UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

GUILHERME SPERB MACHADO

Rollback Support in IT Change Management Systems

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Prof. Dr. Lisandro Zambenedetti Granville
Advisor

Porto Alegre, March 2009

CIP – CATALOGING-IN-PUBLICATION

Machado, Guilherme Sperb

Rollback Support in IT Change Management Systems / Guilherme Sperb Machado. – Porto Alegre: PPGC da UFRGS, 2009.

55 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2009. Advisor: Lisandro Zambenedetti Granville.

1. IT Change Management. 2. Rollback plan. 3. Change failures. 4. ITIL. I. Granville, Lisandro Zambenedetti. II. Título.

First of all, to God.
To my parents Liana and Sezefredo,
my brothers Tiago and Augusto,
my special girlfriend Cynthia,
and my grandmother Maria,
who launched me into the computer world.
In the memory of my grandfather, Lydio.

"If a cluttered desk is a sign of a cluttered mind,
of what then, is an empty desk?"
– Albert Einstein

ACKNOWLEDGEMENTS

First of all, I want to thank God. Without the Lord, this thesis would never ever become possible. God gave me the power and motivation to turn concrete the research.

Second, but not less important, I want to thank my parents Liana and Sezefredo for all the conditions granted through my life. They supported me through hard moments, and taught me innumerous lessons. However, one of them I can consider the most precious: reach your goals in a honest manner so you will be always remembered. Also, I want to thank my brothers Tiago and Augusto that, even do not understanding what I really do for a living (researching what?!), they somehow participated positively during my Master's degree. Mainly playing some games together to relax after paper deadlines. My grandmother Maria, I have no words to express my gratitude for giving me that gift which, by coincidence or not, opened many doors in my life. My grandfather Lydio, I wish you were here to see all of this.

Special thanks for Profs. Ana Cristina Benso and João Batista Oliveira, who gave me the opportunity to "fall in love" with the academic world. I can say that they stimulated me in order to use my knowledge to really produce something in an advantageous aspect. Thanks again. Also, my regards for colleagues and friends from the Pontifical Catholic University of Rio Grande do Sul (PUCRS) by their support and friendship.

Thanks to my Master's colleague Weverton Cordeiro, who participated actively in my research. Opinions, discussions (hard ones), and laughs (not necessary about the academic world) were part of these two years together. Among other things, during our stay in Bristol, UK, I've learned with him how early people should arrive in the airport for a flight. It was terrible. Weverton, success for you in the PhD, and always count on me for anything that you need.

Talking about Bristol, UK, special thanks to David Trastour, Abdel Boulmakoul, Claudio Bartolini, and Robert Fink. Not only for the opportunity to develop research inside a big Lab (HP Labs), but also to make my stay in the UK a great time.

I cannot forget to mention Profs. Lisandro Granville and Luciano Paschoal. They really taught me on how to be a top researcher, and I will never ever forget those lessons. Thanks a lot. I owe much to you.

Thanks to the members of the II/UFRGS Network Group. We learn together so many things, and all of you helped me somehow to conclude this thesis: Ewerton, Raniery, Clarissa, Giovane, Juliano, Roben, Jeferson, Flávio, Fábio, Alan, Cristiano, and Fabrício.

Finally, thanks to my girlfriend Cynthia, who probably suffered a lot during these two years. However, we overcame things like an endless paper deadline, travel to conferences, and the internship in HP Labs. More challenges are bounded to come, and I'm sure we have strength enough to endure. Anyway, thanks for your immense comprehension and support.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer a Deus. Sem Ele, essa dissertação de mestrado nunca seria possível. Deus me concedeu forças e motivação para tornar concreta essa pesquisa.

Em segundo lugar, mas não menos importante, gostaria de agradecer aos meus pais Liana e Sezefredo por todas as condições que tive durante a minha vida. Eles me ajudaram em momentos difíceis, e me deram inúmeros ensinametos. Porém, um deles eu posso considerar o mais precioso: atinja todos os seus objetivos de uma maneira honesta que todos irão se lembrar de você. Também, gostaria de agradecer aos meus irmãos Tiago e Augusto, que, mesmo não entendendo o que eu realmente faço no mundo acadêmico (pesquisando o quê?!), eles de alguma forma participaram positivamente durante a conclusão do mestrado. Principalmente jogando alguns jogos no video-game para tirar o *stress* após a entrega de artigos para conferências. Minha vó Maria, eu não tenho palavras para expressar minha gratidão quando me deste aquele presente que, por coincidência ou não, abriu muitas portas na minha vida. Ao meu vô Lydio, eu realmente queria que você estivesse aqui para ver tudo isso.

