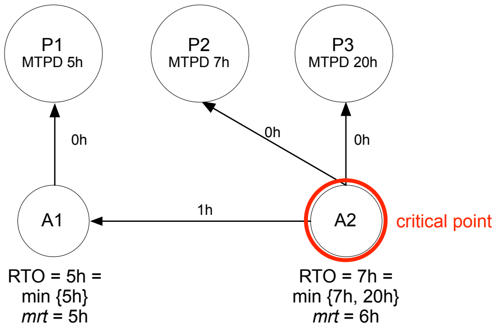


Figure 4.4: An example of critical point found with our tool. The misleading evaluation of A2's RTO to 7h is caused by the lack in considering the relationship of A2 with A1. If A2 is repaired in more than 6h, process P1 will be disrupted for more than its MTPD.

ply with these RTOs, to develop strategies to achieve them in case of disruption and, if necessary, agree with the business unit on an exceeding rate.

Secondly, we found some critical points in the previous setting of RTO values for the application nodes composing the IT infrastructure of the insurance company. In case of applications supporting both business processes and other applications, we found that the RTO was in some cases underestimated (longer than required) because the dependencies between applications had not been taken into consideration when initially assigning RTOs. In the case of Figure 4.4 we see a simple example of such critical point. According to KPMG's method of assigning RTOs, the RTO of *A2* is set to 7 hours. However, in case *A2* is disrupted and becomes operational after 7 hours, *A1* would be unavailable for 6 hours, which conflicts with its RTO (5 hours) since it would cause *P1* to be unavailable for 6 hours (i.e. longer than its MTPD).

Summarising, our tool allows one to perform two different assessments: firstly to set properly RTO values for a given IT infrastructure component; secondly, to support auditors during the BCP validation. Once a BCP has been established and put in place, the validation phase occurs to ensure that the plan is adequate, complete and appropriate w.r.t. the organisation's IT infrastructure [70]. A crucial point is based on the auditing of recovery controls: the auditor must verify that RTO values meet the business requirements. Our tool supports this kind of verification, since every check can be made in an automatic way (after the model is created or updated), possibly discovering weaknesses in the BCP.

4.4 Discussion

In this section we argue the feasibility of our approach and its usefulness to support *dynamic auditing of the BCP*.

Feasibility The main concern regarding the feasibility of our approach is whether organisations are able to collect the required information to build the model. Fortunately, the experience with KPMG shows the data it requires can be available (at least at some financial organisations). More in general, the TDR model requires information to (1) build a timed dependency graph, (2) enumerate incidents together with their estimated frequency and repair time, (3) collect the MTPD for business processes and (possibly) the existing RTO set on the components of the IT infrastructure.

Regarding (1), in Chapter 2 and in Chapter 3 we discussed the feasibility of acquiring information to build a timed dependency graph (the information needed to build a timed *AND* /*OR* dependency graph is almost the same of the one required for a timed dependency graph). The same considerations also hold for this case.

Regarding (2), an inventory of possible incidents has to be compiled during the RA phase of the business continuity management process. Incidents can be derived from existing threat lists which can be found on several standard methods (e.g. in BS 7799-3 [21]). Most of the incidents that are of interest in business continuity management are natural disasters (e.g. flooding, storms, etc.) or infrastructure faults (e.g. power outages). For this kind of incidents there is more information publicly available with respect to other kinds of IT security events (e.g. hacker attacks). In fact, the second ones are more difficult to analyse due to lack of detailed historical records, since organisations normally tend not to disclose information about them. On the other hand, a numerical value about the expected frequency of – say – a lightning hitting a datacentre can be reasonably obtained based on historical records (e.g. from the frequency of storms in the area a datacenter is located).

Finally, regarding (3) we note that according to the BS 25999-1 standard, establishing an MTPD for the critical business functions is a key activity to set up a BCP. Therefore, we can assume an organisation willing to set up or assess an existing BCP to have carried out the BIA. To further substantiate this argument, we note that this data is also collected by tools devised to assist the creation of a BCP. For instance, this was the case in the case-study we carried out. Among the information KARISMA collects via a question-driven procedure, there is a map of the business processes. The Business Impact Analysis focuses on the mapped processes and produces, among others, the MTPD value for them.

Continuous Auditing Finally, we argue that our framework is particularly useful to support a *dynamic* auditing process. The concept of dynamic auditing is well-known among the risk management strategies, particularly in the field of software engineering [61]. The goal of this process is to continuously assess what could go wrong in projects (i.e. what the risks are), determining which of these risks are most relevant, and implementing strategies to deal with them. Even though many of the methodologies for risk management [18, 45], as well as those for BC [41], include a monitoring and reviewing step, this process can be performed with different degrees of granularity, according on how flexible the adopted techniques are. For example, a change on the IT infrastructure, involving the dismantling of a set of applications and machines and the introduction of new software and hardware components, may involve either the assessment of the new components only, or of the whole organisation, depending on how much it is possible to reuse the previous assessment results.

Thanks to the fine granularity and the high degree of independence of the used information (time dependencies, assessment of incidents, importance of processes to the business), our model and tool are particularly suitable to support a dynamic assessment process.

For instance, when dealing with a change in the organisation, be it the rearrangement of the IT infrastructure or a new business strategy, after a simple update of the model, the framework can be used to evaluate the new time constraints within which incidents must be repaired to preserve the business continuity. In the case a new component is added to the information system, it is only necessary to add the new component in the TDR model and specify its functional and temporal relationships with the other components to evaluate its new RTO. On the other hand, if a process becomes more important for the organisations business (due to changes in business strategy), it is possible to change its MTPD and automatically assess the IT infrastructure to verify if it is still able to ensure the new time constraints.

In addition, after the occurrence of an incident, our model allows us to verify if the incident response propagation is compliant with the expected behaviour. It might happen that a time dependency between two applications, that was estimated to be of one hour, is in fact of one hour and a half. Furthermore, one might observe that the response time to an incident exceeds the forecasted RTO. In those cases, the model can be easily updated with the new collected information, thereby allowing to rapidly assess the new situation and develop new and more efficient BC strategies, if needed. This feature adds quality to our solution since it enables the BC team to organically capitalise on practical experience to improve accuracy of the model and of the outcome in time.

In this perspective, the ability to easily refine the model helps at improving the way organisations traditionally deal with incidents. Instead of simply solving the problem when it happens and then forgetting about it, our solution promotes the continuous monitoring of the performances of the repair operations by collecting new information as incidents occur and then use them to improve the efficiency of the response on new occurrences.

Summarising, our system allows one to (a) easily adjust the model to changes in the organisation and/or its business target, without the need of a complete new assessment, and (b) refine the model (i.e. make it more precise) in the moment in which new and more accurate information is available about the actual behaviour of the organisation.

4.5 Related Work

Although the business continuity management process is well described in a number of works [51, 62] and recently has been standardised by the British Standard Institute [41], formal models to support it are still understudied. Despite this situation, there are some specific fields for which specific tools have been developed to accurately evaluate the survivability of IT systems. This is the case of telecommunication networks, where the high availability of the network must be ensured through a proper BCP. Jrad et al. [48] propose a BCP model devised to determine the expected downtime due to disaster events as well as normal and software failures in a networked environment and especially tailored for telecommunication networks. The model can be also used to predict the probability that a disaster will cause a service disruption. The model could be integrated with ours to provide an estimate of incident likelihood and repair time. Our model could then be used to set or assess RTOs on the infrastructure.

Another approach to evaluate the survivability of a system is the one proposed by Cloth and Haverkort in [25]. They describe the system under assessment as a Stochastic Petri net and then automatically convert it into a Continuous Time Markov Chain (CTMC) . Finally they use a model checking engine to obtain a time-probability chart that expresses the recovery probability in relation to the recovery time. The model outcome is typically used to deal with dependability issues in system design, but is not readily usable for the business continuity of large infrastructures. To apply this approach a fine grained description of the system is required. However, obtaining information at this detail level might be hard in terms of time and resources.

In addition to academic work, there exist a number of commercial tools supporting BCP. The most closely related to our work is Shadow Planner [111]. It is an (industrial) application developed to support organisations in assessing risks and establish a BCP. The software has several modules to map the organisations' IT infrastructure, collect BIA information, asset values, etc. Thus, it is able to evaluate the monetary impact of a certain incident. Differently from our approach, it is not based on a model and the relationships between different IT infrastructure components are not properly evaluated. This could hardly affect the way a disruption event is evaluated, resulting in an erroneous planning of countermeasures to ensure business process MTPDs (and the related RTOs).

4.6 Concluding remarks

In this chapter we address IT availability planning by considering the IT-related part of business continuity management.

We present the TDR model and framework to support the set up and validation of a BCP. The model allows one to: (1) determine the *maximum recovery time* an IT component needs to be restored after a disruptive event to comply with the business requirements (i.e. the MTPD of the business processes the IT component supports), (2) validate the compliance of RTOs set on IT components w.r.t. the maximum recovery time, (3) validate the mutual compatibility of RTOs set on IT component that depend on each other and (4) evaluate the residual risk of a given BCP by considering the frequency MTPDs set on business processes may be violated.

Finally, we provide evidence about the feasibility of our approach by reporting on their successful application in the validation of the BCP of an italian primary insurance company, carried out with data collected by the Italian branch of KPMG S.p.a. and discuss the applicability of our model-based technique to similar cases in which the information required by our model is available.

Chapter 5

A^{2}THOS: Availability Analysis and Optimisation in SLAs*

Having addressed the first research question regarding the management of availability risks in Chapter 2 and Chapter 3, and the second research question about business continuity management in Chapter 4, we now address the third and final research question:

"How can we improve the accuracy of current techniques for managing availability-related SLAs, while guaranteeing feasibility within budget?"

In this chapter we focus on the problem of analysing and optimising the availability of so-called mixed sourced IT services, i.e. services which are (partly) managed by an outsourcer and regulated by means of SLAs. To address this topic we introduce a model-based framework called A^{2}THOS.

*This chapter is a minor revision of the paper with the same name [1] submitted for publication at the International Journal of Network Management (NEM) in April 2010.

5.1 Introduction

Nowadays, the IT infrastructure of most large organisations is so complex that it is often organised in terms of *services* that are offered as part of an internal market in which different business units offer and buy IT services to and from each other. In some cases, services are acquired from an external organisation rather than from an internal business unit (outsourcing). Typically, services offered by an internal provider are customised and tailored to support the business goals of the organisation, while those offered by external providers are standardised and large-scale, and therefore are less specific but potentially cheaper than those implemented internally. In some cases, internal providers outsource some sub-services to external ones, for instance when it lacks specific competencies (e.g. SAP configuration). This is a so-called mixed sourcing strategy.

Regardless of whether the service is bought internally or externally, the terms and conditions of the contract are determined in the SLA. (Figure 5.1 summarises the concept of mixed-sourced IT services regulated by SLAs.) For instance, ITIL [62] is one of the most popular frameworks providing guidelines and best practice for a correct IT service management and it describes this process in detail in [64].

In this chapter we focus on IT service availability, which is at the core of customer satisfaction and business success for organisations [63], and indeed it is one of the main topics in a SLA. In fact, a typical SLA includes hard clauses on the *minimal availability* of the service offered (for example, it may include that the service should not be "down" for more than two hours per week, and a penalty fee for each week in which this is not satisfied).

Now, the two concerns we focus on (and at the same time the two questions to which we provide an answer within the limits of the settings of this chapter) are:

1. how can a business unit check and/or guarantee that a given (offered) service will respect some given minimal availability levels;
2. as (1.) while minimising costs.

Let us elaborate on these two points and explain why they are not only relevant, but also non-trivial problems.

An IT service is usually offered by a system consisting of several *components*. These components can interact in non-trivial ways: for instance a component could be crucial to the service in a way that if the component is unavailable then the service becomes unavailable as well; other components may be organised in such a way (e.g. exploiting redundancy) that only if a number of them fails the service will be affected. In addition, a component may depend in a non-trivial way on sub-services which are in turn regulated by other SLAs.

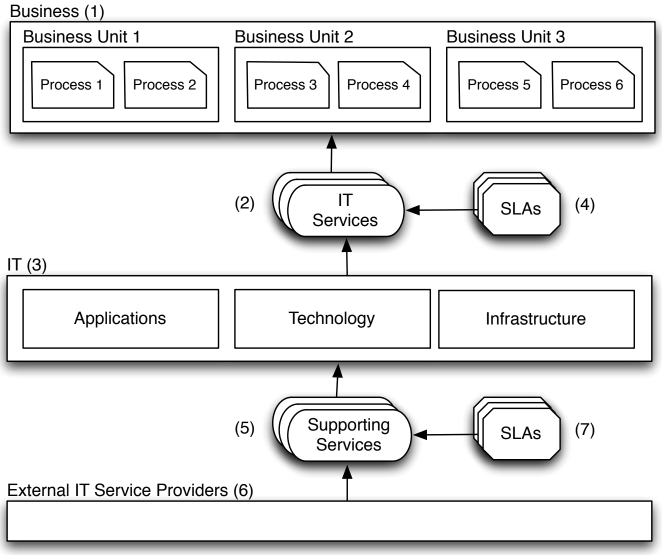


Figure 5.1: Mixed-sourced IT service provision regulated by SLAs. The organisations business units (1) use IT services (2) provided by the internal IT department (3). These services are regulated by means of SLAs (4). In turn, the IT department is using supporting services (5) offered by an external provider (6) to run (part of) the IT infrastructure. Also these services are regulated by means of SLAs (7).

To ensure that the minimal service availability remains within the agreed margins, IT managers can take reactive (e.g. monitoring, measuring) and/or proactive measures. A key proactive measure is planning and designing service availability when services are created or changed. At the business level, planning service availability allows the service provider to set availability figures on the SLAs that both satisfy the customer needs and can be guaranteed by the technical infrastructure providing the service. To achieve this at the technical level the service provider needs to (a) calculate the availability of the IT system providing the service(s) based on the information available on system components, and (b) make appropriate system design choices to support a specific availability level by selecting the system components based on their contribution to the availability of the system.

Reliability studies have introduced a number of by now standard techniques (e.g. Continuous Time Markov Chains (CTMC) [68] and Petri Nets [33]) which allow one to compute system availability when the mean time between component failures and the mean time to repair a component is known. However, in the context of mixed-sourced IT services, this information is usually not available. Instead, SLAs between the external and the internal provider typically only include the *minimal* guaranteed availability of the component. Therefore, it is not possible to apply these standard techniques to calculate the system availability (see Section 5.2 for details).

Regarding the second point, the service catalogue of most IT outsourcing companies include different availability levels (e.g. gold, silver and bronze) with different associated prices (same service, only different availability levels, at different costs). Service providers need to minimise the cost of outsourced (sub)services while guaranteeing that their own service achieves the desired minimal availability level. Given the interactions mentioned above, this is a non-trivial optimisation problem: one needs to determine the combination of minimal availability levels for the sub-services in such a way that the total cost is minimal while ensuring that the resulting service achieves the availability specified in the SLAs. This cannot be solved without the use of specific optimisation algorithms and typically IT managers choose non-optimal, conservative solutions.

Contribution We present A^{2}THOS, a framework for the analysis and optimisation of the availability of mixed-sourced IT services. The framework consists of (1) a modelling technique to represent partially-outsourced IT systems, their components and the services they provide, based on AND /OR *dependency graphs*, (2) a procedure to calculate (a lower bound of) the system availability given the (lower bounds of) components availability, and (3) a procedure to select the optimum availability level for outsourced components in order to guarantee a desired target availability level for the service(s) and to minimise costs.

An *AND /OR* dependency graph is an *AND /OR* graph in which nodes represent system components and services, and edges between nodes represent the functional dependency of one node with the other. We use the graph in order to calculate a state function describing the availability of each service based on the state of the components (operational or not operational). We then use the state function and the information about components availability to determine a lower bound for the availability of the service, by setting up a linear programming problem. Based on this procedure, we finally present the procedure to set up an integer programming problem which allows one to determine the cost-optimal combination of availability levels for outsourced components in order to guarantee a target service availability. We show the practical use of A^{2}THOS by implementing it in a tool which we apply to the service availability planning of an industrial case. In Chapter 3 we argue that reliable quantitative information for risk assessments can be hard to acquire, especially regarding likelihood of threats (e.g. hacker attacks) and precise financial loss estimates. However, in the case of service level management this is not the case. Both internal SLAs between an organisation IT department and business units and external SLAs between the IT department and external service providers include a quantitative value expressing the lower bound of the IT service availability. This allows us to apply in this chapter a quantitative technique.

Limitation of the approach A^{2}THOS uses an *AND* /*OR* dependency graph to represent IT systems, thus it is unable to explicitly represent failure recovery mechanisms such as availability of spare parts. Spare parts are used to implement warm and cold standby mechanisms. For example, to shorten the downtime caused by a server breakdown, the system administrators can keep another server ready to replace the broken one. This second server is the spare part. When it is always running (but not operating) and the workload of the broken server is automatically routed to the spare server, this mechanism is called hot standby. When the workload of the broken server needs to be manually routed to the spare server, this mechanism is called warm standby. When the spare server is not readily available, but it needs a setup phase before the workload of the broken server can be redirected to it, the mechanism is called cold standby. Our representation allows us to explicitly model hot standby mechanisms by using *OR* nodes, but it is not applicable in case of warm and cold standby mechanisms. We share this limitation with other well-known modelling techniques, such as traditional Fault Trees (FTs) and Reliability Block Diagrams (RBDs).

Organisation The rest of the chapter is organised as follows. In Section 5.2 we present the related work in the fields of reliability and IT service composition. In Section 5.3 we provide the mathematical foundation for using them to calculate service availability. In Section 5.4 we present the procedure to find the optimal choice of availability level for outsourced components. In Section 5.5 we describe the tool we created to implement the A^{2}THOS framework and the benchmarks we conducted to test its scalability performances. Finally, in Section 5.6 we show how we applied A^{2}THOS to a practical case of service availability planning in an industrial context. Section 5.8 and Section 5.9 are two appendix sections presenting the proof of one of the theorems we introduce and the representation capabilities of our model with respect to RBDs respectively.

5.2 Related Work

In this section we discuss related works in four relevant areas for our problem: (1) the general approach to calculate system availability, (2) modelling techniques to represent the system under analysis, (3) existing tools and (4) other approaches taking into account availability to optimise IT service composition.

The general approach Referring to a classic formulation [13] taken from the reliability theory, a *repairable system* is a system which can be repaired after a failure.

In the simplest case, the system m for which availability must be determined is represented by the state function $\chi(m,t)$ which assumes value 1 if m is operating within tolerances at time t, 0 otherwise. The general way of calculating the availability of a repairable system is to assume it has an independent, exponential distribution of failure and repair time (a so-called stationary alternating renewal process [58]). However, to do so one must know at least two properties of the system: its failure rate λ, and its repair rate μ. The first property specifies how often the system will fail on average, i.e. its Mean Time Between Failure (MTBF): $\lambda = \frac{1}{MTBF}$. The second one specifies its Mean Time To Repair (MTTR): $\mu = \frac{1}{MTTR}$. Under this assumption the limiting availability is then obtained by the formula $\bar{A} = \frac{\mu}{\mu+\lambda}$.

In the general case, the system can assume more than two states. Such a system is called *complex*. A complex system is a system which is made of interconnected components that as a whole exhibit one or more properties depending on the properties of the individual component. For example, a complex system can be made of two "simple" components (i.e. two components that can independently be either in operative or in repairing state). The state of the system depends on the state of the two components: the system may work properly even if one component only is operative, or it may need both components to be operative. To model the state of the system, a state formula is used. Components can have more than two states (e.g. operative, planned maintenance, emergency repair, etc.). To compute the availability of complex systems, CTMC [68], or Petri Nets [33] are used. To employ such techniques, one has to (1) define a state formula of the system based on the component's state, and (2) know the transaction probability of each component from one state to the other.