Agradecimentos especiais aos Profs. Ana Cristina Benso e João Batista Oliveira, pois me deram a oportunidade para me "apaixonar" pelo mundo acadêmico. Eu posso dizer que estes professores me estimularam a usar o meu conhecimento para realmente produzir algo em um aspecto vantajoso. Obrigado novamente. Também, comprimentos aos meus colegas e amigos da PUCRS.

Obrigado ao meu colega de mestrado Weverton Cordeiro, quem participou ativamente da minha pesquisa. Opiniões, discussões (fortes e filosóficas), e gargalhadas (não necessariamente sobre o mundo acadêmico), foram parte desses dois anos juntos. Entre outras coisas, durante a nossa estadia em Bristol, GB, aprendi com ele o quão cedo as pessoas precisam chegar no aeroporto para um vôo. Foi terrível. Weverton, muito sucesso no Doutorado, e sempre conte comigo para qualquer coisa que precisar.

Falando sobre Bristol, GB, agradecimentos especiais ao David Trastour, Abdel Boulmakoul, Claudio Bartolini, e Robert Fink. Não só pela oportunidade de desenvolver pesquisa de ponta dentro de um grande laboratório como a HP, mas também por tornar a nossa estadia no Reino Unido um ótimo momento.

Não posso esquecer de mencionar os Profs. Lisandro Granville e Luciano Paschoal. Eles realmente me ensinaram como ser um pesquisador de ponta, e nunca irei esquecer os seus ensinamentos. Muito obrigado. Devo muito a vocês.

Obrigado aos membros do Grupo de Redes da UFRGS. Nós aprendemos juntos muitas coisas, e todos vocês me ajudaram de alguma forma a concluir essa dissertação: Ewerton, Raniery, Clarissa, Giovane, Juliano, Roben, Jeferson, Flávio, Fábio, Alan, Cristiano, e Fabrício.

Por último, gostaria de agradecer minha namorada Cynthia, que provavelmente sofreu muito durante esses dois anos. Porém, superamos muitas coisas como *deadlines* de artigos que eram "intermináveis", viagens para conferências, e o estágio na HP Labs. Mais desafios estão por vir, e tenho a real certeza que temos força suficiente para aguentá-los. De qualquer forma, obrigado pelo seu apoio e imensa compreensão.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

AMN	Activity Modeling Notation
API	Application Programming Interface
BPEL	Business Process Execution Language
CAB	Change Advisory Board
CI	Configuration Item
CIM	Common Information Model
CMDB	Configuration Management Database
COBIT	Control Objectives for Information and related Technologies
DAO	Data Access Object
DB	Data Base
DBMS	Database Management System
DML	Definitive Media Library
DMTF	Distributed Management Task Force
DNS	Domain Name Server
ITIL	Information Technology Infrastructure Library
IT	Information Technology
ITSM	Information Technology Service Management
MTTR	Mean Time To Recover
OASIS	Advancing Open Standards for the Information Society
OGC	Office of Government Commerce
OS	Operating System
RFC	Request for Change
ROC	Recovery-Oriented Computing
SLA	Service Level Agreement
WfMC	Workflow Management Coalition
XML	Extended Markup Language

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The current research on IT change management has been investigating several aspects of this new discipline, but they are usually carried out assuming that changes expressed in Requests for Change (RFC) documents will be successfully executed over the managed IT infrastructure. This assumption, however, is not realistic in actual IT systems because failures during the execution of changes do happen and cannot be ignored. In order to address this issue, we propose a solution where tightly-related change activities are grouped together forming atomic groups of activities. These groups are atomic in the sense that if one activity fails, all other already executed activities of the same group must rollback to move the system backwards to the previous consistent state. The automation of change rollback is especially convenient because it relieves the IT human operator of manually undoing the activities of a change group that has failed. To prove concept and technical feasibility of our proposed solution, we have implemented a prototype system that, using elements of the Business Process Execution Language (BPEL), is able to control how atomic groups of activities must be handled in IT change management systems. Results showed that the rollback solution not only generates complete and correct plans given a set of atomicity marks, but also that the rollback plan generation minimally interferes in the change scheduling process.

Keywords: IT Change Management, rollback plan, change failures, ITIL.

Suporte a Rollback em Sistemas de Gerenciamento de Mudanças em TI

RESUMO

As atuais pesquisas em gerência de mudança em um ambiente de TI (Tecnologia de Informação) têm explorado diferentes aspectos desta nova disciplina, porém normalmente assumindo que as mudanças expressas em documentos de Requisição de Mudanças (RFC – Request for Change) são sempre executadas com sucesso sobre uma determinada infraestrutura de TI. Esse cenário, muitas vezes, pode não refletir a realidade em sistemas de TI, pois falhas durante a execução de mudanças podem ocorrer e não devem ser simplesmente ignoradas. Para abordar esta questão, esta dissertação propõe uma solução onde atividades em um plano de mudança podem ser agrupadas, formando grupos atômicos. Esses grupos são atômicos no sentido de que quando uma atividade falha, todas as outras atividades já executadas do mesmo grupo precisam retroceder para o último estado consistente. Automatizar o processo de *rollback* em mudanças pode ser especialmente conveniente no sentido de que não seja necessário um operador humano desfazer manualmente as atividades que falharam de um determinado grupo atômico. Para avaliar a solução proposta e a sua viabilidade técnica foi implementado um protótipo que, usando elementos da linguagem BPEL (*Business Process Execution Language*), torna-se possível definir como sistemas de gerenciamento de mudanças em TI devem se comportar para capturar e identificar falhas. Os resultados mostram que a solução proposta não somente gera planos completos e corretos com base em marcações de atomicidade, mas também que a geração dos planos de *rollback* interfere minimamente no processo de agendamento de mudanças.

Palavras-chave: Gerenciamento de Mudanças em TI, planos de rollback, falhas em mudanças, ITIL.

1 INTRODUCTION

Currently, modern companies and organizations are often unable to deliver high quality services without employing sophisticated IT infrastructures to support their final businesses. Sophisticated IT infrastructures, in turn, are usually accompanied by complex management challenges that often lead to increasing maintenance costs. A rational management of IT infrastructures then becomes a critical issue for any organization that aims at keeping a good financial health. In order to provide a more systematic IT infrastructure management – and thus reduce management costs – the widely recognized Information Technology Infrastructure Library (ITIL) (ITIL, 2009) presents a set of best practices and processes that helps organizations to properly maintain their IT infrastructures.

Among the ITIL processes, *change management* (IT Infrastructure Library, 2007a) is the one that defines how changes in IT infrastructures should be planned, scheduled, implemented, and assessed. The importance of change management resides in the fact that changes in IT infrastructures must be executed in a way that does not lead the managed systems to unknown or inconsistent states. To address this issue, changes required over the IT infrastructures are firstly expressed in Requests for Change (RFC) documents that define which changes are needed but not how they must be performed. The definition of an RFC is the first step of a process that will generate a final *change plan*, which is essentially a workflow of low level activities that, when executed, will evolve the managed system to a new consistent state according to the changes expressed in the original RFC.

Although change management is a relatively new discipline, several important challenges have already been investigated in research projects (KELLER et al., 2004) (BARTOLINI; SAUVE; TRASTOUR, 2006) (REBOUÇAS et al., 2007). Given the complexity of the subject, these investigations have naturally made some assumptions that enabled the investigations to progress. One of these assumptions is that once an RFC is approved and ready to be deployed, the activities of the associated change plan will always succeed and lead the IT infrastructure to the next consistent state. This assumption does not in fact represent a realistic and practical situation in actual IT environments because failures during the execution of changes do happen and cannot be ignored. If it is necessary to completely avoid failures, a company should never change anything on their systems. However, change is an everyday part of IT. In fact, is a key element of successful IT, and for a successful business as well.

According to a survey (PATTERSON et al., 2002), well-managed servers today achieve 99.9% to 99%, or 8 to 80 hours of downtime per year. Therefore, taking availability as a parameter to measure failures during changes, it shows that each hour of unavailability can cost from $200,000 per hour for an Internet service like Amazon.com, to $6,000,000 per hour for a stock brokerage firm. These high values are a representative sample on how failures in IT changes may impact from the whole business to the financial health of

a given company.