In our case, the information available in the SLAs for outsourced components concerns only a minimal availability in a given time frame (e.g. one month). Therefore, classic techniques are not applicable to this problem, as the internal states of each component and the probability of state transition (i.e. failure and repair rate) are only known by the outsourcing company.

System modelling Several approaches have been proposed in the literature for system reliability modelling. FTs and RBDs are the most used ones. However, we should mention that also other approaches have been proposed, e.g. Torres-

Toledano and Sucar [75] use Bayesian networks, and Leangsuksun et al. [54] use an UML representation (although in this second case the authors do not provide the mathematical support for reliability analysis). In FTs, a number of components (called basic events) are linked together to make up a system according to AND/OR relationships. The same behaviour is achieved in RBDs through SERIES/PARALLEL compositions. According to [33], FTs are easy to use, as they do not require very skilled modellers, and relatively fast to evaluate, as it is possible to use very efficient combinatorial solving techniques to obtain most of the reliability indexes.

In FTs, the system state is represented by the top event, i.e. the root of the tree. It is possible to build a boolean equation from the FT, and to reduce it to the *minimal cut set*, i.e. the smallest set of combinations of basic events (component failures) which all need to occur for the top event to take place (system failure) [76]. Based on the minimal cut set, a combination of combinatorial techniques and CTMC or Petri nets is then used to calculate the system (limiting) availability.

According to Flamini et al. [33], the main limitation of FTs and RBDs consists in the lack of modelling power, as they do not allow to model maintenance-related issues explicitly. To solve this problem, FTs and RBDs have been extended into Dynamic FTs [30] and Dynamic RBDs [29], allowing one to model maintenance-related issues.

The modelling notation we use in this chapter (*AND /OR* dependency graphs) can be seen as a condensed form of FTs. With a single *AND /OR* dependency graph we are able to model a forest of FTs sharing (some of) the basic events (i.e. the failure of a component), but with different top events. A single *AND /OR* dependency graph can thus model separately the failure of all the business services which the IT system provides, and for which a specific availability level must be calculated. In fact, it is possible to (automatically) transform any *AND /OR* dependency graph into a forest of FTs, as well as in a set of RBD, as we show in Section 5.9. We share with FTs the use of minimal cut sets, which in our notation are called Dependency Sets (see Section 5.3), but the availability calculation we apply to *AND /OR* dependency graphs is different from the one used in FTs (for the reason we mentioned above). In Appendix A we will give a more detailed explanation of the differences between FTs and the *AND /OR* dependency graphs we use in A^{2}THOS.

Tools IBM Tivoli [100] and HP Business Availability Centre [98] are two of the most popular configuration management tools. These tools are meant to support IT managers in the configuration and maintenance of complex IT systems. Among the many features they possess, they can be used to manage SLAs, including availability levels. One can assign to each IT component the availability level imposed by SLAs, and keep track of the actual availability levels to check for SLA compliancy. However, to the best of our knowledge there is no support for the analytical calculation of the service availability.

Galileo [74], Coral [17], Relex [110] and BlockSim [88] are tools operating with Dynamic FTs. Although integrating the A^{2}THOS engines in one of these tools would be useful, this was not possible: Relex and BlockSim are commercial tools, Coral is mostly a MatLab library without a GUI, and Galileo is free software, but not open source. For these reasons we developed our prototype as an independent Java/Prolog tool.

Availability in service composition In the field of IT service composition, several approaches have been proposed that consider availability as one of the QoS parameters to optimise the performances of the resulting composite IT service. Gu et al. [35] propose QUEST, a framework to schedule dynamically a composite IT service while satisfying QoS requirements (e.g. response time and availability) imposed by SLAs. Zeng et al. [81], Yu et al. [80] and Ardagna et al. [9] propose scheduling techniques to create a cost-optimal execution plan for composite web services which respect QoS parameters (including availability) defined in SLA contracts.

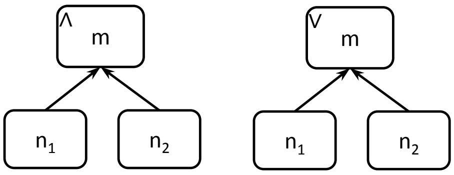
In all these works, an estimation of the availability of the composite service is made by multiplying the availability level of the components (expressed as a real number in the interval [0,1]). This is possible thanks to two simplifying assumptions. First, all the components must be available at the same time for the system to operate (i.e. the system is an AND-combination of its components and it becomes unavailable in the moment that any of its component is unavailable). Secondly, the resulting availability is not a lower bound, i.e. there can be a run of the composite service in which the resulting availability is lower than the calculated one. Differently from these approaches, A^{2}THOS is able to deal with a wider range of dependencies, namely combinations of *AND* and *OR* dependencies. In the sequel we also argue in more detail why *OR* dependencies are necessary to model complex IT services correctly. A^{2}THOS also allows one to calculate an absolute lower bound for the availability, which can be safely included in an SLA contract.

5.3 Analysis of the minimal service availability

We now present the theoretical foundations of A^{2}THOS. Let us first start with an intuitive explanation. We model the system using a AND /OR *dependency* *graph*, in which a node represents a component of the system that at any given time may (or may not) be available. A directed edge from node m to node n indicates that m *depends on* n, i.e. that the availability of m depends also from the availability of n in a way that we are about to explain.

In an *AND /OR* dependency graph, a node m can be unavailable because of an internal failure, or because (some) nodes it depends on are unavailable. To model internal failure, to each node m we associate a (virtual) *internal node* m'. We say that the internal node m' is unavailable if the node is unavailable because of an internal failure. Therefore, internal nodes are just a notation artefact with no other fundamental purpose than indicating the internal failure of a node.

To model the fact that m becomes unavailable because one or more nodes it depends on are unavailable, we then consider nodes of two types: *AND* and *OR* .



(a) *AND* (b) *OR*

Figure 5.2: Two simple *AND /OR* dependency graphs, respectively with *AND* and *OR* nodes. Each of these nodes can also be indicated as internal nodes (i.e. m' is the internal node of m).

If m is a node in an *AND /OR* dependency graph and $n_1, \ldots, n_k$ are the nodes m depends on, we say that

- m is *unavailable* at time t iff its internal node m' is unavailable at time t *or*
  - at least one node in $n_1, \ldots, n_k$ is unavailable at time t, in case m is an *AND* node.
  - $n_1, \ldots, n_k$ are *all* unavailable at time t, in case m is an *OR* node,

Formally,

Definition 5.1 (*AND /OR* dependency graph)**.** *An AND /OR* dependency graph $\langle N,\ E\rangle$ *is a directed and acyclic graph (DAG) where* N *is the set of nodes, and is partitioned in* AND-N *and* OR-N*, and* E *is the set of edges* $E \subseteq \{\langle u, v\rangle \mid u, v \in N\}$.

Given a graph $\langle N,\ E\rangle$, we call N' the set of the internal nodes of g; $N' = \{n'$ internal of $n \mid n \in N\}$.

Running example - Part 5.1. *In this example we analyse the availability of an IT system providing two IT services (`Service1` and `Service2`), and implemented by means of three applications (`App1`, `App2` and `App3`) running on five different servers (`Srv1`, `Srv2`, `Srv3`, `Srv4`, `Srv5`). `Service1` is implemented by `App1` and `App2` in such a way that the service goes off-line only when both applications are off-line (OR dependency). `Service2` is implemented by `App3`, and `App3` depends on `App2` to work properly. `App1` is a distributed application running on `Srv1`, `Srv2` and `Srv3` in such a way that it can operate only if both `Srv1` and either `Srv2` or `Srv3` are on-line. `App2` runs on `Srv3`, and `App3` runs in both `Srv4` and `Srv5` with a load-balancing mechanism, such that it can continue to operate even if one of them is off-line. Finally, the system is protected by the firewall `FW1`. According to this description, we build the* AND /OR *dependency graph $g = \langle N, E\rangle$ as follows:*
AND-N = $\{$`Service1, Service2, FW1 App1, App2, App3, Srv1, Srv2, Srv3, Srv4, Srv5`$\}$, OR-N = $\{$ `OR1, OR2, OR3` $\}$, *and* $E = \{$ $\langle$`Srv1, App1`$\rangle$, $\langle$`Srv2, OR2`$\rangle$, $\langle$`Srv3, OR2`$\rangle$, $\langle$`Srv3, App2`$\rangle$, $\langle$`Srv4, OR3`$\rangle$, $\langle$`Srv5, OR3`$\rangle$, $\langle$`OR2, App1`$\rangle$, $\langle$`OR3, App3`$\rangle$, $\langle$`App1, OR1`$\rangle$, $\langle$`App2, OR1`$\rangle$, $\langle$`App2, App3`$\rangle$, $\langle$`App3, Service2`$\rangle$, $\langle$`FW1, Service1`$\rangle$, $\langle$`FW1, Service2`$\rangle$, $\langle$`OR1, Service1`$\rangle$ $\}$.
To model the OR *dependencies correctly we added three virtual nodes to act as logical gates: `OR1`, `OR2` and `OR3`. These nodes do not correspond to any existing component of the system, and therefore they cannot fail by themselves. Similarly, also the two nodes representing services (`Service1` and `Service2`) correspond to system functionalities which cannot fail by themselves. Figure 5.3 shows the* AND /OR *dependency graph of our running example.*

In classic reliability theory, the internal state of a node is described by a random variable which assumes value 1, corresponding to the functioning state, and 0, corresponding to the failed state. Accordingly, the state of a system made of multiple components is described by a vector of random variables, each describing the state of one component. It is also possible to describe the state of a (sub)system with values 0 or 1 as a function of the states of the (sub)system components.

We represent these concepts by means of the *state function* χ. Given a node m, $\chi(m', t)$ is 0 iff m at time t suffers an internal failure, and is 1 otherwise. Similarly, $\chi(m, t) = 0$ indicates that the node m is unavailable at time t. As explained above, the *state function* of a node m is a function of the state of its internal node and the state functions of the nodes it depends on. This is formalised in the next definition.

Definition 5.2 (State Function)**.** *Let $g = \langle N, E\rangle$ be an* AND /OR *dependency*

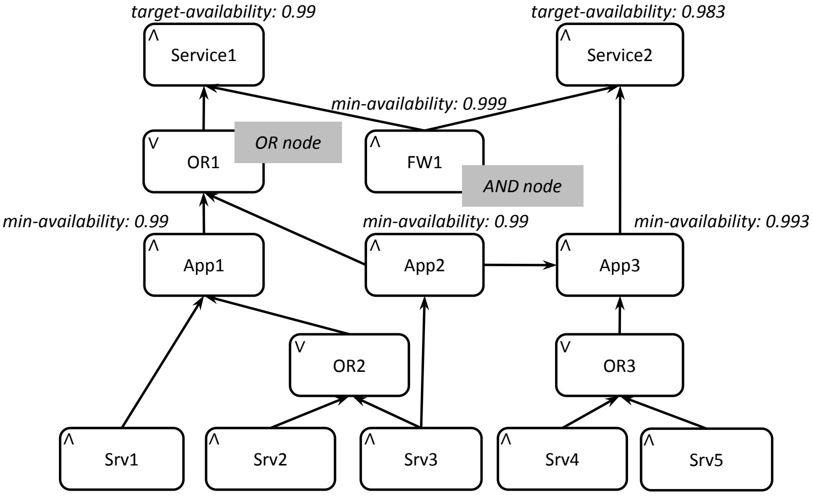


Figure 5.3: The *AND* /*OR* dependency graph representing the system we analyse in our running example. AND nodes are represented by the $\wedge$ symbol, OR nodes by the $\vee$ symbol.

graph. We say that χ *is a* state function *for* g *iff* $\chi: (N \cup N') \times \mathbb{R}^{+} \to \{0,1\}$*, and for each* $m \in N$ *and* $t \in \mathbb{R}^{+}$ *the following holds: let* $n_1, \ldots, n_k$ *be the nodes in* N m *depends on. Then*

$$\chi(m,t) = \begin{cases} \chi(m',t) \cdot \chi(n_1,t) \cdot \ldots \cdot \chi(n_k,t), & \textit{if } m \textit{ is an } \text{AND } \textit{node} \\ \chi(m',t) \cdot \max(\chi(n_1,t),\ldots,\chi(n_k,t)), & \textit{if } m \textit{ is an } \text{OR } \textit{node} \end{cases} \tag{5.1}$$

Using this function, we can represent the part of a time interval $[t_0, t_1]$ in which a given node is available as the infinite set $\{t \in [t_0, t_1] \mid \chi(m', t) = 1\}$.

So, given the state function of all the internal nodes in an *AND* /*OR* dependency graph, one can iteratively compute the state function of all the nodes in the graph (here the fact that the graph is acyclic guarantees that the above function is well defined). It should also be noted that $\chi(m,t)$ is a random variable, as a function of random variables.

According to the dependability theory [13], the *interval availability* of a node m is the fraction of a given interval of time that m operates within tolerances. Supposing the given interval of time is $[t_0, t_1]$, the formula of interval availability is given by:

$$\bar{A}(m, t_0, t_1) = \frac{1}{t_1 - t_0} \int_{t_0}^{t_1} \chi(m,t)dt \tag{5.2}$$

The *limiting interval availability*, or steady-state availability, is the expected fraction of time in the long run that the system operates within tolerances ($\lim_{t\to\infty} \bar{A}(m, t_0, t)$).

The formulas for availability we just described, however, are too general to perform numerical calculations. $\chi(m, t)$, being a random variable, will be governed by a distribution function. Therefore, in reliability theory, it is common practice to assume a stationary alternating renewal process, i.e. a system with independent, exponential distributions of failure and repair times. Under this assumption, the formula for the limiting interval availability is given by $\frac{\mu}{\lambda+\mu}$, where λ is the failure rate and μ is the repair rate. λ and μ can be estimated for a real system based on the MTBF and MTTR, and this allows one to compute the limiting interval availability for a real system (under the given assumptions).

In SLAs, the agreed minimal availability is always indicated as fraction of uptime *in a given time frame* (e.g. 0.98 uptime per month). Notice that the presence of the time frame is crucial: for instance, guaranteeing 0.99 uptime per month is more difficult than guaranteeing 0.99 uptime per year. In the first case the system may not be off-line for more than 7.2 hours in a row, while in the second case the system may be off-line for up to 87.6 hours in a row. Equation, (5.2) can be seen as a formalisation of the availability parameter used in SLAs.

The availability of a component is also given under the assumption that any other component it depends on is always available. For example, for server management, the SLA ensures a given availability level, provided that the data centre the server is deployed in and the network the server is connected to are operating within tolerances.

Now, the technical question we are going to address in the rest of this section, the answer of which will form the basis of our approach, is the following.

Let us fix a reference time interval $[t_0, t_1]$ *and suppose that we know a lower bound for the availability of the internal nodes of the nodes in a graph, i.e. for each* $n' \in N'$*, we know an* $\alpha_{n'}$ *such that the state function* χ *satisfies the following equation:*

$$(av(n') =) \frac{1}{t_1 - t_0} \int_{t_0}^{t_1} \chi(n', t) dt \ \geq \ \alpha_{n'} \tag{5.3}$$

where av(n') is the fraction of the time interval $[t_0, t_1]$ *in which* n' *was operating within tolerance. Given an arbitrary node* $m \in N$*, what can we say about av*(m) *in the same time period? In particular, can we compute a lower bound for it?*

Since in SLAs the components availability is given, in the context of this chapter we do not need to make assumptions on the distribution of component faults and repairs. This allows us not to define the distribution function that governs the internal state of components.

Dependency Sets

To answer the question above, we have to introduce the concept of dependency set. The dependency set of a node m is the set of the smallest sets of internal nodes in the *AND /OR* dependency graph which, if all unavailable at the same time, will cause the failure of m. The elements of a dependency set have the same property as the minimal cut sets of a FT, and can be obtained similarly by representing the graph as a boolean equation and the using substitution methods to reduce the equation. We will now present a more formal definition of dependency sets.

Definition 5.3. *Consider an* AND /OR *dependency graph* $g = \langle N, E \rangle$ *and a node* $m \in N$. *The* dependency set *of* m, $\text{DEPS}_m \subseteq \wp(\text{N'})$, *is defined inductively as follows.*

- *If* m *is a leaf node, then* $\text{DEPS}_m = \{\{m'\}\}$.
- *If* m *has children* $n_1 .. n_k$*; let* $\text{DEPS}_{n_1}, \ldots, \text{DEPS}_{n_k}$ *be the dependency set of* $n_1, \ldots, n_k$ *and assume (without losing generality) that for every* i, $\text{DEPS}_{n_i} = \{D_{i,1}, \ldots, D_{i,l_i}\}$, *then:*
  - *if* $m \in$ AND-N *then*
    $\text{DEPS}_m = \{\{m'\}\} \cup \bigcup_{i \in [1\ ..\ k]} \text{DEPS}_{n_i}$*;*
  - *if* $m \in$ OR-N *then*
    $\text{DEPS}_m = \{\{m'\}\} \cup \{D_{1,j_1} \cup \cdots \cup D_{k,j_k} \mid D_{i,j_i} \in \text{DEPS}_{n_i}\}$.

Running example - Part 5.2. *By applying the recursive Definition 5.3 to our example* AND /OR *dependency graph, we obtain the following dependency sets for* `Service1` *and* `Service2`*:* $\text{DEPS}_{\texttt{Service1}} = \{\{\textit{FW1'}\}, \{\textit{App1', App2'}\}, \{\textit{App1, Srv3'}\}, \{\textit{Srv1', App2'}\}, \{\textit{Srv1', Srv3'}\}, \{\textit{App2', Srv2', Srv3'}\}, \{\textit{Srv2',Srv3'}\}$. $\text{DEPS}_{\texttt{Service2}} = \{\{\textit{FW1'}\}, \{\textit{App2'}\}, \{\textit{App3'}\}, \{\textit{Srv3'}\}, \{\textit{Srv4', Srv5'}\}\}$. *For the sake of presentation we did not include in the dependency sets the internal nodes that cannot fail by themselves (i.e.* `Service1`$'$, `Service2`$'$, `OR1`$'$, `OR2`$'$ *and* `OR3`$'$*). It is easy to see that when the nodes of any of the elements of* $\text{DEPS}_{\texttt{Service1}}$ *are unavailable at the same time,* `Service1` *is unavailable, and the same for* $\text{DEPS}_{\texttt{Service2}}$.

The dependency set of a node is always a set of sets of internal nodes, so without loss of generality, we can always write $\mathit{DEPS}_m = \{D_1, \ldots, D_k\}$.

As for minimal cut sets in FTs, a relevant property of DEPS_m is that, if the internal m' of m is available at a given time t, then m is not available only if there exists DEPS_m such that at least all the internals of the nodes contained in one element D of DEPS_m are all unavailable. More formally, if we fix a time t then

$$\chi(m,t) = 0 \Leftrightarrow \exists D \in \mathit{DEPS}_m, \forall d \in D, \chi(d,t) = 0 \tag{5.4}$$

For the sake of presentation we skip the (straightforward) demonstration of this property.

As an example let us consider the two toy cases described in Figure 5.2. In case (a), *DEPS*$_m = \{\{m'\}, \{n'_1\}, \{n'_2\}\}$; so if $\chi(m,t) = 0$ and $\chi(m',t) = 1$, then either $\chi(n'_1, t) = 0$ or $\chi(n'_2, t) = 0$. In case (b), *DEPS*$_m = \{\{m'\}, \{n'_1, n'_2\}\}$; so if $\chi(m,t) = 0$ and $\chi(m',t) = 1$, then both $\chi(n'_1, t) = 0$ and $\chi(n'_2, t) = 0$.

The following theorem states that if we know a lower bound for the availability of the internal nodes of an *AND /OR* dependency graph then we can effectively compute an optimal lower bound of the availability of each node $m \in N$ in the graph. In the theorem we will also explain the meaning of an optimal availability lower bound.