In this thesis we focus our attention to the necessity of handling failures on change plan execution, in order to avoid managed IT infrastructures of ending up in undesired states. To address this problem, we firstly define that, after the deployment of an RFC, the managed infrastructure must have either successfully evolved to a new state, or returned to the state previous to the change request. In other words, RFC deployment is treated as a single atomic transaction. To support this behavior, we have proposed a rollback model to support failures in change plans. Whenever a failure occurs during a change plan execution, a rollback procedure is invoked to undo changes executed so far and abort the ongoing change plan. In fact, we go further in our proposal and observe that for some IT scenarios it would be too restrictive to consider an RFC the only possible atomic element. We thus propose that additional atomic elements can be complementarily defined in a granularity finer than an RFC, for example, at the change plan level (*i.e.* change actions). Therefore, the operator can define which specific actions should have a rollback procedure using the atomic concept.

In order to materialize atomic transactions on IT management in the different levels of a change, we have employed a set of techniques in a prototype system called CHANGELEDGE, developed to evaluate our proposed solution. In particular, we have explored some exception-related mechanisms (*e.g.*, fault handlers, asynchronous notifications, and flow controllers) present in the Business Process Execution Language (BPEL) (OASIS Standards, 2007). In our prototype, atomic transactions, that are defined by human system operators at RFC and change plan levels, are translated into BPEL constructions by a translating algorithm. This algorithm basically converts instantiated classes from our proposed model to a BPEL document using the WS-BPEL standard, turning the model-based change plan into an actual deployable plan. The translating algorithm is important in the process, since it adds BPEL constructions in order to catch failures in the change deployment, then enabling to invoke rollback plans in failure events. However, we do not address, in details, the translating algorithm that takes change plans and transforms it into BPEL documents. We assume that such translating algorithm is somehow trivial, since the proposed model and the WS-BPEL standard are expressed as a workflow. Nevertheless, this thesis discuss all BPEL constructions that are used in order to build mentioned BPEL documents. In addition, the algorithm that generates rollback plans is addressed, showing how rollback actions are organized to undo failures occured in a given change plan. Therefore, a set of experiments has been carried out to observe the impact of our proposal in the whole management system as well as on the managed IT infrastructure. These experiments are focused in scenarios aiming to observe both qualitative and quantitative analysis.

Considering the whole proposal, this thesis presents an end-to-end solution to enable rollback support in IT change management systems. Essentially, the end-to-end solution is composed by all steps contained in the change plan specification process with rollback support: from building a change plan with recovery capabilities to the change deployment scheme invoking rollback plans for execution.

The remainder of this thesis is organized as follows. In Chapter 2 we briefly review some definitions, discuss the rollback support for computing systems, and introduce the ITIL also showing the change management process as well. The proposed solution to incorporate rollback support in change plans is presented in Chapter 3, while the associated prototype is described in Chapter 4. The evaluation carried out in this research is then presented in Chapter 5, that also includes a set of commented results for different

scenarios. Finally, we present the Chapter 6, where conclusions with the contributions are considered, closing this thesis with a section to discuss a set of additional issues in the field of the research.

2 BACKGROUND

In this chapter we present some definitions related to rollback recovery (Section 2.1), the most relevant related work related to this thesis (Section 2.2), and a briefly description about concepts in the field of IT change management with a succinct comparison between standards (Section 2.3).

2.1 Definitions

Since rollback recovery has been studied in various forms and in connection with many fields of research, it is perhaps impossible to provide an extensive coverage of all the issues related to rollback within the scope of this thesis. However, and despite of the word rollback be a widely and well-known used term in the computer science, its definition and different techniques are reviewed for a better reading comprehension of the thesis. In this way, such definitions are explained and commented on how they are applied inside of the context of the research.

According to the American Heritage Dictionary of English Language (AMERICAN HERITAGE DICTIONARY, 1996), the word *rollback* (data management) has the following definitions:

1. A turning back or retreat, as from a previously position or policy.
2. Reverting data in a database to an earlier state, usually in response to an error or aborted operation.
3. In a transaction based database system, transactions are considered atomic. If an error occurs while performing a transaction, the database is automatically rolled back to the state at the previous commit.

Therefore, even then that such definition is applied to data management, it is possible to observe that a general definition to rollback is "to revert something to a previous state". In this thesis, the act of "revert something" may be faced as any process or mechanism that return to a former state of the IT enviroment. For example, if it is performed an action to modify a server configuration, the rollback for such action is to "revert the server configuration" to reach the previous state of the IT infrastructure.