Theorem 5.1. *Let $g = \langle N, E\rangle$ be an* AND /OR *dependency graph, $[t_0, t_1]$ be a time interval, and for each $n' \in N'$ let $\alpha_{n'}$ be a real value $\alpha_{n'} \in [0, 1]$. Then, for each $m \in N$ we can compute α_m, such that for each state function χ for g the following holds:*

$$\textit{IF } \forall n' \in N' \qquad \frac{1}{t_1 - t_0}\int_{t_0}^{t_1} \chi(n', t)dt \;\geq\; \alpha_{n'} \tag{5.5}$$

$$\textit{THEN} \qquad \frac{1}{t_1 - t_0}\int_{t_0}^{t_1} \chi(m, t)dt \;\geq\; \alpha_m \tag{5.6}$$

α_m is optimal if we can find a χ for g such that (5.5) holds and in (5.6) equality holds.

We provide the proof for this theorem in Section 5.8. As a result of the proof, we obtain a method to calculate the lower bound α_m of the availability of any node m in the graph. The method consists in solving the following linear programming problem.

$$\alpha_m = \begin{cases} \text{minimize } 1 - u_1 - \cdots - u_k \\ \text{subject to} \\ u_1 = (1 - a_{1,1}) = \cdots = (1 - a_{1,l_1}) \\ \vdots \\ u_k = (1 - a_{k,1}) = \cdots = (1 - a_{k,l_k}) \\ \forall n \in N, \sum_{d_{i,j} \in D_n} 1 - a_{i,j} \geq 1 - \alpha_{n'} \\ a_{1,1}, \ldots a_{1,l_1}, \ldots, a_{k,1}, \ldots, a_{k,l_k} \geq 0 \end{cases} \tag{5.7}$$

Intuitively, the availability of m is minimal when m is sequentially disrupted by the simultaneous failure of all the internal nodes of each element of *DEPS*$_m$. Without loss in generality, we can write *DEPS*$_m = \{D_1, \ldots, D_k\}$, and for each D_i

we can write $D_i = \{d_{i,i}, \dots, d_{i,l_i}\}$. According to this notation, u_i in (5.7) represent the unavailability caused to m by D_i and $a_{i,j}$ represent the availability of the element $d_{i,j} \in D_i$. Given two elements $D_1, D_2 \in$ *DEPS*$_m$, these two elements might not be pairwise disjoint (i.e. $D_1 \cap D_2 \neq \varnothing$) because some elements in D_1 and D_2 refer to the same node. In (5.7) we call D_n the set of elements $d_{i,j}$ which all refer to the same node n. The objective function of (5.7) represents the availability of node m, which is expressed as 1 less the unavailability caused by each element in *DEPS*$_n$. The first k conditions impose that the internal nodes of each element $D_i \in$ *DEPS*$_n$ are unavailable at the same time: this ensures m is disrupted because of the simultaneous failure of all the internal nodes in D_i. The subsequent condition imposes for each node $n \in N$ that the availability of its internal node n' is not less than $\alpha_{n'}$: in this way we ensure that, even if an internal node n' is contained in more than one element of *DEPS*$_n$, the unavailability caused by its failure will not exceed its upper bound $(1 - \alpha_{n'})$. The last condition ensures no negative value can be used to represent availability. A solution to (5.7) can be found by using the simplex algorithm.

From now on, we call α_m determined from (5.7) the *minimal aggregated availably level* of m.

Running example - Part 5.3. *According to the dependency sets we previously determined and to (5.7), the linear programming problem which determines* $\alpha_{\mathtt{Service1}}$ *is:*

$$\begin{cases} \textit{minimize } 1 - u_1 - u_2 - u_3 - u_4 - u_5 - u_6 - u_7 \textit{ subject to} \\ u_1 = (1 - a_{\mathtt{FW1},1}) \\ u_2 = (1 - a_{\mathtt{App1},2}) = (1 - a_{\mathtt{App2},2}) \\ u_3 = (1 - a_{\mathtt{App1},3}) = (1 - a_{\mathtt{Srv3},3}) \\ u_4 = (1 - a_{\mathtt{Srv1},4}) = (1 - a_{\mathtt{App2},4}) \\ u_5 = (1 - a_{\mathtt{Srv1},5}) = (1 - a_{\mathtt{Srv3},5}) \\ u_6 = (1 - a_{\mathtt{App2},6}) = (1 - a_{\mathtt{Srv2},6}) = (1 - a_{\mathtt{Srv3},6}) \\ u_7 = (1 - a_{\mathtt{Srv2},7}) = (1 - a_{\mathtt{Srv3},7}) \\ 1 - a_{\mathtt{FW1},1} \geq 1 - \alpha_{\mathtt{FW1}'} = 0.001 \\ (1 - a_{\mathtt{App1},2}) + (1 - a_{\mathtt{App1},3}) \geq 1 - \alpha_{\mathtt{App1}'} = 0.01 \\ (1 - a_{\mathtt{App2},2}) + (1 - a_{\mathtt{App2},4}) + (1 - a_{\mathtt{App2},6}) \geq 1 - \alpha_{\mathtt{App2}'} = 0.005 \\ (1 - a_{\mathtt{Srv1},4}) + (1 - a_{\mathtt{Srv1},5}) \geq 1 - \alpha\mathtt{Srv1}' = 0.001 \\ (1 - a_{\mathtt{Srv2},6}) + (1 - a_{\mathtt{Srv2},7}) \geq 1 - \alpha_{\mathtt{Srv2}'} = 0.001 \\ (1 - a_{\mathtt{Srv3},3}) + (1 - a_{\mathtt{Srv3},5}) + (1 - a_{\mathtt{Srv3},6}) + (1 - a_{\mathtt{Srv3},7}) \geq 1 - \alpha\mathtt{Srv3}' = 0.01 \\ a_{\mathtt{FW1},1}, a_{\mathtt{App1},2}, a_{\mathtt{App1},3}, a_{\mathtt{App2},2}, a_{\mathtt{App2},4}, a_{\mathtt{App2},6}, a_{\mathtt{Srv1},4}, \\ a_{\mathtt{Srv1},5}, a_{\mathtt{Srv2},6}, a_{\mathtt{Srv2},7}, a_{\mathtt{Srv3},3}, a_{\mathtt{Srv3},5}, a_{\mathtt{Srv3},6}, a_{\mathtt{Srv3},7} \geq 0 \end{cases}$$

Which gives a lower bound for the availability of `Service1` *of 0.984. Similarly, we can determine* $\alpha_{\mathtt{Service2}} = 0.972$. *Figure 5.4 shows one possible scheduling* *for the failure of the components on which* `Service1` *depends on, resulting in* `Service1` *having an availability of* $\alpha_{\texttt{Service1}}$ *(0.984).*

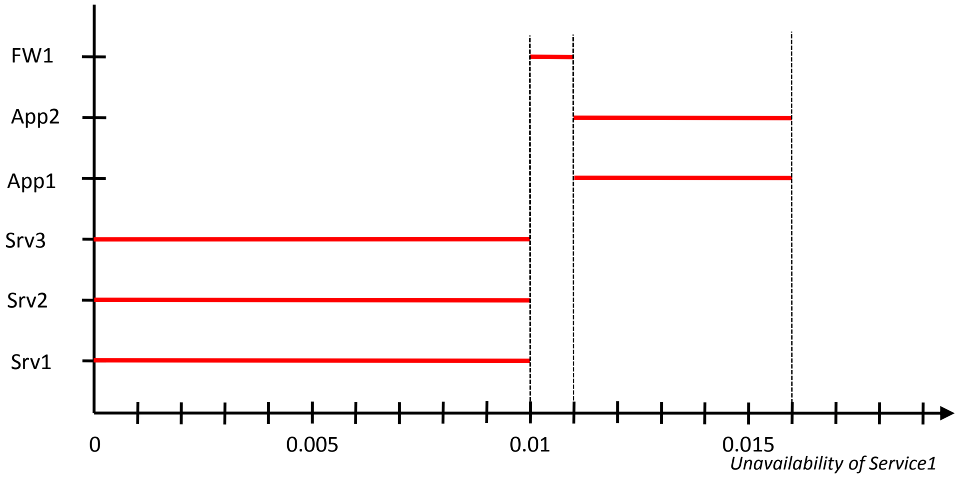


Figure 5.4: One possible scheduling for the failure of `FW1`, `App1`, `App2`, `Srv1`, `Srv2` and `Srv3` resulting in `Service1` having an availability of 0.984. System components are on the vertical axis and the components unavailability fraction of time ($\in [0, 1]$) is on the horizontal axis.

5.4 Optimisation of outsourced services

In the last section we have seen that determining the minimal availability level of a complex system is a non-trivial problem, that can be solved by reducing it to an optimisation problem. A relevant application of this result is the minimisation of the costs of outsourced subcomponents. Given that outsourcing has a cost that may depend (also) on the minimal availability guaranteed for the outsourced component, a manager typically needs to minimise the costs of the outsourcing while guaranteeing that the services provided by the system meet the target availability.

The situation is the reverse from the one in the previous section: instead of *calculating* the service minimal availability *given* the minimal availability of the various system components, one wishes to calculate what is the least expensive combination of components given the target minimal availability of the services. Thus, availability level optimisation consists in determining the assignment of an availability level to the components of the system for which it is possible to choose among different availability levels, so that:

1. a *minimal aggregated availability level* is ensured for the services provided by the system;

2. the cost of the assignment is minimal.

To this end we distinguish among three types of nodes in an *AND /OR* dependency graph: *target* availability nodes, *variable* availability nodes and *given* availability nodes. More formally, given an *AND /OR* dependency graph $\langle N, E\rangle$, $N = N_T \cup N_V \cup N_G$, where N_T, N_V and N_G are the pairwise disjoint sets of target, variable and given availability nodes.

Target availability nodes are the nodes modelling the services provided by the system. The target expresses the minimal availability level which the system is or should be able to guarantee regarding a given functionality (service). Typically, we define a target availability level on the service nodes of the *AND /OR* dependency graph, whenever there is an SLA (be it company internal or not) which imposes a certain level of availability for them.

Definition 5.4 (Target availability level)**.** *Given an* AND /OR *dependency graph* $\langle N, E\rangle$*, and* $N_T \subset N$ *the set of target availability nodes, the target availability level of a node is a mapping* target-availability : $N_T \to [0, 1]$.

Running example - Part 5.4. *Our example system provides two main functionalities, described in the* AND /OR *dependency graph by the* `Service1` *and* `Service2` *nodes. The functionality described by* `Service1` *is more mission-critical than the one described by* `Service2`*, and an SLA set on the system ensures a minimal availability level of 0.99 for* `Service1` *and of 0.983 for* `Service2`*. Accordingly,* target-availability(`Service1`) = 0.99 *and* target-availability(`Service2`) = 0.983.

Variable availability nodes model the situation in which it is possible to choose the availability level of a component among different options. A typical case of *variable* availability level is when the management of system components is outsourced to another department or to another company: in these cases, the system manager may have the possibility to choose for each component a different availability level (e.g. gold, silver and bronze), with different quality level and a different associated price.

We model the domain of a variable availability level by means of a set of *availability options*.

Definition 5.5 (Availability option)**.** *Let* $\langle N, E\rangle$ *be an* AND /OR *dependency graph,* N_V *be a set of variable availability nodes and* N'_V *be the set of the internals of the elements in* N_V*.*

- *An* availability assignment *for N_V is a function* aa : $N'_V \rightarrow [0,1]$.
- *An* availability option *for N_V is a pair $\langle \mathrm{aa}, c \rangle$, where* aa *is an availability assignment and $c :\in \mathbb{R}$ is the cost associated to it. We call O the set of all the availability options.*

Usually, a company might choose between different outsourcing options. Notice that the outsourcing option is given on a set of nodes, and not on a single node. This allows us to model bulk discounts, i.e. the fact that outsourcing – say – ten components is usually less expensive than ten times the outsourcing of a single components. (For those who are familiar with optimisation problems: this introduces a form of non-monotonicity in the outsourcing offers, where outsourcing more services could be potentially less expensive than outsourcing a smaller number of services. This non-monotonic aspect of the problem makes it more difficult to find the optimal solution.)

Table 5.1: An example of availability level options price catalogue

Availability level	*Minimal quantity*	*Windows price*	*UNIX price*
0.99	1 server	1000 Euro	900 Euro
0.99	6 servers	900 Euro	800 Euro
0.995	1 server	1300 Euro	1200 Euro
0.995	11 servers	1200 Euro	1100 Euro
0.998	1 server	1500 Euro	1400 Euro
0.998	6 servers	1400 Euro	1300 Euro

Running example - Part 5.5. *An outsourcing company has provided an offer for managing the servers in our example system. The offer includes different availability level options for the management of Windows servers and UNIX servers. Table 5.1 summarises the price catalogue for this offer. In our example* `Srv1`*,* `Srv2` *and* `Srv3` *are Windows servers with variable availability and both* `Srv4` *and* `Srv5` *are UNIX servers with variable availability. The set of variable availability nodes is $N_V = \{\texttt{Srv1}, \texttt{Srv2}, \texttt{Srv3}, \texttt{Srv4}, \texttt{Srv5}\}$. According to the price catalogue, there are* `number of availability levels` *at the power of* `number of variable availability nodes` *($3^5 = 243$) possible combinations for the minimal availability level of the elements in N'_V, i.e. $|O| = 243$. One of these combinations is $\langle \mathrm{aa}_1, 4800 \rangle$ where $\mathrm{aa}_1(\texttt{Srv1}') = \mathrm{aa}_1(\texttt{Srv2}') = \mathrm{aa}_1(\texttt{Srv3}') = \mathrm{aa}_1(\texttt{Srv4}') = \mathrm{aa}_1(\texttt{Srv5}') = 0.99$, and $\mathrm{c} = 3 \cdot 1000 + 2 \cdot 900 = 4800$.*

To guarantee that the condition 1) of our problem is met, we extend the definition of minimal aggregated availability to be applied also to nodes with a variable availability. According to this, given a node $t \in N_T$ with target availability, we call *minimal-availability*(t, o) the minimal aggregated availability of t when for all nodes n with variable availability $\alpha_{n'}$ is determined by o.

Finally, a node with *given* availability model components for which the minimal availability is known and not variable.

Definition 5.6 (Given availability level)**.** *Given an* AND /OR *dependency graph* $\langle N, E\rangle$*, and* $N_G \subset N$ *the set of target availability nodes, the given availability level of a node is a mapping* given-availability : $N_T \rightarrow [0, 1]$.

Running example - Part 5.6. *In our example the components whose minimal availability is given are* `FW1`*,* `App1`*,* `App2` *and* `App3`*. Therefore, according to Figure 5.3 we have that* given-availability(`FW1`) $= 0.999$*,* given-availability(`App1`) $= 0.99$*,* given-availability(`App2`) $= 0.99$ *and* given-availability(`App3`) $= 0.993$*.*

We now can give a more formal definition of the problem at hand. Let $\langle N, E\rangle$ be an *AND /OR* dependency graph, $N = N_T \dot\cup N_V \dot\cup N_G$, a function *target-availabiliy* on N'_T, a function *given-availability* on N'_G, a set O of availability options for N_V and a function *minimal-availability* for $N_T \times O$. Find the option $o \in O$ with minimal cost such that $\forall t \in N_T$, *minimal-availability*$(t, o) \geq$ *target-availability*(t).

As a result, we obtain the optimal assignment of variable availability levels as the solution to the linear programming problem with variables in a finite domain given by (5.8).

$$\begin{cases} \text{choose } o = \langle aa, c\rangle \in O \text{ to} \\ \text{minimize } c \\ \text{subject to:} \\ \textit{mimimum-availability}(t_1, o) \geq \textit{target-availability}(t_1) \\ \vdots \\ \textit{minimal-availability}(t_P, o) \geq \textit{target-availability}(t_P) \end{cases} \tag{5.8}$$

Running example - Part 5.7. *Recall that the nodes with variable availability are* `Srv1`*,* `Srv2`*,* `Srv3`*,* `Srv4` *and* `Srv5`*. The set of availability options O is made of 243 elements. We want to ensure that the lower bound of the monthly availability is 0.99 for* `Service1` *and 0.983 for* `Service2`*. Consequently, the optimisation problem is as follows:*

$$\begin{cases} \textit{choose } o = \langle \text{aa}, c \rangle \in O \textit{ to} \\ \textit{minimize } c \\ \textit{subject to:} \\ \text{minimal-availability}(\texttt{Service1}, o) \geq 0.99 \\ \text{miniaml-availability}(\texttt{Service2}, o) \geq 0.983 \end{cases}$$

Which gives us a (optimal) solution with cost 6300 Euro *when* $\alpha_{\texttt{Srv1}'} = 0.998$*,* $\alpha_{\texttt{Srv2}} = 0.99$*,* $\alpha_{\texttt{Srv3}} = 0.998$*,* $\alpha_{\texttt{Srv4}} = 0.99$ *and* $\alpha_{\texttt{Srv5}} = 0.998$*.*

5.5 Implementation and benchmarks

Implementation We have implemented a prototype of A²THOS to run our lab experiments and to support case-studies. The prototype is written in Java and prolog in about 10,000 lines of code. We chose to use the ECLiPSe [95] prolog platform since it provides a flexible yet powerful set of constraint solvers which we need to deal with the linear programming problems of A²THOS. The available solvers include *fd*, a solver for finite domain integer problems, *ic* a solver for hybrid integer/real-interval problems and *eplex*, an interface to an (external) simplex solver library.

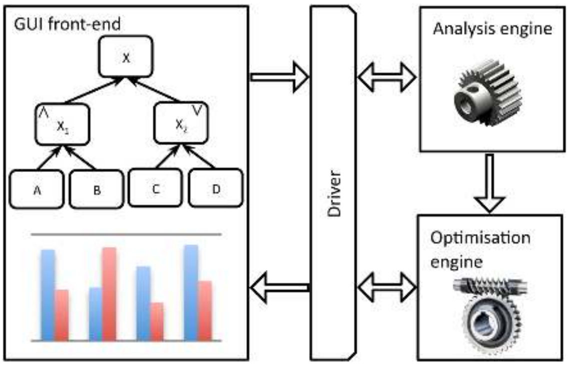


Figure 5.5: A²THOS architecture

Figure 5.5 shows the software architecture of our prototype. It consists of four interacting components: the GUI front-end, the driver, the analysis and the optimisation engines. The GUI front-end manages the interaction with the final user. It is implemented as a standalone Java application and it allows the user to quickly create the *AND /OR* dependency graph by dragging and dropping nodes and edges, to annotate each node with its availability figure(s) or availability level options and to view the analysis and optimisation results. The analysis engine solves the availability analysis problem, described in Section 5.3. It is implemented in prolog by using the *eplex* (simplex algorithm) solver of the ECLiPSe platform. The optimisation engine solves the availability optimisation problem, which we describe in Section 5.4. It is also implemented in prolog by using the *fd* solver of the ECLiPSe platform. Finally, the driver is written in Java and manages the interaction of the Java components with the prolog ones. It uses the JavaECLiPSe interface to build a prolog optimisation problem from the *AND /OR* dependency graph and the other availability-related information inserted by the user. It then translates the results given by the engines in a format that can be presented to the user by the Java GUI front-end.

Benchmarks To be of practical use, our prototype needs to deliver a solution to the linear programming problems in a reasonable time. Unfortunately, the simplex algorithm has a worst-case exponential complexity [72], and solving by brute-force linear programming problems with variables in a finite domain has an exponential complexity in the number of variables and their domain size. This means that the implementation does not scale, and therefore we have to benchmark whether it can tackle the size of a real-world IT system. In the sequel we show that it does so, nevertheless we want to stress that our implementation is just a proof of concept and its speed can with no doubt be improved: our goal is to demonstrate how this can be done, and not that of providing a fast implementation.

We benchmarked the performance of our prototype by running it on inputs with growing size. We run our test on a machine with an Intel Pentium 4 CPU running at 3.6 GHz and with 2 GB RAM.

First, we benchmarked the availability analysis. Here, the complexity of the simplex algorithm is determined by the number of variables and constraints of the linear programming problem it solves. Therefore, we generate inputs for the analysis engine by increasing the number of nodes and by adjusting the node types and edges to obtain a growing number of constraints (and associated variables) for the linear programming problem of (5.7). We set the maximum number of nodes for our tests to 250 and the maximum number of constraints to 600. In our experience these numbers correspond to a fairly large IT system. To increase randomness we also repeat several times (five) the test for a certain number of nodes and constraints, and we then calculate the average computation time. The results are shown in Table 5.2. Our tests indicate that given a fixed number of constraints, the computational time is basically linear in the number of nodes, and that our prototype is able to handle an *AND /OR* dependency graph of 250 nodes and 600 constraints on average in less than a minute.

Table 5.2: Performance of the simplex algorithm for availability analysis

Nodes	*Constraints*	*Time (s)*
15	10	0.00001
15	20	0.002
60	10	0.001
60	20	0.005
60	60	0.02
120	10	0.004
120	20	0.01
120	100	0.09
120	150	0.22
120	250	0.8
120	300	1.3
120	600	20.2
250	10	0.009
250	20	0.011
250	100	0.22
250	150	0.5
250	250	1.6
250	300	2.6
250	600	41.1

Secondly, we benchmarked the optimisation algorithm. Our prototype implementation works by exhaustive searching the space of all available options and choosing the best one. The algorithm is thus optimal (it finds the best solution, every time), but its complexity is exponential in the number of variables (which in this case corresponds to the number of nodes with variable availability). Again, the fact that the algorithm is exponential means that we cannot expect it to scale up indefinitely, and it is therefore important to assess via benchmarks how big of a problem it is able to tackle.

We carried out these benchmarks as follows: we create a simple program which takes as input the desired number of nodes with availability level options, the average number and the average size of the dependency sets and generates a random *AND* /*OR* dependency graph with random availability level options which match the given parameters. We set three possible availability levels for nodes with variable availability, since that is the most common configuration in outsourcing scenarios (gold, silver and bronze). We then solve the problem with our optimisation engine and note the execution time. We use an increasing number of nodes with variable availability (up to 50) and we specify different average number and average sizes of the dependency sets. We repeat several times (five) the test for each configuration in order to increase randomness.

Table 5.3: Performance of the availability optimisation algorithm with 50 variable availability nodes

Independent nodes	*Time (s)*
10	≤ 0.01
15	0.01
20	197.20
25	5418.50
30	≥ 21600.00

Our results indicates that the computational time is mostly influenced by the number of independent nodes. By independent node here we mean a node that appears in only one element of a dependency set. We report in Table 5.3 the results of our tests with 50 nodes. As the number of independent nodes increases, the computational time increases as well. We are able to solve a problem with 50 nodes among which 25 independent in one hour and a half. However trend is – as expected – exponential, and with 30 independent nodes we exceed six hours of computation. This is due to the fact that the solver has to explore all the possible combinations of values for the variables associated to independent nodes, while the domain of the other variables is limited by the problem constraints.

Our benchmark indicates that the crucial factor influencing the computation time is the number of independent nodes (outsourced components) which contribute independently to the system availability (*AND* dependency), and that the algorithm as it is now is always able to handle situations with up to 25 such nodes. In practice, this number is sufficient to model a single medium/large IT system, as we will show in the next section. It is worth noting that one can break a huge IT system into independent subsystems and apply the algorithm to them one by one. In this light, 25 outsourced components represent a limit which is basically never exceeded (in our industrial test case, which was carried out at a multinational company, we had a maximum of 6 independent nodes).

In the unlikely case that one would need to apply the algorithm to a too large system (e.g. exceeding the 40 AND-independent), one could still refer to the optimisation problem we have reported in Section 5.4, but then use a non-exhaustive algorithm to find a solution to it. Non-exhaustive algorithms (e.g. those based on *local search* [31]) have the disadvantage that they do not guarantee finding the optimal solution (they usually find a local optimum, which is not guaranteed to be a global optimum as well), but could probably easily scale to hundreds of independent nodes.

5.6 Methodology - practical use of A^{2}THOS

In this section we present a case-study we carried out on the IT infrastructure of a large multinational company. With this case-study we want to address three important questions regarding A^{2}THOS:

1. can A^{2}THOS be applied to a practical case, i.e. does it not require information not available in practice?
2. does the prototype we implemented scale up to conditions of practice?
3. does A^{2}THOS yield information useful for its intended users?

Let us now present the context in which we carried out the case-study. The multinational company (from now on we call it the Company) has a global presence in over 50 countries and counts between 100.000 and 200.000 employees. Our case-study was conducted at the site of IT facilities for the Company's European branch.

The Company IT department supports the business of hundreds of other departments by offering thousands of applications accessed by approximately one hundred thousands employee workstations and by many hundreds of business partners. IT services are planned, designed, developed and managed by the IT infrastructure department located at the Company's headquarters. These services (e.g. e-mail or ERP systems) are part of the IT infrastructure which is used by all the different Company's branches all over Europe.

For efficiency reasons, like in most other large organisations, business units exchange services by means of an "enterprise internal market". One business unit pays another one for the use of a given service and the service provider unit finances its activities by means of these funds. Within this "internal market", the quality of the provided services is regulated by means of SLAs. Among the other Quality of Service (QoS) parameters, SLAs include the *minimal ensured availability* of the offered services.

IT services are designed internally by the IT department and then partly outsourced for implementation and management to another company. We call this company the Outsourcer. The Outsourcer is a market-leading international IT services provider. The outsourced tasks include application and server management, help-desk and problem solving. Although the servers running the IT services are owned by the Company and physically kept within its data centres, the Outsourcer manages the OS and the software running on them. The Outsourcer has signed contracts with the Company which include SLAs regarding both the security of the information managed by the outsourcing company and the availability of the outsourced services.

The Company and the Outsourcer have established a standard contract regulating the application and server management service provisioning. The standard contract is made of several *building blocks*, e.g. UNIX server management, Windows server management or Oracle database management. Every time the IT department of the Company needs to deploy a new IT service, a new request is issued to the Outsourcer to provide the building blocks needed by the service. The QoS parameters of each building block are also standardised. Regarding availability, for each building block the Company can choose among different guaranteed minimal availability levels. The price for the provisioning of each building block with specific QoS parameters is part of the price catalogue of the Outsourcer.

One of the problems the IT infrastructure managers of the Company have to deal with is how to determine the minimal availability level of new IT services. In fact, this availability level is meant to be used to set up the Company internal SLAs between the service provider (the IT department) and the service users (the other departments of the Company). It is important that the IT infrastructure manager is as precise as possible in determining the minimal availability level to be agreed with the internal users. In fact, a too low value could prevent the agreement to be reached, as the service users may not be willing to pay for a service which does not fit their needs in terms of availability. On the other hand, a too high value may impact the budget of the IT department, as for each time that the SLA is not respected the department has to pay a penalty. Ultimately, if the SLAs are violated too many times the service users may decide to terminate the service delivery contract before the IT department has compensated the initial service establishment costs. The reverse problem faced by the IT infrastructure manager is: if the service user has a specific requirement for the availability of the service, which QoS levels should be agreed with the Outsourcer for the outsourced building blocks such that the resulting internal service availability level meets the user requirements?

As we said, traditional approaches to the availability analysis are not quite applicable to this context. In fact, traditional availability analysis of complex systems require the analyst to know for each system component the Mean Time To Failure (MTTF) and Mean Time To Recovery (MTBR) parameters. The personnel of the IT department can measure (or estimate) these parameters for the portion of the IT system which is under its direct control. However, it cannot do this for the parts (a large majority) that are managed by the Outsourcer. The only information the IT manager can rely on for outsourced components is the guaranteed minimal availability level agreed with the Outsourcer. Therefore, the IT infrastructure manager currently estimates the service availability levels based on simple heuristics (e.g. if he need 0.99 availability for the service, he will choose at least 0.99 availability for each building block).

In our case-study we addressed this problem using the A^{2}THOS framework. We structured the case-study in two sub-cases. In the first sub-case we carried out the availability analysis of an IT system which is already in place for some years. In the second sub-case we carried out both the availability analysis and the optimisation for a new IT system which is about to be deployed. Our results have been used by the IT manager both to set the internal SLAs for the new service and to choose the proper availability level of the building blocks of the system.

In the first sub-case, the IT system we analysed is the authorisation and authentication system of the Company, called Oxygen. To carry out the availability analysis we first needed to represent Oxygen as an *AND /OR* dependency graph. We extracted the information from the network diagram, the functional specification document, and the security architecture and design document. The procedure we followed is described in Chapter 3. We used our tool to represent the *AND /OR* dependency graph of the system and to annotate nodes with their minimal availability level. The resulting graph consists of 65 nodes and 112 edges. Among the nodes are 13 IT services, 32 applications, 14 servers equally distributed between 2 datacenters and connected simultaneously to 2 different network segments by means of 2 different firewalls.

The second step of this sub-case is to determine the minimal monthly availability for the nodes in the graph. We extracted this information from the SLA documentation attached to the standard contract signed between the Company and the Outsourcer. Finally, we extracted the current minimal monthly availability of the IT services supported by Oxygen from the Company internal SLAs documentation.

We used the analysis engine of our tool to carry out the availability analysis: the whole algorithm completed in less than one minute for Oxygen.

Table 5.4 reports the results of our analysis. We report in the first column the (anonymised) service, in the second column the minimal monthly availability level of each service calculated with our tool and the existing minimal monthly availability level reported in the internal SLAs in the third column. Compared to the estimates made by the Company IT manager, we observe that the internal SLA specifies for Service1, Service4, Service5 and Service10 a minimal availability level which could not be guaranteed even in the case when the Outsourcer respects all its SLAs with the Company. This is a possible risk for the IT manager for the reasons we discussed above. On the other hand, we also see that the min-

Table 5.4: Results of the availability analysis on Oxygen

Service	*Calculated* α	*Existing* α
Service1	0.96	0.99
Service2	0.98	0.98
Service3	0.98	0.98
Service4	0.96	0.98
Service5	0.97	0.99
Service6	0.99	0.98
Service7	0.99	0.98
Service8	0.99	0.98
Service9	0.99	0.98
Service10	0.96	0.98
Service11	0.99	0.98
Service12	0.99	0.98
Service13	0.99	0.98

imal monthly availability level we calculated for Service6, Service7, Service8, Service10, Service11, Service12 and Service13 is higher than the one specified in SLAs. This is also a criticality for the IT manager, as he is spending more money than needed to guarantee the availability level of the outsourced Oxygen building blocks.

The system we analysed in the second sub-case is called Hydrogen and provides similar functionalities as Oxygen, but for the Company external contractors. Hydrogen has been designed after Oxygen and is now in the final development phase. In this phase, the internal SLAs with the Hydrogen service users are already set, and the Company IT manager has to issue a request to the Outsourcer for the building blocks to deploy Hydrogen. He also has to specify in the request the desired availability level for each building block. Therefore, in this second phase of our case-study we use the availability level optimisation of the A^{2}THOS framework.

The first step of this sub-case is the same as in the previous case: building the *AND /OR* dependency graph. To carry out this step we follow the same procedure we adopted for Oxygen (and described in more detail in Appendix B). The resulting graph is made of 26 nodes and 33 edges. Secondly, we annotated the nodes with given availability. These nodes represent the datacenters and the network segments. We acquired this information from the IT department personnel, which keeps track of the monthly availability performances of their main IT infrastructure components. We set the given availability as the lowest monthly availability value observed in the monitoring data. Finally, we extracted the availability options for the eight variable availability nodes (servers). We obtained the required information from the building block description documents and the price catalogue provided by the Outsourcer to the Company. According to these documents, the Outsourcer offers three availability levels for the six Unix servers (0.995, 0.998 and 0.999) and two levels for the two Windows ones (0.995, 0.998). The resulting number of availability options is 733.

We used our tool to obtain the optimal configuration of availability levels for the servers of Hydrogen: the whole algorithm completed in less than one minute for Hydrogen. If the Company IT managers adopted the same strategy chosen for Oxygen (i.e. to choose the lowest availability level for all the outsourced components), they would have spent as little as possible, but two services of Hydrogen would have had a minimal availability lower than the one set in the internal SLAs. The optimal combination computed with our tool ensures that the minimal availability is compliant with the SLAs for all the services, with a cost which is only 2% greater. We also considered the effect of adding a further availability level (0.990) to the price catalogue of the outsourced components, extrapolating the cost from the existing ones. The resulting optimal allocation in this case would be ~30% lower. The IT managers will take their decisions based on these results. In more detail, they will choose the optimal allocation we computed for Hydrogen, and will negotiate with the Outsourcer the introduction of a new availability level of 0.990 for servers.

Let us now address the questions we listed at the beginning of this section.

Can the method be applied to a practical case? The fact that we were able to successfully carry out the case-study presented above supports a positive answer to this question. However, it only shows that *we* can use the method. Our application has not revealed obstacles to usage by *other* people, but further evidence would be needed to substantiate this claim positively. It is also interesting to discuss which other contexts A^{2}THOS can be applied to. The information required to use A^{2}THOS can be summarised in (1) the components of the IT system under analysis and their functional dependencies, (2) the minimal availability level of each system component and (3) the different availability levels which can be chosen for outsourced components and the associated cost. To use A^{2}THOS in other contexts, these three pieces of information must be available. We learnt from this case-study that most of the information regarding (1) can be extracted from the system functional and design documentation. The lacking parts can be easily integrated by interviewing the technical personnel which designed or implemented the system. Information regarding (2) is normally present in the SLA documentation for outsourced components. For components which are managed internally, this information can be extracted by measuring the component's performance over time (as it was done by the Company in this case.) It is also possible to calculate availability levels analytically, by using standard reliability techniques, as we mentioned in Section 5.2. Information regarding (3) is only available if the IT service provider allows its customers to choose among different availability options. Although some IT service providers do not provide this feature, we learnt during our case-study that the Outsourcer is applying this strategy to all of its customers. Therefore there are indications that say that A^{2}THOS is also applicable (at least) to these customers.

Does the prototype we implemented scale up to conditions of practice? In this case-study we applied A^{2}THOS to two distributed systems which are used by a large multinational company. The size of these systems is comparable to the size of the other systems the Company is using. The first scalability issue regards the time needed to build an *AND /OR* dependency graph for these systems. As we already argued above, the information required is available, but building an accurate *AND /OR* dependency graph for an IT manager can be time consuming. As the size of the system grows, the difficulty of choosing an (close to) optimal combination of availability levels for outsourced components grows more rapidly than the difficulty of building the graph. This suggests that it may be worth using A^{2}THOS for large systems, whose optimum component availability level combination is hard to find. Secondly, the case-study confirms that our prototype can tackle large IT systems. We motivate this statement by the following two observations. First, the IT manager(s) of the Company make decisions about the availability of outsourced system components for each new system introduced in the IT infrastructure. In other words, the unit for the decision of the IT manager is limited to one system at a time. This is not surprising, if we consider that an organisation's IT infrastructure is incrementally built following the needs of the organisations: every time a new system is added to the infrastructure, only the availability issues of that system are taken into account. Secondly, the size (in terms of number of components) of the IT systems we analysed in our case-study is comparable to the size of the other systems in the Company's IT infrastructure. We expect to find the same system size in other (large) organisations as well. According to this two observations, we can argue that the performances of our prototype are sufficient in many practical situations, even with an IT system up to three times larger than the ones we considered.

Does A^{2}THOS yield information useful for its intended users? The feedback we had from the IT management of the Company suggests a positive answer to this question. In particular, they found the information useful for: (1) taking informed decision about the planning of the availability of their IT services, (2) improve the quality of the IT services provided to the business units of the Company and (3) justify to the upper management the outsourcing costs in a more precise way. This provides support for our claim of potential usefulness of A^{2}THOS for its intended contexts of application.

5.7 Concluding remarks

In this chapter we address availability planning from the angle of SLM. To this end, we present A^{2}THOS, a framework for the analysis and optimisation of the availability of mixed-sourced IT services. The framework consists of (1) a modelling technique to represent partially-outsourced IT systems, their components and the services they provide, based on AND /OR *dependency graphs*, (2) a procedure to calculate (a lower bound of) the system availability given the (lower bounds of) components availability, and (3) a procedure to select the optimum availability level for outsourced components in order to guarantee a desired target availability level for the service(s) and to minimise costs. The engine we have used is a proof-of-concept, and its speed can certainly be improved. This is however outside the scope of this work.

We have analysed the SLAs of a few organisations and we concluded that in case of outsourcing, these SLAs were sub-optimal: the final availability levels were achievable with less expensive means (in some cases, even better availability levels were achievable at a lower cost). This is not surprising as the optimisation of the SLAs is a non-trivial problem which in practice is “solved” by educated guess by the chief IT officer. We have shown in this chapter that this problem can also be tackled effectively using our modelling framework. Our benchmarks show that – even though the underlying problem is exponential – A^{2}THOS can tackle IT systems which are three times larger than the ones we could find at a multinational company. Given that the optimisation of the SLAs can save (immediately) company money while leaving the global service level unchanged, we believe that out framework could be profitably applied in practice.

5.8 Proof of Theorem 5.1

Proof. Assuming without loss in generality that $\mathit{DEPS}_m = \{D_1, \ldots, D_k\}$ and $D_i = \{d_{i,1}, \ldots, d_{i,l_i}\}$ we see that:

$$\chi(m,t) = \prod_{i \in [1 \ .. \ k]} (\mathit{max}_{j \in [1 \ .. \ l_i]} \chi(d_{i,j}, t)) \tag{5.9}$$

and calling $\chi(D_i, t) = \mathit{max}_{j \in [1 \ .. \ l_i]} \chi(d_{i,j}, t)$ we obtain

$$\begin{aligned} av(m) &= \frac{1}{t_1 - t_0} \int_{t_0}^{t_1} \chi(m,t) dt & (5.10) \\ &= \frac{1}{t_1 - t_0} \prod_{i \in [1 \ .. \ k]} \chi(D_i, t) dt & (5.11) \end{aligned}$$

Let $\tau(a) = \{t \in [t_0, t_1] \mid \chi(a,t) = 1\}$ be the subset of the interval $[t_0, t_1]$ in which a functions correctly, since $\chi : N \times [t_0, t_1] \times \{0, 1\}$ and χ is measurable (for instance, in our context we can assume that, once we fix a node a, the graph of $\chi(a,t)$ switches from 0 to 1 a finite number of times) we have that

$$\int_{t_0}^{t_1} \chi(a,t) \cdot \chi(b,t) dt = \int_{\tau(a)} \chi(b,t) dt = \int_{\tau(a) \ \cap \ \tau(b)} 1 dt$$

Based on this, (5.10) becomes

$$av(m) = \frac{1}{t_1 - t_0} \int_{\tau(D_1) \ \cap \ \ldots \ \cap \ \tau(D_k)} 1 dt \tag{5.12}$$

By set theory, if τ is a set and $A, B \subseteq \tau$, we have that $A \cap B = \overline{\overline{A}^{\tau} \cup \overline{B}^{\tau}}^{\tau}$, where $\overline{A}^{\tau}$ is the complement of A w.r.t. τ. Then, let $\tau = [t_0, t_1]$, we have that

$$av(m) = \frac{1}{t_1 - t_0} \int_{\overline{\overline{\tau(D_1)}^{\tau} \ \cup \ \ldots \ \cup \ \overline{\tau(D_k)}^{\tau}}^{\tau}} 1 dt \tag{5.13}$$

Recall that, if $X \subseteq \tau$ and both are measurable,

$$\int_{\overline{X}^{\tau}} f dt = \int_{\tau} f dt - \int_X f dt$$

thus:

$$av(m) = \frac{1}{t_1 - t_0} \left[\int_{\tau} 1 dt - \int_{\overline{\tau(D_1)}^{\tau} \ \cup \ \ldots \ \cup \ \overline{\tau(D_k)}^{\tau}} 1 dt \right] \tag{5.14}$$

Recall that if A and B are measurable and f with positive values, we have that $\int_{A\cup B} f dt \le \int_A f dt + \int_B f dt$, where the equality holds if A and B have empty intersection. Therefore we have that

$$av(m) \ge 1 - \frac{1}{t_1 - t_0}\left[\int_{\overline{\tau(D_1)}^{\tau}} 1dt + \cdots + \int_{\overline{\tau(D_k)}^{\tau}} 1dt\right] \tag{5.15}$$

Where the inequality becomes an equality if the D_is are pairwise disjoint.