According to Elnozahy *et al.* (ELNOZAHY et al., 1996), rollback recovery techniques can be separated in two primary categories: checkpoint-based rollback recovery, which relies solely on checkpointed states for system state restoration; and log-based rollback recovery, that uses checkpointing and message logging. The first scheme finds a set of mutually consistent checkpoints, one per node, and returns each node to the state saved

in its checkpoint. In the log-based scheme, a process can deterministically recreate its pre-failure state even if this state has not been checkpointed, by logging and replaying the nondeterministic events in their exact original order.

In the context of this thesis, we do not explore in details the internal algorithm of the employed rollback recovery technique. In fact, this research focuses on enabling the rollback support in IT change management systems, which the main processes are to specify atomic elements and generate a consistent rollback plan. However, we mention and explain which recovery techniques are used in the implemented prototype.

The last definition to be addressed in this section is closely related to rollback, and needed to perform the rollback process. In the aforementioned definition of rollback we can observe that transactions are considered "atomic". According to the American Heritage Dictionary of English Language (AMERICAN HERITAGE DICTIONARY, 1996), the word "atomic" is defined as indivisible or immeasurably small. On the other hand, the definition of "atomic transaction" is one which is guaranteed to complete successfully or not at all. For this reason, applying this concept to the rollback in IT change management, atomic change parts may be faced as indivisible. In other words, whether the change part is performed successfully, or the change cannot be persisted over the IT environment.

2.2 Related Work

In recent years, some research has been carried out in the area of rollback-recovery on network/service operations and management community. In parallel, and to have a comparison parameter, we describe relevant aspects based on studies in actual IT scenarios aiming to show the importance of employing a recovery mechanism in IT management systems. Therefore, in this section, we cover some of the most important investigations and scenarios concerning these topics. First, Subsection 2.2.1 presents some researches covering rollback in the different levels, and what is the main gap to be fulfilled regarding such problem. Second, Subsection 2.2.2 briefly describes some surveys made in actual IT companies in order to analyze the treatment of failures in the industry and how failures may impact on their business.

2.2.1 Rollback in the Different Levels

As mentioned before, rollback support is a complex subject in diverse computer science disciplines. Several aspects of computing systems (*e.g.*, faulty underlying communications, dependencies among distributed components, services unavailability) make saving consistent states and subsequent rolling back to them a task that cannot always be properly or successfully accomplished. Nevertheless, some mechanisms to support rollback have already been proposed in research investigations and even market products. In this subsection we review rollback-related work in different levels that has inspired the design of our proposed solution.

At the device level, a common way to implement rollback support is to download a device's configuration file to a configuration server, deploy a new configuration, and use the previous one again if the new configuration turns the device behavior unstable. An evolution of this solution can be seen in devices where candidate configurations are stored inside the managed devices themselves, dispensing with an external configuration server. Recently, the NETCONF protocol (ENNS, 2006), proposed by the Internet Engineering Task Force (IETF), has incorporated the notion of transactions in a configuration task, which prevents the managed devices of evolving to unknown states.

Solely, rollback at the device level is not sufficient for complex IT scenarios because often different devices and services are dependent of one another. For example, if the installation of a new Web server that requires additional configuration of the border firewall fails, not only the server installation itself needs to be undone, but also the configurations on the border firewall must be returned to the previous state. Rollback at the network level (above the device level) is then required. In an earlier work, we have proposed a Policy-Based Network Management (PBNM) system (ALVES et al., 2006) where failures in the deployment of QoS policies return the managed devices to the previous state using an adapted version of the two-phase commit protocol.

Andrzejak *et al.* (ANDRZEJAK; HERMANN; SAHAI, 2005) have investigated automatic workflow generation that can adapt the underlying IT system in reaction to failures. However, this initial work does not present many details about automatic correction of graphs in response to partial failures. In addition, the authors recognize that the proposed solution has some bottlenecks, such as complexity limit related to the number of objects and existing operators, upper bound on the sum of costs, and on specification of the actions.

Candea *et al.* have proposed the Recovery-Oriented Computing project (ROC) (CANDEA et al., 2004) that focuses on the construction of systems with a fast and high failure recovery capability, in spite of building systems that are immune to any kind of error. Among the recovery techniques discussed in the ROC project, the *system-level undo/redo* can be closely observed. This technique is based on a system that controls actions to be undone (or redone), taking into consideration actions previously executed. However, the ROC project techniques do not consider dependencies between managed elements in the network level to perform the recovery process.