Intuitively,

$$\int_{\overline{\tau(D_i)}^{\tau}} 1dt$$

is the unavailability caused by the nodes in D_i in the time interval $\tau = [t_0, t_1]$. In fact, let $D_i = \{d_{i,1}, \ldots, d_{i,l_i}\}$, we have that

$$\int_{\overline{\tau(D_i)}^{\tau}} 1dt = \int_{\overline{\tau(d_{i,1}) \cup \ldots \cup \tau(d_{i,l_i})}^{\tau}} 1dt \tag{5.16}$$

By set theory, if $A, B \subseteq \tau$ we have that $\overline{A \cup B}^{\tau} = \overline{A}^{\tau} \cap \overline{B}^{\tau}$; then

$$\int_{\overline{\tau(D_i)}^{\tau}} 1dt = \int_{\overline{\tau(d_{i,1})}^{\tau} \cap \ldots \cap \overline{\tau(d_{i,l_i})}^{\tau}} 1dt \tag{5.17}$$

Recall that, if A and B are measurable,

$$\int_{A\,\cap\,B} 1dt \le min\left(\int_A 1dt, \int_B 1dt\right)$$

thus:

$$\int_{\overline{\tau(D_i)}^{\tau}} 1dt \le min_{j\in[1\,..\,l_i]}\left((t_1 - t_0) - \int_{\tau(d_{i,j})} 1dt\right) \tag{5.18}$$

We can substitute in (5.18) the inequality with an equality as we are interested in the upper bound of the unavailability caused by the elements in D_i. Substituting (5.18) into (5.15) we obtain:

$$\begin{array}{rcl} av(m) & \ge & 1 \\ & & -\ min_{i\in[1\,..\,l_1]}\left(1 - \frac{1}{t_1-t_0}\int_{\tau(d_{1,i})} 1dt\right) \\ & & -\ \ldots \\ & & -\ min_{i\in[1\,..\,l_k]}\left(1 - \frac{1}{t_1-t_0}\int_{\tau(d_{k,i})} 1dt\right) \end{array} \tag{5.19}$$

Let us now distinguish two cases: (a) D_is are pairwise disjoint, i.e. $\forall i, j\ D_i \cap D_j = \varnothing$, and (b) they are not pairwise disjoint. In (a) the inequality sign in (5.15) becomes an equality sign. Therefore (5.15) becomes

$$av(m) = 1 - \frac{1}{t_1 - t_0}\left[\int_{\overline{\tau(D_1)}^{\tau}} 1dt + \cdots + \int_{\overline{\tau(D_k)}^{\tau}} 1dt\right] \tag{5.20}$$

and we can use (5.19) to determine $av(m)$. Since all D_is are pairwise disjoint, if $d_{i,j} = n'$, then $\frac{1}{t_1-t_0}\int_{\tau(d_{i,j})} dt = \frac{1}{t_1-t_0}\int_{t_0}^{t_1} \chi(n', t)dt$. Therefore, if we set $\forall n' \in N'$ $av(n') = \alpha_{n'}$, we determine α_m from (5.19) by substituting $\frac{1}{t_1-t_0}\int_{\tau(d_{i,j})} dt$ with $\alpha_{n'}$ when appropriate.

In fact, in this case the various D_is are independent from each other and to calculate the minimal availability (given the constraints on the availability of the internal nodes) we can restrict the search to those schedulings in which the various D_is are unavailable in non-overlapping time frames and all the elements of any D_i are unavailable at exactly the same time. For example, consider the case of Figure 5.2-b. *DEPS*$_m = \{\{m'\}, \{n'_1, n'_2\}\}$ and, according to (5.19), $\alpha_m = 1 - (1 - \alpha_m) - \mathit{min}((1 - \alpha_{n_1}), (1 - \alpha_{n_2}))$. The availability of m reaches α_m when (1) m' is unavailable for $1 - \alpha_{m'}$ but not at the same time of n_1 and n_2, and (2) n_1 and n_2 are unavailable at the same time for $\mathit{min}((1 - \alpha_{n_1}), (1 - \alpha_{n_2}))$. This also shows that it exists a state function for which, when $avn' == \alpha_{n'}$, then $av(m) = \alpha_m$.

In the general case (b), the elements D_i of *DEPS*$_m$ are not pairwise disjoint, i.e. $\exists i, j \mid D_i \cap D_j \neq \varnothing$. By using (5.19) in this case we would obtain a value of α_m which is not minimal. To determine the availability lower bound α_m we then set-up a linear programming problem. For the sake of presentation, we call $a_{i,j}$ the (unknown) quantity

$$\frac{1}{t_1 - t_0}\int_{\tau_{d_{i,j}}} 1dt$$

and u_i the (unknown) quantity

$$1 - \frac{1}{t_1 - t_0}\int_{\overline{\tau(d_{i,1})}^{\tau} \cap \ldots \cap \overline{\tau(d_{i,l_i})}^{\tau}} 1dt$$

By substituting $a_{i,j}$ and u_i in (5.15) where possible, we obtain the objective function we need to minimise to find the lower bound we aim at. The first k constraints are derived from (5.17) and ensure that the nodes belonging to a certain D_i can be unavailable all at the same time for u_i.
Given the definition of dependency set, if we call $D(n') = \{d_{i,j} \mid d_{i,j} = n'\}$), then we know that

$$\begin{array}{rcl} av(n') & = & \frac{1}{t_1-t_0}\int_{t_0}^{t_1} \chi(n', t)dt \\ & = & 1 - \sum_{d_{i,j} \in D(n')} 1 - \frac{1}{t_1-t_0}\int_{\tau(d_{i,j})} 1dt \end{array} \tag{5.21}$$

From (5.21) we derive a set of constraints which ensure that the lower bound of the availability caused by each internal node n' in all the elements of *DEPS*$_m$ is the known value $\alpha_{n'}$.

As a result we get the following linear programming problem:

$$\alpha_m = \begin{cases} \text{minimize } 1 - u_1 - \cdots - u_k \\ \text{subject to} \\ u_1 = (1 - a_{1,1}) = \cdots = (1 - a_{1,l_1}) \\ \vdots \\ u_k = (1 - a_{k,1}) = \cdots = (1 - a_{k,l_k}) \\ \forall n \in N, \sum_{d_{i,j} \in D_n} 1 - a_{i,j} \geq 1 - \alpha_{n'} \\ a_{1,1}, \ldots a_{1,l_1}, \ldots, a_{k,1}, \ldots, a_{k,l_k} \geq 0 \end{cases} \tag{5.22}$$

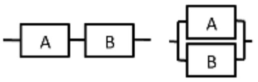
A solution to this problem can be found by using the simplex algorithm. Such a solution indicates both the lower bound for the availability of m, i.e. the minimal value of (5.15) With this we then proved our theorem. □

5.9 Representation capabilities

In order to apply our approach to the real-world one could wonder if the technique we adopt to represent the complex system (*AND /OR* dependency graphs) is expressive enough. A very popular and widely used approach to represent complex systems for reliability and availability analysis is using RBDs. In this section we show that our representation is at least as expressive as an RBD.

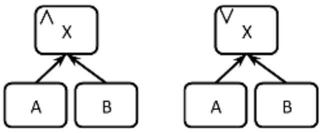
An RBD is a graphic representation of the complex system where every component is represented by a block (rectangle) and it is connected to other components, in series or parallel form. A serial connection between two blocks (see Figure 5.6a) means that the system (composed by the two blocks) is operational when both blocks are operational. A parallel connection (see Figure 5.6b) between two blocks means that the system is operational when at least one of the two blocks is operational. The whole system is then modelled as a combination of series and parallel blocks. A group of interconnected components can be represented as a single macro-component. In turn, macro-components can be connected to other components (e.g. see Figure 5.8) and grouped again. Hence, to prove that our representation is as expressive as an RBD, we need to show how each one of the three main operations on RBDs can be equally expressed as an *AND /OR* dependency graph.

If we consider the serial system of Figure 5.6a, made of only two components, the corresponding *AND /OR* dependency graph is given in Figure 5.7a. To represent the system we use an *AND* node X, which depends on nodes A and B.



(a) Series (b) Parallel

Figure 5.6: RBD



(a) Series (b) Parallel

Figure 5.7: *AND* /*OR* dependency graph

Similarly, the parallel system of Figure 5.6b is equivalent to the *AND* /*OR* dependency graph in Figure 5.7b there the *OR* node X depends on nodes A and B.

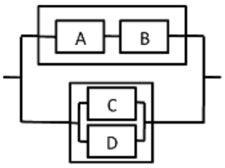


Figure 5.8: RBD parallel composition

Regarding the composition operation, in Figure 5.8 we show the parallel composition of two components, where each of them is a serial composition two sub-components, in parallel with two other sub-components. Figure 5.9 shows the same system represented as an *AND* /*OR* dependency graph. We model as an *AND* node X_1 the serial composition of sub-components A and B, and as an *OR* node X_2 the parallel composition of sub-components C and D. Finally, we add an *OR* node X which represents the parallel composition of the two above mentioned components. It is easy to see that we can model any combination of grouping and series/parallel compositions in the RBD, with a combination of *AND* and *OR* nodes.

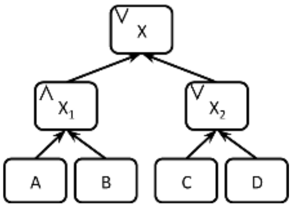


Figure 5.9: *AND* /*OR* dependency graph parallel composition

Chapter 6

Concluding Remarks

We now summarise the contribution of this thesis, in relation to the research aim and the related research questions discussed in Chapter 1. We also highlight future research directions in the area of availability planning.

6.1 Summary and conclusions

In the introductory chapter we formulate the following research aim:

"Design and validate techniques that improve the accuracy and effectiveness of availability planning, while guaranteeing feasibility within budget."

We assume in Chapter 1 (confirmed by our findings in Sections 3.4, 4.3 and 5.6) that organisations are willing to improve their control over the availability of their IT infrastructure to support business needs and reduce costs. To optimise the IT availability level with regard to costs (large size) organisations adopt a combination of risk, business continuity and service level management. We call availability planning the set of analysis activities in risk, business continuity and service level management by which organisations set the requirements and take decisions regarding the availability of the IT systems supporting their business. There are several standard methods that draw the guidelines for carrying out risk, business continuity and service level management. To be as generally applicable as possible, these standards do not include implementation details. Therefore, each organisation needs to find the specific set of techniques that put into practice standard guidelines and suit their needs. However, techniques supporting availability planning are often too resource-consuming or they are error prone, relying on people's ability to cope with the increasing complexity of IT systems. Based on these observations we then formulate in Chapter 1 three research questions which address the research aim from the angle of risk management, business continuity and service level management.

Below we list the research questions and the results of our explorations. The first research question is:

"How can we improve the accuracy of current techniques for assessment and mitigation of availability-related IT risks, while guaranteeing feasibility within budget?"

In Chapter 2 we focus on the selection of a cost optimal mitigation strategy for availability-related IT risks. We start with the observation that risk management methodologies do not take into consideration the propagation of availability incidents in the IT infrastructure when evaluating the strategies to cope with them. This can result in the selection of unbalanced mitigation strategies that overprotect some IT components considered important and leave under-protected other (only apparently more marginal) components whose unavailability may still cause by propagation a major damage to the organisation. To cope with this problem we introduce the TD model and framework to carry out an accurate risk mitigation by using the data collected during a quantitative risk assessment. We use this framework in a case study making use of data of a primary insurance company provided by KPMG. We run an optimisation study of the cost/benefit ratio of countermeasures, taking into account incident propagation and thus increasing the accuracy of the selection process. Our case study provides evidence that the framework based on the TD model can support decision makers in a more accurate selection of countermeasures with respect to current practice, without requiring them knowledge about the underlying timed model checker involved. However, the case study with KPMG also brings to light some limitations: (1) the model needs reliable quantitative information: it can only be used in organisations that carry out quantitative risk assessments, (2) how to obtain such information is not covered in depth, although in Section 2.5 we give some first indications, (3) we are able to support only some of the possible risk mitigation strategies (i.e. the risk reduction ones): risk avoidance, sharing and retention practices are not covered, (4) our approach does not automatically deal with risk mitigation strategies that are required by regulations: for them a manual pre-processing is still needed.

In Chapter 3 we introduce the QualTD model and framework which is meant to be used to systematically assess availability-related IT risks in combination with existing risk assessment methods. We use the framework based on the QualTD model in a case study to test both its accuracy and feasibility. The results of the case study and the review of existing risk management methodologies provide evidence that the QualTD model constitutes an improvement towards the accuracy of risk assessment techniques adopted at the company we ran the case study at. We registered more accurate results (~20%) and a reduction by one fourth of the number of choices left to the risk assessor (see Section 3.4.3). Our observations indicate that similar advantages can be obtained for organisations adopting a risk assessment approach similar to the one used by the company in the case study. In more detail, based on the literature study on existing risk assessment methods, we see that our framework adapts naturally to the methods that address risks as a combination of threat, vulnerability and impact (see Section 3.5.1). On the other hand, we also observe that our approach is not compatible with methods based on security goals and requirements (e.g. EBIOS). Regarding feasibility, the case study showed a small time overhead connected to the creation of the QualTD model with respect to the company approach (see Section 3.4.3). However, we expect that an organisation may be willing to accept that overhead in case the increased accuracy is its desired goal. Finally, we note that our approach does not address the following aspects of risk assessments. (1) Risk ownership (i.e. the process of determining who is responsible for managing the identified risks) is not addressed in depth, although it is a major success factor for the effectiveness of risk assessments. (2) We do not discuss specific techniques to identify threats and vulnerabilities. However, this topic is addressed in detail by other techniques (e.g. HazOP [22] and its extension for security, Security HazOP [79]).

The second research question is:

"How can we improve the accuracy of current techniques for creating and maintaining business continuity plans, while guaranteeing feasibility within budget?"

In Chapter 4 we focus on the analysis phase that precedes and follows the creation of the IT-related part of a BCP. We observe that RTOs are an important tool to ensure that a BCP meets the business continuity requirements of the organisation. We also observe that RTOs are manually set on the components of the IT infrastructure. This is error prone and time consuming, to the point that some IT components can receive RTOs that do not satisfy business requirements, or not receive an RTO at all. A BCP which is based on such wrong premises can be ineffective. To cope with this problem we introduce the TDR model and tool and we test it in a case study run with the data of an insurance company provided by KPMG. We use the TDR model to determine the maximum recovery time for each component of the IT infrastructure and to assess that RTOs satisfy business requirements and are consistent with each other. The results of the case study provide evidence that the TDR model is able to identify critical points in which manually set RTOs were not compliant with business requirements or within each other, thus allowing a more accurate analysis of the business requirements. The model can also be used to set RTOs in first instance, by using the calculated maximum recovery time as a starting value. However, due to the limited scope of the case study we could not determine to what extent the resulting BCP was influenced by these critical points. We believe this could be important to set with more precision to what extent the TD model is useful for supporting business continuity planning. We note that using the TDR model was feasible in our case study, and that the information needed to build it can be assumed to be available for organisations that follow the BS25999 business continuity standard and use supporting tools similar to the one used by KPMG. However, the analysis phase of business continuity management is not limited to RTOs. The main activities also involve Recovery Point Objectives (RPOs), threat analysis and impact scenarios. RPOs are used to set the acceptable latency of data that will be recovered for the information assets needed to run critical business functions. Differently from RTOs that are applied to IT components, RPOs are applied to information assets. The TDR model does not support the assessment of RPOs, which requires additional information about the lifecycle of data and on the business processes that use it. However, it could be useful in the analysis of electronic backup strategies, for example to check the consequences of IT faults on the availability of the backups. The threat analysis has the purpose of identifying the threats that may compromise business continuity. As for the threat identification in risk assessments, the TDR model does not cover this topic. The model could in principle be used to support the analysis of impact scenarios, to determine which IT systems are affected (also by propagation) when a given threat materialises, similarly to what we do with the TD and QualTD model. However, this feature was not tested and to support this claim we would need at least one new specific case study.

The third research question is:

"How can we improve the accuracy of current techniques for managing availability-related SLAs, while guaranteeing feasibility within budget?"

In Chapter 5 we focus on the analysis and optimisation of IT service availability. We note that more and more frequently large organisations tend to delegate (part of) the management of their IT service components to external outsourcing companies. IT managers need to choose among different outsourcing companies and different offers the one that suits their needs and their budget, and sign SLAs with the outsourcing company to regulate (among others) the availability of the outsourced service components. At the same time, IT managers need to sign similar SLAs with the other business units of the organisation, to guarantee the quality of the IT services provided by the organisation's IT department. To achieve the best trade-off between outsourcing costs and the desired final service availability, IT manager then need to optimise the choice of outsourcing companies and offers. Traditional reliability techniques cannot be used for this purpose as they require information that is usually not reported in SLAs (i.e. MTBF and MTTR). Therefore, IT managers are forced to take decisions based on their own experience and on concepts like trusted partners and business relationships: this can lead to a non-optimal allocation of resources and could cause frequent violations of the internal SLAs. To cope with this problem we introduce A^{2}THOS, a framework for the analysis and optimisation of the availability of mixed-sourced IT services. We provide a mathematical foundation of the algorithms A^{2}THOS is based on. We then test A^{2}THOS in a case study carried out on two IT mixed-sourced IT services of a large multinational company. A first result of the case study suggests that A^{2}THOS provides a more accurate evaluation of the minimal service availability compared with the manual estimate made by IT managers, particularly when the availability estimated by IT managers was lower than what guaranteed by A^{2}THOS. This seems to be confirmed, in the context of the case study, by the empirical data on the past availability of some of the Oxygen services, which is much better than the one guaranteed in the SLAs. A second indication of the case study is that A^{2}THOS was used to find the optimal combination of outsourcing offers based on the resulting service availability. In this case we were able to achieve the same service availability at a ~30% lower price. However, the selection of outsourcing offers is often based on different other parameters: availability and price are only two of them. Other parameters could be measurable, such as incident response time, average service response latency or more difficult to quantify, such as the reputation of the outsourcing company or the location where services are being held. Therefore, we realise that the selection process cannot be delegated solely to the result of an optimisation problem. The coverage of all possible selection parameters is out of the scope of this chapter. For this reason, we see a potential for the applicability of A^{2}THOS only in the context of a wider framework for supporting the selection of outsourcing offers, in which A^{2}THOS is used as a specific component for availability of mixed-sourced services.

The main conclusions we can draw from our findings are:

1. many organisations that use IT to support their business can optimise the availability of their IT infrastructure with positive effects on their business capabilities (see Sections 3.4, 4.3 and 5.6);

2. it is possible to develop techniques that support IT availability planning while guaranteeing feasibility within budget from the angle of risk, business continuity and service level management (see Sections 2.5, 3.4, 4.5, 5.5 and 5.6);

3. the structure of an IT architecture can be used for practical availability planning (see Sections 2.3, 3.3 and 5.2).

Regarding the *feasibility within budget* mentioned in the second point, based on the results of our case studies we cannot guarantee that the techniques we propose in this thesis are within the budget (i.e. feasible in terms of time and resources) of any organisation. To guarantee this, we would need to estimate the “budget” of all organisations. However, the case studies we conducted give evidence that the cost of adopting our techniques is in the same order of magnitude as the cost of the ones currently adopted by medium-large organisations. The same thing could not be said for other solutions, e.g. very detailed model checking.

The third point is derived by observing that the techniques that we present in this work are all based on a common representation of the architecture of (part of) the IT infrastructure. This representation is captured by the timed dependency graphs, and their extended versions timed *AND* /*OR* dependency graphs and *AND* /*OR* dependency graphs, that we use to (a) list the relevant components of the IT infrastructure and the business processes it supports and (b) capture the functional dependencies among components to model the propagation of incidents caused by component’s failure. We believe that using the same underlying view of the IT infrastructure for risk, business continuity and service level management contributes to create a common view about the availability issues of the IT infrastructure among the different IT managers involved. In turn, this can lead to a better alignment of the efforts towards availability planning.

An interesting point that we encountered during our research regards the different views on the two main paradigms for risk management: the qualitative and quantitative one. Based on our experience we can conclude that the most suitable approach to adopt depends on the context. A fine-grained quantitative risk assessment can provide loss estimates that can be used to carry out optimisation studies. However, it will never be more accurate than the input numbers it is based on: when collecting accurate numbers is hard this may not be the right approach to take. On the other hand, a well-motivated qualitative risk assessment can provide useful information without requiring numbers, but its results are not readily spendable in optimisation processes. Based on our experience we observe that organisations in which other risks (e.g. financial risks) are successfully assessed following the quantitative paradigm choose to address IT risks following the same paradigm. This is often the case for banks and insurance companies. The reason for this is twofold: first, the managers of these organisations are used to address risks in that way, and therefore they tend to push the IT department to stick to the standards they like. Secondly, in these kinds of organisations there are in places processes and metrics specifically tailored for quantitative risk assessments (think for example of an insurance company, for which quantifying risks is part of the primary business), which can be adapted or partly re-used for IT. The same cannot be said for other kinds of organisations, in which reliable numerical values are difficult to obtain as there are not enough clear data or metrics for this [39]. In this case the qualitative paradigm is the preferred one.

Finally, we want to stress once again that the techniques we introduced in this work are not an alternative to standard methodologies. Although they do not provide implementation details, standard methodologies do provide the guidelines to successfully build into the organisation the processes of risk, business continuity and service level management. Failure in properly setting up the processes (including human factors and information communication) can compromise their effectiveness as much as the lack of implementation details. Instead, we believe our techniques can be successfully integrated with risk, business continuity and service level management processes based on standard methodologies.

6.2 Future work

The results of this thesis open several possible future research directions.

Service degradation The TD and QualTD models can be extended to include service degradation as part of the impact estimation process. For example, a DoS attack could not succeed in shutting down completely an IT system, but it could slow down its response time. Currently, this kind of impact can only be addressed by considering the system operational if it can respond within a time and not operational otherwise. Including different operational levels can increase the accuracy of risk evaluation when such an accuracy is required.

Dependency types The dependency graphs used by our techniques can be extended to support a wider range of dependency types. Currently, the TD and TDR models only support AND dependencies, while the QualTD and A^{2}THOS both support AND/OR dependencies. There are however other dependency types; for example the dependency "n out of m components are available" can be used to model the case where if a node x depends on m nodes, it is considered available if at least n of the m nodes are available. In this thesis we followed the engineering principle of starting with the simpler and most common cases. Therefore, we did not include these dependency types as in our case studies we never found a configuration requiring them. However, there could be cases in which the supporting more dependency types could be a success factor for the adoption of our techniques.

Inclusion of more sophisticated optimisations The optimisation problems and algorithms of Chapter 2 and Chapter 5 can be extended to support more sophisticated micro-economic models. Burgess gives in [23] an overview of the economic game theory techniques (e.g. promise theory and principal agent theory) that can be used to support decisions about the economic aspects of IT management. We believe that integrating our optimisation algorithms with these techniques is an interesting research problem which could increase the value of our techniques by giving IT managers extra support in taking decisions.

Integration with configuration management tools Configuration management is a field of management that focuses on maintaining a system's requirements, design and operational information consistent with its performance and functional attributes throughout its life. IT managers maintain such information by using configuration management tools (e.g. IBM Tivoli [100]). Configuration management tools could be used to feed our models with up-to-date information about the architecture of IT systems, their functional design details and their past availability performances. In this scenario, IT managers could carry out availability planning semi-automatically by using our techniques integrated with configuration management tools.

Appendix A

Dependency Graphs Analysis, FTA and FMEA

Dependency graphs are used by all models presented in this thesis to represent the (portion of) IT infrastructure to be analysed and its behaviour in case of unavailability of one of the components. Dependency modelling shares some similarities with other popular techniques such as Fault Tree Analysis (FTA) and Failure Mode and Effects Analysis (FMEA) both in the way the model is built and in the kind of analysis one can carry out with them. Therefore, we find useful to give a top level description of what dependency modelling is compared to these other two techniques. Table A.1 summarises the main features of each technique.

In FTA [76], a fault tree is a Rooted Directed Acyclic Graph (RDAG). The graph is built starting from the root, which represents a so-called top-event i.e. an undesired event involving the system (system or sub-system failure). Each node in the graph is a mixture of logical gates (AND/OR) and events. The graph is built top-down using information about events causing system disruption by trying to enumerate all the combinations of other events that could cause the top level event.

The FMEA [59] technique is a list based on the enumeration of all the possible system (and system component's) failure modes and on the analysis of the effects of failures. Failure modes are ranked according to a qualitative estimate of their effect severity. According to Allen Long [103], FMEA is a bottom-up technique and it is particularly suitable for the analysis and ranking of single failures of components (as opposed to fault trees and dependency graphs which can also be used to study the effects of combined simultaneous failures). A key preparatory step for carrying out FMEA is to draw an architecture diagram of the system under analysis which describes how the system is made and how it works. The schema is meant to be used both for the enumeration of failure modes and for the evaluation

Table A.1: Global picture of the differences between FTA, FMEA and Dependency modelling.

	Input	*Output*	*Structure*	*Nodes*	*Edges*
FTA	Information about the disruption events in the system.	All the combinations of events that can lead to the top event.	Rooted Directed Acyclic Graph	Events and logical gates.	Cause and effect relationship between events
FMEA	System architecture diagram (including components and connections between components).	Component failures plus the effect of component failures on the system.	List	System components failing in a certain mode (entries in the list).	N/A
Depdendncy modelling	Information about the architecture of the system.	A model of the system that can be used to simulate the effect of a component failure on the system.	Directed Acyclic Graph.	System components.	Cause and effect relationship about component's failure.

of the failures effects. Indeed, most of the information needed to prepare such a schema can also be used to build a dependency graph.

A dependency graph is a Directed Acyclic Graph (DAG) based on information about the system architecture. Differently from a fault tree, a dependency graph has nodes representing system components (as opposed to events) and edges expressing a cause and effect relationship among components meaning that the failure of one component causes the failure of the other. As we mentioned in Chapter 5, a single dependency graph can model a “forest” of fault trees, each having the failure of one of the functionalities of the system under analysis as top event. Dependency graphs refer to the architecture of IT systems to enumerate the combinations of components failures that may cause the unavailability of one of the system's functionalities.

Appendix B

Building Dependency Graphs

In Chapter 3 we describe how we build a dependency graph for the case-study in which we tested the QualTD model. In this appendix we generalise the description by providing general guidelines that can be used for the models and tools described in this thesis.

The guidelines are derived from the experience we earned over time during the case-studies we did. We organise these guidelines in the form of a tutorial consisting of five steps. The steps describe how to gather information and how to use such information to model the graph. The tutorial is meant to answer the following questions:

- Who are the stakeholders of this model and what do they want to know? In other words, what information does my model needs to include?
- What is the system that I need to model, what does it do and what are its boundaries?
- How do I get information about the system?
- How do I model the various system components?
- How can I discover and model dependencies among system components?

We will also provide a small example of the outcome of each step in an "Example" paragraph, referring to the running example of Chapter 3.

Although the exposition may suggest the process of building dependency graphs is linear, in practice one could easily reach a certain step only to discover that the information gathered in a previous step is incomplete or even incorrect. Should this happen, it is necessary to rewind the process to one of the previous steps, add the missing or incomplete information, and carry on.

Step 0: Terms and concepts

To avoid confusion using this tutorial one needs to understand the following terms and concepts and name them explicitly for the case-study at hand (i.e. name the analyst, the system or the stakeholders).

- **Analysis** The activity for which the dependency graph is required. This can either be in the context of risk management, business continuity or service level management.
- **Analyst** The person or persons in charge of carrying out the analysis. Should be independent from the stakeholders.
- **Analysis result** The information gained after carrying out the analysis, i.e. the output in the form of the TD model, the QualTD model, the TDR model or A^2THOS.
- **Stakeholder** The person or persons who will use the analysis results as part of an availability planning activity.
- **System** The target of the analysis, a composition of hardware, software and procedures that make a unit in an IT infrastructure.
- **Sub-system** In a logical decomposition of a system, a portion of the system: it can be a mixture of hardware and software that serve for one specific purpose on the system.
- **System component** A portion of the system or system block that is modelled by the analyst as an atom (i.e. it is assumed not to be made of other components).
- **Dependency** A functional relation between two components such that the availability of one component influences the availability of the other.
- **Business Process** A (business-related) procedure supported by IT.
- **Service** A functionality provided by the system to its intended users.
- **IT service** A functionality provided by a system or a (set of) components to other components.
- **Application** Software running on a server as an OS process.
- **Server** The combination of hardware and operating system making a computer.

- **Location** A physical place where servers and network components are stored (e.g. a data centre).
- **Network component** The logical segment of a computer network or a device in the computer network devised to control the network traffic (e.g. switches, routers firewalls, etc.).

Step 1: Stakeholders and goals

Before the analyst starts building the dependency graph, the first step consists of determining (1) what kind of analysis the dependency graph is needed for and (2) what kind of results are expected by the stakeholders.

This is important to keep the analyst focused on the important modelling decisions, i.e. on the information that (directly or indirectly) involve the analysis results.

For example, suppose the system under analysis is managed by more than one unit/department: only one of them is a stakeholder (i.e. only one unit department is interested in the analysis results) and only the portion of the system managed by the stakeholder is of interest in the analysis. In this case, the different sub-systems can be modelled with a different granularity: the sub-systems managed by the stakeholder in more detail, the others in less detail, thus focusing the analysis efforts on the aspects relevant for the stakeholders.

In many cases there is more than one stakeholder: the analyst has to determine which stakeholder is responsible for the different sub-systems to be able to both acquire information from the right source and to report the relevant analysis results to each stakeholder.

Example (part 1) At the end of Step 1 the analyst may have acquired the following information:

1. the dependency graph is needed for a risk assessment on the availability of two IT systems;
2. the systems are developed, managed and maintained by one department of the organisation which is also the requester of the risk assessment;
3. the systems are used to manage holiday reservations for employees and customer relationship management;
4. the department only manages servers and applications running on servers related to the two systems: other IT services such as network connectivity,

server name resolution (DNS) and even standalone applications are managed by another department and are not of interest for the current risk assessment.

Step 2: Global picture

The second step consists in creating a global picture of how the system works. This is very important to set up a "skeleton" of the dependency graph and to determine what other information the analyst needs to acquire.

The global system picture consists of a map of the main sub-systems, together with their function and the interactions among them. To this end, when considering one sub-system, the analyst should determine if the sub-system needs other sub-systems or system components to function properly, and make sure they are represented in the map.

The global picture can be obtained in two main ways: either by reading existing documentation (e.g. requirements and specification documents) or by means of interviews with people who know how the system is (will be) made and how it works. This can refer to more than one person. For example, one could learn how the system is made by asking the developer who made it, or from a system administrator who performs the maintenance, while the information about what the system does can be better obtained by the people who designed the system or by an experienced system user.

When time is not a problem, the best thing to do is combining the two approaches. By first reading the documentation the analyst will make a first concept of the system, which can then be refined during the subsequent interview sessions.

In both cases (documentation or interview) it is helpful to draw a first, rough, picture of the system architecture. For example, when extracting information from the documentation, a picture will help explaining the function and the connection among the different sub-systems. In case of interviews, the picture will help communicating information and make sure information is communicated with the least ambiguity as possible.

Example (part 2) After Step 2 the analyst will have determined the main sub-systems subject to analysis and drawn a rough picture of them (see Figure B.1). The information acquired explains that the holiday reservation system is composed of two sub-systems: a web application handling application logic and user interaction and an Oracle database to store data. In turn, the CRM system consists of two main sub-systems: a standalone GUI running on the employees workstations and an Oracle database with stored procedures to store and handle CRM data.

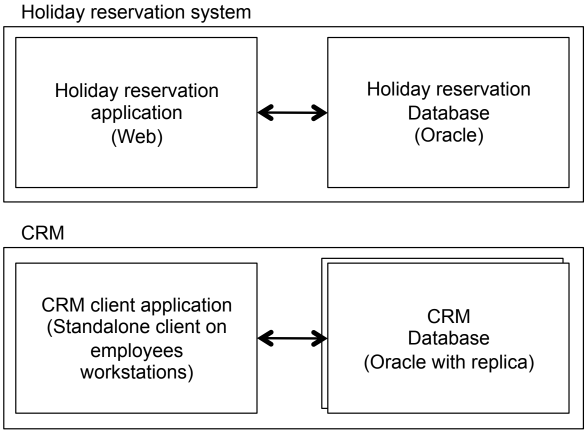

Holiday reservation system
Holiday reservation application (Web)
Holiday reservation Database (Oracle)
CRM
CRM client application (Standalone client on employees workstations)
CRM Database (Oracle with replica)

Figure B.1: Global picture of the system of the running example of Chapter 3.

Step 3: System boundaries

Setting system boundaries is important to limit the scope of the analysis to a size that is both feasible for the analyst and meaningful for the stakeholders. This activity mainly consists of deciding:

1. which functionalities provided by the system will be analysed;
2. which sub-systems will be included in the dependency graph;
3. the level of detail adopted in modelling each sub-system.

To do so, one has first to decide which system functionalities are of interest in the current analysis. The analyst can derive this information from the results of Step 1. For example, if a certain system functionality is not used by the any of the stakeholders, it can be left out from the dependency graph both to reduce the model complexity and to improve the readability of the graph for the stakeholders.

After enumerating the functionalities, the analyst has to determine which subsystems should be included in the model. This list will be used in next steps to start building the dependency graph. The information needed to compile the list of components can be extracted from the global system picture of Step 2. When compiling the list, a useful trick is to mark each sub-system in the global picture as soon as it is added to the list. In this way, after the list is complete, there should be no unchecked components, except for those that contribute to implement a functionality that has been deliberately left-out from the analysis.

Finally, for each sub-system on the list, the analyst has to determine wether it is in scope of the analysis (according to the results of Step 1). Some sub-systems may be general IT services required for the system to function properly (e.g. Internet connectivity, DNS servers, etc.). Sub-systems of this kind can be modelled less precisely than the others, for example by representing them as a single IT service node, regardless on how the service is implemented.

Example (part 3) After Step 3 is complete, the analyst can enumerate the following two services as in scope with the purposes of the analysis and the responsibilities of the stakeholder:

1. Holiday reservation management service: this functionality includes the complete management of holiday reservations.
2. CRM repository service: this functionality includes the storage and batch processing of customer information, purchase orders and invoices.

According to the global picture derived from Step 2 and the analysis of stakeholders and goals of Step 1, the following sub-systems are identified:

1. Holiday reservation application
2. Holiday reservation database
3. CRM database

Notice that the CRM client application has been left out from the list, as it is not managed by the department requesting the analysis and it is therefore not relevant for this risk assessment. The network connectivity sub-systems enabling the systems to be accessed through the organisation's LAN have been left out from the picture for the same reason.

Step 4: Detailed system information

In this step, each sub-system is split into its basic components (e.g. applications, servers, network components, premises, IT services etc.).

After creating a list of the sub-systems the analyst has to acquire detailed information about every component of the sub-system. To do so, the analyst has to find information describing how each sub-system is implemented and how it is used.

In more detail, the analyst should determine the following information:

1. what applications are used to implement the sub-system, normally one sub-system is made of a single application, but there might be cases in which two or more interacting applications are used to build a single high-level component;

2. by which server(s) run the identified applications, in case the component is implemented by more than one server what is the purpose of the split (load balancing, warm/cold/hot standby, etc.);

3. for each identified server, to which network is the server connected and where is it physically located;

4. whether there is any other application running on the same server(s) for different purposes;

5. how each component interacts with the others (e.g. which network protocol is used, which server/service pairs communicate, etc.): this is used to determine the dependencies among components.

Sources of information for this can be:

- *System documentation*. Documentation includes functional specifications, system design documents and network diagrams.

- *Interviews*. In case system documentation is not available, the same information can be obtained by interviewing the personnel in charge of the system design, development and maintenance.

- *Architecture design tools*. In case the system architecture is documented using an architecture design tool, it is possible to extract from it most of the information needed to build the dependency graph. Architecture design tools are meant to support the design, analysis, visualisation and maintenance of enterprise IT architectures. There exist a number of frameworks describing formalisms for architecture design (e.g. TOGAF [113], Zachman [114] or Archimate [84]). Each framework is supported by one or more tools that implement the framework and help users in drawing IT architectures and doing automatic analysis over them. In one of our case-studies we successfully built a dependency graph from an IT architecture described using the ArchiMate framework. The architecture was designed with the Architect tool from BiZZdesign [87]. Architect supports a simple scripting language to interact with an existing architecture project. We used the scripting language to automatically export a dependency graph which could then be used to run our models and tools.

- *Infrastructure monitoring tools*. In case the system is already deployed in a production or even in a test environment, the analyst can use IT infrastructure monitoring tools such as Nagios [107] or Hyperic [99] (when available). These tools are typically made of several agents deployed on the hosts of the IT infrastructure and a centralised repository collecting information from agents. The analyst can consult the repository to learn (among others) about the hosts deployed on the network, the services that are running on them, their availability and the resource consumption (e.g. CPU, memory, network bandwidth, etc.). This information can also help to ensure the completeness of information about the components that make up the system (as it is probable that some components escape the attention during interview sessions or not be mentioned in the documentation).

- *Network traffic*. A way of collecting information about the service dependencies is to observe the network traffic exchanged by the identified hosts for a sufficiently long time (e.g. one week). This helps determining with more precision which services communicate with each other and how they do that. As a consequence, dependencies among applications are more clearly defined. Collecting network data and deriving dependency information could also be automated, to speed up the process in case of large systems, for example with *tcpdump2csv*, a tool that parses network traffic captured through *tcpdump* and creates a CSV file which can then be used to derive dependencies between servers/applications. A list of other tools that can be used to discover dependencies from network traffic is available in [91, 104].

Finally, it is also necessary to make sure no other sub-system (or component) has been left out from the global picture. This can be done by cross-checking the global picture with the information on design documentation and (when available) with information coming from the deployed system (e.g. list of processes, traffic exchanged from hosts etc.).

Example (part 4) After Step 4 the analyst discovers the following detailed information:

1. *What applications are used to implement the sub-systems?*

- The holiday reservation application sub-system is implemented by means of a web application written in Java and running on a Tomcat servlet engine.
- The holiday reservation database sub-system is implemented as a schema in an Oracle DBMS.
- The CRM database is implemented as a schema and several stored procedures in two replicated Oracle DBMS instances. The client GUI application accesses the first instance by default, but if the instance is unavailable it automatically tries to connect to the second one (hot standby).
- The same Oracle instance is used for both the holiday reservation database and the primary CRM database.

2. *In which server(s) are the identified applications running?*

    - The web application for holiday reservation runs on Server 1.
    - The first Oracle instance runs on Server 1.
    - The second Oracle instance runs on Server 2.

3. *To which networks is each server connected?* This information is not relevant as no network component is included in the dependency graph.

4. *Are there other applications running on the servers?* Reports from the system administrators confirm no other application is running on Server 1 and Server 2.

5. *How does each component interact with the others?*

    - The web application for holiday reservation is accessed by users through the HTTPS protocol.
    - The web application for holiday reservation accesses the database through the Oracle JDBC driver.
    - The CRM GUI interacts with the Oracle database through the native Oracle driver installed on the client workstations.

Step 5: Drawing nodes and edges

At this stage, the analyst has all the information to build the dependency graph. Here we will use the notation of an *AND /OR* dependency graph, (see Chapter 5).

In this notation, each node of the dependency graph can be either an AND node (i.e. the node becomes unavailable when any of the nodes it depends on is unavailable) or an OR node (i.e. the node becomes unavailable when all the nodes it depends on are unavailable). It is however straightforward to translate this notation into the slightly different ones we adopted for timed dependency graphs in Chapter 2 and Chapter 4 (with the difference that *OR* dependencies are not included) and for timed *AND /OR* dependency graphs in Chapter 3, in which the AND/OR behaviour of a dependency is annotated in the edge modelling the dependency itself.

First, the analyst has to represent as graph nodes both the services identified during Steps 1-3 and the components identified during Step 4. All nodes can be initially modelled as AND nodes. As we described in Step 4, depending on the results of Step 2, the sub-systems that are in the analysis scope may be modelled more precisely than the ones that are outside its scope. For example, a typical modelling approach for in-scope sub-systems is to represent each server (hardware and operating system) and each application running on servers as separate node in the graph, while for out-of-scope sub-systems one single node can be used, disregarding its implementation details. It is useful to mark differently nodes that represent different types of components (e.g. locations, network components, servers, applications, services, processes, etc.), this can be done either by using a naming convention (e.g. the name of an application always starts with "APP"), or by using different colours.

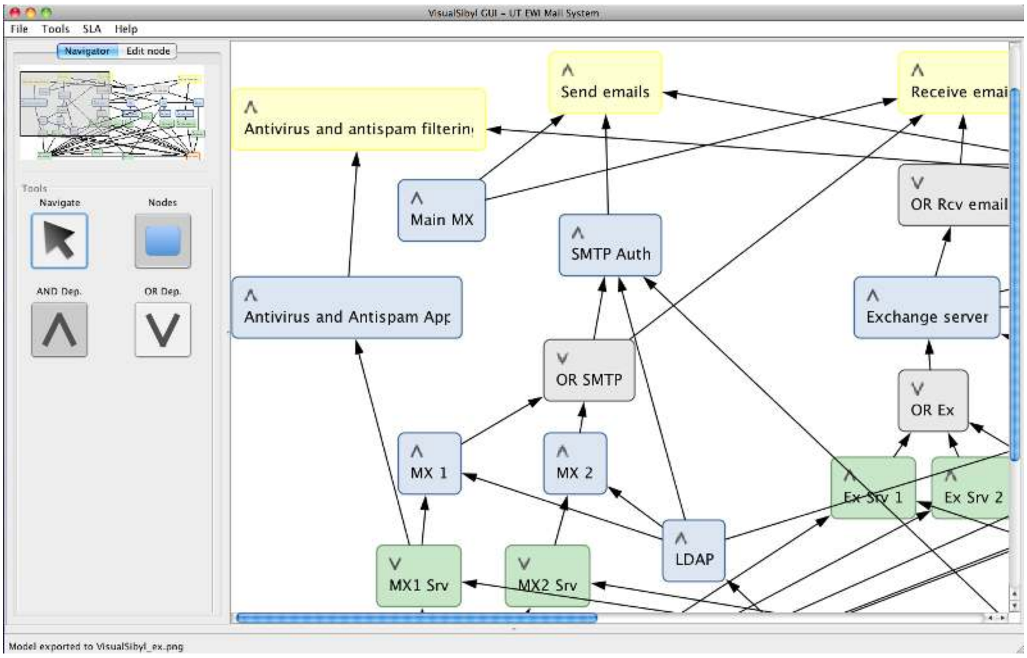
Then, nodes have to be linked together according to their dependency relations discovered during Step 4. The analyst has to link node a with node b if b depends on a in such a way that, if a is unavailable, b becomes unavailable in turn. It is useful to start with the "trivial" dependencies, for example by linking server nodes with the application nodes running on them, or linking location nodes with the server nodes deployed in those locations. Next, the analyst can link the application nodes with the service nodes describing the system functionalities. Notice that one functionality could be granted by more than one application, and there could be alternative applications providing the same functionality. In this cases there is an OR dependency among the application nodes and the service node. Since we created all AND nodes when modelling system components and functionalities, we now need to add a logic OR node to which the service node depends on. This OR node in turn depends on the applications implementing the service, but it does not model any existing physical or logical system component. Creating such OR nodes is useful since in this way one can create multiple groups of different OR relations among components. This allows the analyst to model dependencies such as a depends on $(b \vee c) \wedge (d \vee e)$, or any other combination. Finally, functional dependencies among application nodes can be modelled by following the same procedure. To this end, it is important to make sure that any dependency relation in the model represents the behaviour of the real system. A mistake one can easily make is to link together two nodes because one has a sort of functional dependency on the other, even if the failure of the former does not cause the complete failure of the latter. This kind of situation has to be handled in a different way, for example by linking the two nodes to the node that models the functionality (service) expressed by the two. In this way, if any of the two nodes fails, the functionality is unavailable and the semantic of the dependency is preserved. For non-trivial dependencies, it might be useful to annotate the edge modelling the dependency with a short description of the dependency type and a reference to the source where the dependency information was taken. This improves the readability of the the dependency graph for future analyses and helps the analyst when presenting the analysis results.

After the dependency graph has been built, it is useful to validate it against the documentation or the knowledge of the system designers/developers. To validate against documentation the analyst has to check that the components described in the documentation are all present as nodes in the dependency graph, and that all the functional relationships described in the documentation and their semantics are correctly captured in the graph. To validate against the knowledge of the system designers/developers it is useful to organise a (preferably joint) interview session in which the analyst describes the behaviour of the system starting from the dependency graph: during the interview, the analyst should make sure that nodes and dependencies are understood by the designer/developers and that they confirm the behaviour of the system in case of component failure is correctly modelled by the graph.

In theory, the analyst could describe a dependency graph by any representation technique that allows a computer program to use the representation and run the analysis algorithms (e.g. a list representing nodes and an adjacency matrix to represent edges). In practice, using tools that provide a graphic representation is of great help both to ease the creation process for the analyst and to make the representation understandable to stakeholders and interviewees.

We have developed for our case-studies a prototype GUI tool called VisualSibyl [115] which allows the analyst to do all the basic operations to draw dependency graphs (i.e. drag and drop nodes and edges, specify node types) and is available for download. VisualSibyl is based on the Netbeans Visual Libraries for graphically rendering the graph structure. It also implements the A^{2}THOS algorithms we described in Chapter 5. Figure B.2 shows the drawing of a dependency graph using VisualSibyl.

An alternative choice to VisualSibyl consists of using Microsoft Visio: the



Model exported to VisualSibyl_ex.png

Figure B.2: Using VisualSibyl to draw dependency graphs.

analysis algorithms can then be implemented through Visio macros written in the Visual Basic Script language.

Example (part 5) The results of Step 5 are shown in figure B.3. According to the results of Step 4 there are 7 identified nodes: two services (`eHoliday` and `CRM repository`), three applications (`Holiday reservation WebApp`, `Oracle DB instance 1` and `Oracle DB instance 2`) and two servers (`Server 1` and `Server 2`). All nodes have been represented as AND nodes and coloured according to their type.

The identified dependencies are 7 and are also derived from the detailed information of Step 4. Both `Holiday reservation WebApp` and `Oracle DB instance 1` depend on `Server 1`, while `Oracle DB instance 2` depends on `Server 2`. The `eHoliday` service depends on both the `Holiday reservation WebApp` and the `Oracle DB instance 1` to work properly, as the failure of one of the two nodes would cause the stop of the holiday management functionality. The `CRM repository` service depends on the two Oracle database instances to run: in this case we have an OR dependency, as both instances need to be unavailable for the service to be unavailable as well.

Therefore, to represent the OR dependency, we add an OR node on which the `CRM Repository` service depends on. In turn, the OR node depends on both `Oracle DB instance 1` and `Oracle DB instance 2`.

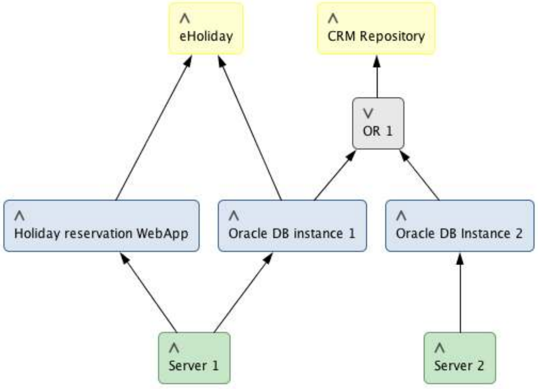


Figure B.3: The dependency graph resulting from the running example of Chapter 3.

Author References

Journal Publications

[1] E. Zambon, S. Etalle, and R.J. Wieringa. A^{2}THOS: Availability Analysis and Optimisation in SLAs. *International Journal of Network Management*, 2010. Submitted for publication in April 2010.

[2] E. Zambon, S. Etalle, R.J. Wieringa, and P.H. Hartel. Model-based Qualitative Risk Assessment for Availability of IT Infrastructures. *SOSYM: Software and System Modelling*, pages 1–28, 2010.

Refereed Conferences

[3] D. Bolzoni, E. Zambon, S. Etalle, and P.H. Hartel. Poseidon: a 2-tier Anomaly-based Network Intrusion Detection System. In *Proc: 4th IEEE Int. Information Assurance Workshop (IWIA2006)*, pages 144–156, London, UK, April 2006. IEEE Computer Society Press.

[4] A. Morali, E. Zambon, S. H. Houmb, K. Sallhammar, and S. Etalle. Extended eTVRA vs. Security Checklist: Experiences in a Value-Web. In *ICSE '09: Proc. of the 31th IEEE International Conference on Software Engineering*, IEEE, pages 130–140. IEEE Computer Society Press, May 2009.

[5] E. Zambon, D. Bolzoni, S. Etalle, and M. Salvato. A model supporting Business Continuity auditing & planning in Information Systems. In *Proc. of the First International Conference on Global Defense and Business Continuity (ICIMP&BC '07)*, IEEE, pages 33–42. IEEE Computer Society Press, 2007. (Subsumed by Chapter 4 of this thesis).

International Workshops

[6] A. Morali, E. Zambon, S. Etalle, and P. Overbeek. IT Confidentiality Risk Assessment for an Architecture-Based Approach. In *BDIM '08: Third IEEE International Workshop on Business-Driven IT Management*, IEEE, pages 31–40. IEEE Computer Society Press, 2008.

[7] E. Zambon, D. Bolzoni, S. Etalle, and M. Salvato. Model-Based Mitigation of Availability Risks. In *BDIM '07: Second IEEE/IFIP International*

Workshop on Business-Driven IT Management, pages 75–83, Munich, May 2007. IEEE Computer Society Press. (Subsumed by Chapter 2 of this thesis).

General References

[8] J. Ø. Aagedal, F. den Braber, T. Dimitrakos, B. A. Gran, D. Raptis, and K. Stëlen. Model-Based Risk Assessment to Improve Enterprise Security. In *EDOC '02: Proc. 6th International Enterprise Distrubuted Object Computing Conference*, pages 51–63. IEEE Computer Society, 2002.

[9] D. Ardagna and B. Pernici. Global and Local QoS Guarantee in Web Service Selection. In *Business Process Management Workshops*, pages 32–46, 2005.

[10] Y. Asnar and P. Giorgini. Modelling Risk and Identifying Countermeasure in Organizations. Technical report, University of Trento, 2006. oai:UNITN.Eprints:1035.

[11] S. Bagchi, G. Kar, and J. Hellerstein. Dependency Analysis in Distributed Systems using Fault Injection: Application to Problem Determination in an e-commerce Environment. In *DSOM '01: Proc. 2001 International Workshop on Distributed Systems: Operations & Management* , 2001. `http://www.research.ibm.com/PM/DSOM2001_dependency_final.pdf`.

[12] F. Baiardi, S. Suin, C. Telmon, and M. Pioli. Assessing the Risk of an Information Infrastructure Through Security Dependencies. *Critical Information Infrastructures Security*, 4347/2006:42–54, 2006.

[13] R.E. Barlow and F. Proschan. *Mathematical Theory of Reliability*. SIAM: Society for Industrial and Applied Mathematics Philadelphia, 1996.

[14] J. Bengtsson and W. Yi. Timed Automata: Semantics, Algorithms and Tools. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *LNCS*, pages 87–124. Springer-Verlag, 2003.

[15] S.P. Bennet and M.P. Kailay. An Application of Qualitative Risk Analysis to Computer Security for the Commercial Sector. In *Eighth Annual Computer Security Applications Conference*, pages 64–73. IEEE Computer Society Press, April 1992. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?isnumber=5913&arnumber=228232&count=25&index=15`.

[16] J. Bennett. Business continuity and availability planning. *Infosecurity*, 4(1754-4548):38, 2007.

[17] H. Boudali, P. Crouzen, and M.I.A. Stoelinga. A compositional semantics for Dynamic Fault Trees in terms of Interactive Markov Chains. In *Proc. of the 5th International Symposium on Automated Technology for Verification and Analysis*, pages 441–456. LNCS, 2007.

[18] P. Bowen, J. Hash, and M. Wilson. NIST SP 800-100 - Information Security Handbook: A Guide for Managers. Technical report, NIST National Institute of Standards and Technology, 2006.

[19] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. Technical report, University of Trento, 2002. oai:UNITN.Eprints:84.

[20] A. Brown, G. Kar, and A. Keller. An Active Approach to Characterizing Dynamic Dependencies for Problem Determination in a Distributed Application Environment. In *IM '01: IEEE/IFIP International Symposium on Integrated Network Management*, pages 377–390, 2001.

[21] BS 7799-3 - Information security management systems - Part 3: Guidelines for information security risk management, 2006.

[22] BSI. *BS IEC 61882:2001 : Hazard and operability studies (HAZOP studies). Application guide.* British Standards Institute, 2001.

[23] M. Burgess. System Administration and Micro-Economic Modelling. In J. Bergstra and M. Burgess, editors, *Handbook of Network and System Administration*, chapter 6.3, pages 729–773. Elsevier, 2007.

[24] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[25] L. Cloth and B. R. Haverkort. Model Checking for Survivability. In *QUEST '05: Proc. 2nd Int. Conference on the Quantitative Evaluation of Systems*, pages 145–154. IEEE Computer Society, 2005.

[26] R. Cocchiara. Beyond disaster recovery: becoming a resilient business. Technical report, IBM, 2005. http://ibm.com/services/its/resilience.

[27] B. Cunningham, T. Dykstra, E. Fuller, C. Gatford, A. Gold, M.P. Hoagberg, A. Hubbard, C. Little, S. Manzuik, G. Miles, C.F. Morgan, K. Pfeil, R. Rogers, T. Schack, and S. Snedaker. *The Best Damn IT Security Management Book Period.* Syngress Publishing, November 2007.

[28] F. den Braber, I. Hogganvik, M. S. Lund, K. Stolen, and F. Vraalsen. Model-based security analysis in seven steps — a guided tour to the CORAS method. *BT Technology Journal*, 25(1):101–117, 2007.

[29] S. Distefano and L. Xing. A New Approach to Modeling the System Reliability: Dynamic Reliability Block Diagrams. In *RAMS '06: Annual Reliability and Maintainability Symposium*, pages 189–195, Washington, DC, USA, 2006. IEEE Computer Society.

[30] J.B. Dugan, S.J. Bavuso, and M.A. Boyd. Dynamic Fault-Tree Models for Fault-Tolerant Computer Systems. *IEEE Transactions on Reliability*, 41(3):363–377, September 1992.

[31] Elsevier, editor. *Handbook of Constraint Programming*. F. Rossi, P. van Beek and T. Walsh, August 2006.

[32] A. Evangelidis, J. Akomode, A. Taleb-Bendiab, and M. Taylor. Risk Assessment & Success Factors for e-Government in a UK Establishment. In *Electronic Government*, volume 2456/2002, pages 93–99. Springer Berlin / Heidelberg, 2002.

[33] F. Flammini, N. Mazzocca, M. Iacono, and S. Marrone. Using Repairable Fault Trees for the evaluation of design choiches for critical repairable systems. In *HASE '05: Proc. Ninth IEEE International Symposium on High-Assurance Systems Engineering*, pages 163–172, Washington, DC, USA, 2005. IEEE Computer Society.

[34] K. Goseva-Popstojanova, A. Hassan, A. Guedem, W. Abdelmoez, D.E.M. Nassar, H. Ammar, and A. Mili. Architectural-level risk analysis using UML. *IEEE Transactions on Software Engineering*, 29:946 – 960, October 2003.

[35] X. Gu, K. Nahrstedt, R.N. Chang, and C. Ward. QoS-Assured Service Composition in Managed Service Overlay Networks. In *International Conference on Distributed Computing Systems*, page 194, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[36] C.A. Gunter, E.L. Gunter, M.A. Jackson, and P. Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37–43, May/June 2000.

[37] D.S. Herrmann. *Complete Guide to Security and Privacy Metrics*. Auerbach Publications, Boston, MA, USA, 2007.

[38] G. J. Holzmann. *The SPIN model checker*. Addison-Wesley, 2003.

[39] K.J. Soo Hoo. *How much is enough: a risk management approach to computer security*. PhD thesis, Stanford University, Stanford, CA, USA, 2000.

[40] F. Innerhofer-Oberperfler and R. Breu. Using an Enterprise Architecture for IT Risk Management. In *ISSA '06: Proc. Information Security South Africa Conference*, 2006. URL: http://icsa.cs.up.ac.za/issa/2006/Proceedings/Full/115_Paper.pdf.

[41] British Standards Institute. BS 25999-1 - Business continuity management - Part1: Code of practice, 2006.

[42] ISO/IEC 13335:2001 - Information Technology - Security techniques - Guidelines for the management of IT security, 2001.

[43] ISO/IEC 15408:2006 - Common Criteria for Information Technology Security Evaluation. `http://www.commoncriteriaportal.org/thecc.html`, September 2006.

[44] ISO/IEC 17799:2000 - Information Security - Code of Practice for Information Security Management, 2000.

[45] ISO/IEC 27001:2005 - Information technology – Security techniques – Information security management systems – Requirements, 2005.

[46] ISO/IEC 27002:2005 - Information technology – Security techniques – Code of practice for information security management, 2005.

[47] S. Jha and J. M. Wing. Survivability analysis of networked systems. In *ICSE '01: Proc. 23rd Int. Conference on Software Engineering*, pages 307–317. IEEE Computer Society, 2001.

[48] A.M. Jrad, C.K. Chan, and T.B. Morawski. Incorporating the downtime due to disaster events in the network reliability model. In *NETWORKS 2004: Proc. 11th International Telecommunications Network Strategy and Planning Symposium*, pages 365–371. IEEE Computer Society, 2004.

[49] G. Kar, A. Keller, and S. Calo. Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis. In *NOMS '00: Proc. of the 7th IEEE/IFIP Network Operations and Management Symposium*, pages 61–75. IEEE Press, 2000.

[50] I-J. Kim, Y-J. Jung, JG. Park, and D. Won. A Study on Security Risk Modeling over Information and Communication Infrastructure. In *SAM '04: Proc. of the International Conference on Security and Management*, pages 249–253. CSREA Press, June 2004.

[51] W. Lam. Ensuring Business Continuity. *IT Professional*, 4(3):19–25, 2002.

[52] K. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Romijn. As Cheap as Possible: Efficient Cost-Optimal Reachability for Priced Timed Automata. *LNCS*, 2102:493–506, 2001.

[53] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.

[54] C. Leangsuksun, H. Song, and L. Shen. Reliability Modeling Using UML. In *Software Engineering Research and Practice*, pages 259–262, 2003.

[55] A. Lenstra and T. Voss. Information Security Risk Assessment, Aggregation, and Mitigation. In *ACISP '04: Information Security and Privacy: Australasian Conference*, 2004.

[56] R.P. Lippmann and K.W. Ingols. An Annotated Review of Past Papers on Attack Graphs. Technical report, Defense Technical Information Center OAI-PMH Repository [http://stinet.dtic.mil/oai/oai] (United States), 1998. http://en.scientificcommons.org/18618950.

[57] Z. Liu and M. Joseph. Verification of Fault Tolerance and Real Time. In *FTCS '96: 26th IEEE Symposium on Fault Tolerant Computing Systems*, pages 220–229. IEEE Computer Society, 1996.

[58] H. Maciejewskia and D. Caban. Estimation of repairable system availability within fixed time horizon. *Reliability Engineering & System Safety*, 93(1):100–106, January 2006.

[59] Military Standard MIL-STD-1629A. *Procedures for Performing a Failure Mode, Effects and Criticality Analysis*. USA Department of Defense, November 1980.

[60] R.R. Muntz, E. de Souza e Silva, and A. Goyal. Bounding Availability of Repairable Computer Systems. *SIGMETRICS Performance Evaluation Review*, 17(1):29–38, 1989.

[61] R. L. Murphy, C. J. Alberts, R. C. Williams, R. P. Higuera, A. J. Dorofee, and J. A. Walker. *Continuous Risk Management Guidebook*. Carnegie Mellon Software Engineering Institute, 1996.

[62] Office of Government Commerce (OGC). *Introduction to the ITIL Service Lifecycle*. TSO (The Stationery Office), 2007.

[63] Office of Government Commerce (OGC). *ITIL Version 3 Service Design*. TSO (The Stationery Office), 2007.

[64] Office of Government Commerce (OGC). *ITIL Version 3 Service Strategy*. TSO (The Stationery Office), 2007.

[65] M. C. Paulk, C. V. Weber, B. Curtis, and M. B. Chrissis. *The capability maturity model: guidelines for improving the software process*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[66] R. Pawson and N. Tilley. *Realistic Evaluation*. Sage Publications, 1997.

[67] T. Reitan. System Reliability. In J. Bergstra and M. Burgess, editors, *Handbook of Network and System Administration*, chapter 6.4, pages 775–809. Elsevier, 2007.

[68] S. Ross. *Introduction to Probability Models, Seventh Edition*. Harcourt Academic Press, 1989.

[69] J.E.Y. Rossebo, S. Cadzow, and P. Sijben. eTVRA, a Threat, Vulnerability and Risk Assessment Method and Tool for eEurope. In *ARES '07: Second International Conference on Availability, Reliability and Security*, pages 925–933. IEEE Computer Society Press, April 2007. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4159893`.

[70] S. A. Sayana. Auditing Business Continuity. *Information Systems Control Journal*, 1:11–13, 2005. http://www.isaca.org/TemplateRedirect.cfm?/template=/ContentManagement/ContentDisplay.cfm&ContentID=23553.

[71] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated Generation and Analysis of Attack Graphs. *IEEE Symposium on Security and Privacy*, page 273, 2002.

[72] S. Smale. On the average number of steps of the simplex method of linear programming. *Mathematical Programming*, 27(3):241–262, October 1983.

[73] G. Stoneburner, A. Goguen, and A. Feringa. NIST SP 800-30 - Risk Management Guide for Information Technology Systems. Technical report, NIST National Institute of Standards and Technology, 2002.

[74] K.J. Sullivan, J.B. Dugan, and D. Coppit. The Galileo fault tree analysis tool. In *Proc. of Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing*, pages 232–235. IEEE Computer Society, 1999.

[75] J.G. Torres-Toledano and L.E. Sucar. Bayesian Networks for Reliability Analysis of Complex Systems. In *IBERAMIA '98: Proceedings of the 6th Ibero-American Conference on AI*, pages 195–206, London, UK, 1998. Springer-Verlag.

[76] W.E. Vesely, F.F. Goldberg, N.H. Roberts, and D.F. Haasl. Fault Tree Handbook. Technical report, US Nuclear Regulatory Commission NUREG-0492, 1981.

[77] R.J. Wieringa and J.M.G. Heerkens. Designing requirements engineering research. In *CERE '07: Workshop on Comparative Evaluation in Requirements Engineering*, pages 36–48. IEEE Computer Society Press, October 2007. `http://eprints.eemcs.utwente.nl/13002/`.

[78] R.J. Wieringa, N. Maiden, N. Mead, and C. Rolland. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering Journal*, 11:102–107, November 2006.

[79] R. Winther, O. Johnsen, and B.A. Gran. Security assessments for safety critical systems using hazops. In *Proc. of SAFECOMP 2001*, pages 14–24. Springer, 2001.

[80] T. Yu and K-J. Lin. Service Selection Algorithms for Web Services with End-to-End QoS Constraints. In *IEEE International Conference on E-Commerce Technology*, pages 129–136, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[81] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1291834`.

Web References (Last Accessed: October 2010)

[82] C. J. Alberts and A. J. Dorofee. OCTAVE Criteria. Technical Report ESC-TR-2001-016, Carnegie Mellon - Software Engineering Institute, December 2001. http://www.cert.org/octave/.

[83] Alion Science and Technology. CounterMeasures. http://www.countermeasures.com.

[84] The ArchiMate project, 2009. http://www.archimate.org.

[85] Risk management - AS/NZS 4360:2004. http://www.riskmanagement.com.au/, October 2004.

[86] Basel II: Revised international capital framework. http://www.bis.org/publ/bcbsca.htm, 2005.

[87] BiZZdesign. Architect. http://www.bizzdesign.nl/index.php/tools/bizzdesignarchitect, 2010.

[88] BlockSim: System Reliability and Maintainability Analysis Software Tool. http://www.reliasoft.com/BlockSim/.

[89] CISCO Systems. Cisco 2007 Annual Security Report. http://www.cisco.com/web/about/security/cspo/docs/Cisco2007Annual_Security_Report.pdf, 2007.

[90] CobiT 4.1 - Control Objectives for Information and related Technology. http://www.isaca.org, 2007.

[91] L. Cottrell. Network Monitoring Tools. http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html, 2010.

[92] CRAMM v5.1 Information Security Toolkit. http://www.cramm.com, 2009.

[93] A. Deladrière and M. Morrison. The risk management challenge. http://www.bankingfinance.be/40915/default.aspx, March 2008.

[94] EBIOS - Expression des Besoins et Identification des Objectifs de Sécurité - Section 2: Approach. http://www.ssi.gouv.fr/en/, 2004.

[95] The ECLiPSe Constraint Programming System. http://87.230.22.228/.

[96] ENISA. Risk Management: Implementation principles and Inventories for Risk Management/Risk Assessment methods and tools. Technical report, European Network and Information Security Agency (ENISA), June 2006. http://www.enisa.europa.eu/rmra/rm_home.html.

[97] Federal Office for Information Security (BSI). GSTool. http://www.bsi.bund.de/english/gstool/.

[98] HP. HP Business Availability Center. https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-15-25_4000_100__, August 2009.

[99] Hyperic. http://www.hyperic.com.

[100] IBM. IBM Tivoli. http://www.ibm.com/software/tivoli/, August 2009.

[101] BSI Standard 100-1 - Information Security Management Systems (ISMS). http://www.bsi.de/english/gshb/, 2005.

[102] http://www.kpmg.com.

[103] R. Allen Long. Beauty & the Beast – Use and Abuse of Fault Tree as a Tool. http://www.fault-tree.net/papers/long-beauty-and-beast.pdf, Last checked in 2010.

[104] R. Marty. SecViz - Security Visualization. http://www.secviz.org/, 2010.

[105] McAfee. In the Crossfire - Critical Infrastructure in the Age of Cyber War. http://resources.mcafee.com/content/NACIPReport, 2010.

[106] MEHARI 2007 - Risk Analysis Guide. https://www.clusif.asso.fr/en/clusif/present/, April 2007.

[107] Nagios. http://www.nagios.org.

[108] NIST National Vulnerability Database. http://nvd.nist.gov/, 2009.

[109] PriceWaterhouseCoopers. BERR Information Security Breaches Survey 2008. http://www.pwc.co.uk/pdf/BERR_ISBS_2008(sml).pdf, 2008.

[110] Relex Software Corporation. http://www.relex.com.

[111] Shadow-Planner, Business Continuity Management software. http://www.office-shadow.com/.

[112] Sarbanes-Oxley Act of 2002. http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=107_cong_bills&docid=f:h3763enr.tst.pdf, 2002.

[113] The Open Group. TOGAF (The Open Group Architecture Framework), 2003. http://www.opengroup.org/architecture/togaf8-doc/arch/.

[114] The Zachman Institute for Framework Advancement. Zachman Framework. http://www.zifa.com/, 2007.

[115] Emmanuele Zambon. VisualSibyl: a GUI frontend for A^{2}THOS. http://www.vf.utwente.nl/~zambone/visual_sybil, 2010.

Nomenclature

BC	Business Continuity
BCP	Business Continuity Plan
BIA	Business Impact Analysis
COBIT	Control Objectives for Information and related Technology
CRM	Customer Relationship Management
CTMC	Continuous Time Markov Chain
DoS	Denial of Service
FT	Fault Tree
GIT	Global IT Infrastructure
IT	Information Technology
ITIL	Information Technology Infrastructure Library
MTBF	Mean Time Between Failure
MTPD	Maximum Tolerable Period of Disruption
MTTR	Mean Time To Repair
QualTD	Qualitative Time Dependency
RA	Risk Assessment
RBD	Reliability Block Diagram

RM	Risk Mitigation
RMC	Risk Management and Compliance
RPO	Recovery Point Objective
RTO	Recovery Time objective
SLA	Service Level Agreement
SLM	Service Level Management
TD	Time Dependency
TDR	Time Dependency and Recovery
ToA	Target of Assessment
TVA	Threat/Vulnerability Assessment

Samenvatting

De beschikbaarheid van de IT-infrastructuur binnen een organisatie is van essentieel belang voor de ondersteuning van bedrijfsactiviteiten. IT-uitval veroorzaakt concurrerende aansprakelijkheid, tast de financiële prestaties en reputatie van een bedrijf aan. Om het maximum aan IT-beschikbaarheid te bereiken binnen het beschikbare budget, moeten organisaties een reeks analyseactiviteiten uitvoeren om prioriteiten vast te stellen en beslissingen te nemen op basis van de bedrijfsbehoeften. Deze reeks analyseactiviteiten wordt: "IT-beschikbaarheidsplanning" genoemd.

De meeste (grote) organisaties leiden de IT-beschikbaarheidsplanning af van één of meer van de drie belangrijkste facetten: informatie risico beheer, bedrijfscontinuïteit en service level-beheer. Informatie risico beheer bestaat uit het identificeren, analyseren, evalueren en tegengaan van risicos die de informatie, verwerkt door een organisatie, en de informatieverwerkings-systemen kan beïnvloeden. Bedrijfscontinuïteit bestaat uit het creëren van een logistiek plan, genoemd bedrijfscontinuïteitsplan, die de procedures en alle benodigde informatie bevat om de cruciale processen van een bedrijf te herstellen na een ernstige verstoring. Service level-beheer bestaat hoofdzakelijk uit het organiseren, documenteren en vezekeren van een bepaald kwaliteitsniveau (bijv. de beschikbaarheid) van de door het IT-systeem aangeboden diensten aan de business units van een organisatie.

Er bestaan verscheidene standaarddocumenten welke een organisatie voorzien van richtlijnen voor het opzetten van processen risico-, bedrijfscontinuïteit- and service level-beheer. Echter, om te zorgen dat deze richtlijnen zo algemeen mogelijk toepasbaar zijn omvatten ze geen implementatie details. Derhalve dient elke organisatie afzonderlijk een IT-beschikbaarheidsplanning te ontwikkelen naar eigen behoefte. Om praktisch bruikbaar te zijn moeten deze technieken nauwkeurig genoeg zijn om de stijgende complexiteit van de IT-infrastructuur te ondervangen en daarnaast uitvoerbaar blijven binnen het beschikbare budget van de organisatie. Zoals we in dit proefschrift beargumenteren, basisbenaderingen die tegenwoordig toegepast worden door organisaties zijn haalbaar maar vaak met een gebrek aan nauwkeurigheid.

In dit proefschrift presenteren we een graafgebaseerd framework voor de beschikbaarheidsafhankelijkheden van de componenten van een IT-infrastructuur èn ontwikkelen we technieken gebaseerd op dit framework om de IT-beschikbaarheidsplanning te ondersteunen. In meer detail behandelen we:

- het "Time Dependency model", dat IT-managers ondersteunt bij het vaststellen van een reeks tegenmaatregelen om IT-beschikbaarheids gerelateerde risico's te verlagen met minimale kosten;
- het "Qualitative Time Dependency model", dat bedoeld is om systematisch de aanwezigheid van IT-beschikbaarheidsgerelateerde risico's vast te stellen in combinatie met bestaande risicoanalysemethodiek;
- het "Time Dependency and Recovery model", dat een middel verstrekt aan IT-managers om hesteltijd doeleinden op de componenten van een IT-architectuur te bepalen of te bevestigen, die vervolgens worden gebruikt om het IT-gerelateerde gedeelte van een bedrijfscontinuïteitsplan te creëren;
- A^{2}THOS, om te controleren of de beschikbare SLA's, die de provisioning van de IT-diensten tussen de business units van dezelfde organisatie reguleren, kunnen worden nageleefd wanneer de uitvoering van deze diensten gedeeltelijk is uitbesteed aan externe bedrijven, èn om dienovereenkomstig te kiezen voor aanbiedingen van externe bronnen.

We doen case studies om met behulp van de gegevens van een primaire verzekeringsmaatschappij en een grote multinational de voorgestelde technieken te testen. De resultaten laten zien dat organisaties zoals verzekeringsmaatschappijen of fabrikanten, welke gebruik maken van IT, ter ondersteuning van hun bedrijf, kunnen profiteren van de optimalisering van de beschikbaarheid van hun IT-infrastructuur. Het is mogelijk technieken te ontwikkelen die de IT-beschikbaarheidsplanning ondersteunen en realiseerbaar zijn met beperkte uitgaven. Het door ons voorgestelde framework laat zien dat de structuur van de IT-architectuur praktisch kan worden toegepast met dergelijke technieken om de nauwkeurigheid ten opzichte van de huidige praktijk te verhogen.

Titles in the IPA Dissertation Series since 2005

E. Ábrahám. *An Assertional Proof System for Multithreaded Java - Theory and Tool Support- .* Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue.* Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms.* Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications.* Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures.* Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability.* Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions.* Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments.* Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression.* Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages.* Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations.* Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics.* Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems.* Faculty of

Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting*. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs*. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations*. Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data*. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data*

Space. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization*. Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML*. Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models*. Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications*. Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition*. Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems*. Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains*. Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use*. Faculty of Science, Mathematics and Computer Science, RU. 2007-10

J.A.R. Noppen. *Imperfect Information in Software Development Processes*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems*. Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML*. Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery*. Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J. Mathijssen. *Logical Calculi for Reasoning with Binding*. Faculty of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks*. Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data*. Faculty of Mathematics and Computer Science, TU/e. 2007-19

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical

Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates*. Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines*. Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras*. Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit*. Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

M. Farshi. *A Theoretical and Experimental Study of Geometric Networks*. Faculty of Mathematics and Computer Science, TU/e. 2008-12

G. Gulesir. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

F.D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. Faculty of Science, Mathematics and Computer Science, RU. 2008-14

P. E. A. Dürr. *Resource-based Verification for Robust Composition of Aspects*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

E.M. Bortnik. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calamé. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation*. Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. Brijder. *Models of Natural Computation: Gene Assembly and Membrane Systems*. Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. Koprowski. *Termination of Rewriting and Its Certification*. Faculty of Mathematics and Computer Science, TU/e. 2008-24

U. Khadim. *Process Algebras for Hybrid Systems: Comparison and Development*. Faculty of Mathematics and Computer Science, TU/e. 2008-25

J. Markovski. *Real and Stochastic Time in Process Algebras for Performance Evaluation*. Faculty of Mathematics and Computer Science, TU/e. 2008-26

H. Kastenberg. *Graph-Based Software Specification and Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

I.R. Buhan. *Cryptographic Keys from Noisy Data Theory and Applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

R.S. Marin-Perianu. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean*. Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques*. Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. Rodriguez Yakushev. *Towards Getting Generic Programming Ready for Prime Time*. Faculty of Science, UU. 2009-9

K.R. Olmos Joffré. *Strategies for Context Sensitive Program Transformation*. Faculty of Science, UU. 2009-10

J.A.G.M. van den Berg. *Reasoning about Java programs in PVS using JML*. Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. Khatib. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

H.L. Jonker. *Security Matters: Privacy in Voting and Fairness in Digital Exchange*. Faculty of Mathematics and Computer Science, TU/e. 2009-15

M.R. Czenko. *TuLiP - Reshaping Trust Management*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

T. Chen. *Clocks, Dice and Processes*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

C. Kaliszyk. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web*. Faculty of Science, Mathematics and Computer Science, RU. 2009-18

R.S.S. O'Connor. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory*. Faculty of Science, Mathematics and Computer Science, RU. 2009-19

B. Ploeger. *Improved Verification Methods for Concurrent Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-20

T. Han. *Diagnosis, Synthesis and Analysis of Probabilistic Models*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

R. Li. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis*. Faculty of Mathematics and Natural Sciences, UL. 2009-22

J.H.P. Kwisthout. *The Computational Complexity of Probabilistic*

Networks. Faculty of Science, UU. 2009-23

T.K. Cocx. *Algorithmic Tools for Data-Oriented Law Enforcement*. Faculty of Mathematics and Natural Sciences, UL. 2009-24

A.I. Baars. *Embedded Compilers*. Faculty of Science, UU. 2009-25

M.A.C. Dekker. *Flexible Access Control for Dynamic Collaborative Environments*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

J.F.J. Laros. *Metrics and Visualisation for Crime Analysis and Genomics*. Faculty of Mathematics and Natural Sciences, UL. 2009-27

C.J. Boogerd. *Focusing Automatic Code Inspections*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

M.R. Neuhäußer. *Model Checking Nondeterministic and Randomly Timed Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

J. Endrullis. *Termination and Productivity*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

T. Staijen. *Graph-Based Specification and Verification for Aspect-Oriented Languages*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

Y. Wang. *Epistemic Modelling and Protocol Dynamics*. Faculty of Science, UvA. 2010-05

J.K. Berendsen. *Abstraction, Prices and Probability in Model Checking Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2010-06

A. Nugroho. *The Effects of UML Modeling on the Quality of Software*. Faculty of Mathematics and Natural Sciences, UL. 2010-07

A. Silva. *Kleene Coalgebra*. Faculty of Science, Mathematics and Computer Science, RU. 2010-08

J.S. de Bruin. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications*. Faculty of Mathematics and Natural Sciences, UL. 2010-09

D. Costa. *Formal Models for Component Connectors*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

M.M. Jaghoori. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services*. Faculty of Mathematics and Natural Sciences, UL. 2010-11

R. Bakhshi. *Gossiping Models: Formal Analysis of Epidemic Protocols*. Faculty of Sciences, Department of Computer Science, VUA. 2011-01

B.J. Arnoldus. *An Illumination of the Template Enigma: Software Code Generation with Templates*. Faculty of Mathematics and Computer Science, TU/e. 2011-02

E. Zambon. *Towards Optimal IT Availability Planning: Methods and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03