Among the current researches, it is possible to verify that no solution consider the failure recovery process in a wider view of the IT infrastructure, specially when changes are deployed. As mentioned before, this fact may result into a problem when changes modify managed elements that have interdependencies (*e.g.*, services, firewalls, load balance policies, and others). Therefore, a solution that consider interdependencies between managed elements in IT infrastructures, building a consistent rollback plan related to the actual IT environment scenario, becomes necessary.

In the change management research field, as far as the authors of this paper are aware of, there is no other research line that addresses the question of employing rollback as a mechanism to maintain the managed IT infrastructure in a consistent state. The importance of rollback in IT change management can be also directly observed in the ITIL documents, where remediation plans are explicitly mentioned as a requirement of change management. Moreover, ITIL defines that an RFC should have remediation plans in order to be approved by the Change Advisory Board (CAB). This fact requires not only remediation plans to be generated, but also that the generation process should be optimized enough for not interfering on the change scheduling. However, even with the ITIL best practices, no proper support is found in the current systems.

2.2.2 Treatment of Failures in Actual IT Scenarios

According to Murphy, anything that can go wrong, will go wrong. That seems to be true for IT environments, and especially true during changes. Thus, many impacts may rise from failures in ongoing production IT infrastructures. However, the most easy perceptible impact is the cost. As an example, the business cost of service downtime may vary from industry to industry, and from one kind of failure to another. For companies

like Amazon, e-Bay, and Google, any failure regarding online transactions represents a potential loss of revenues and reputation – sometimes the latter can be even worse than the former. Even 99.999% in the level of availability may not be sufficient, since this represents a full eight hours of downtime each year. Therefore, especially for companies as mentioned above, the value of availability and the consequences of IT failure can justify almost any cost.

In a study commissioned by Dimension Data (UK) – in partnership with technology vendors InfoVista, OPNET, and EMC[2] – it was taken 200 interviews with companies listed on the FTSE Group[1] (FTSE Group, 2009), in order to provide a real depth of insight into the issues of IT operations management (WHITEPAPER: IT Operations Management Solutions, 2005). From such survey, it was reported that, on average, a company will lose 235 hours per year in different faults, or nearly 14% of a typical man year (1,725 contracted hours). More precisely, it was established that, on average, 5.3 network faults were reported in three months, with an average resolution time of 2.3 hours. Server faults were slightly less common, with the same average resolution time. Finally, application faults were the main cost, taking each one an average of 4.5 hours to be solved. Since it were reported that there are around eight application faults per three-month period, strong impacts may be faced if no back-out plans are previously defined.

Nevertheless – and not being a surprise by talking about big enterprises – all areas, from servers to networks and applications, are monitored for faults by the majority of companies interviewed in the survey. However, only 22% of firms currently use root cause analysis tools to identify the exact point of failure and recover the system in some manner – even if it has simple actions.

Talking specifically about changes, and still based on the mentioned survey, 50% of the CIOS admitted they lacked effective change management processes (also the standards) and tools. In addition, only 8% of all companies reported plans to make an investment in change management solutions in the years to come.

Observing such survey results, despite it was released in the end of 2005, it gives concrete values to conclude that change management systems with recover capabilities tend to be very promising in the near future. This conclusion can be strengthened in a survey commissioned by Hewlett-Packard and conducted by GCR Custom Research in the year of 2007 (HEWLETT-PACKARD NEWS WEBSITE, 2007). Such survey shows that backup and recovery efforts, with an improved IT management, are one of the targets for higher investments in the future years. It means that, comparing both studies in a rude manner, we can note a possible growth in the interest of IT management standards (including change management) and recovery capabilities.

2.3 ITIL and Change Management

The Information Technology Infrastructure Library (ITIL) is a set of concepts and policies for managing information technology (IT) infrastructure, development and operations. It is published in a series of books, each one covering an IT management topic. Basically, the ITIL library is divided by the ITIL Core, which provides best practices applicable to organizations of all sizes and types; and the ITIL Complementary Guidance, which comprises a complementary set of publications with guidance specific to industry sectors, organization types, operating models, and technology architectures (ITIL, 2009).

[1]FTSE is a British provider of stock market indices and associated data services. It is a private company, 50/50 joint venture of Financial Times and London Stock Exchange.

The ITIL Core is composed of five books, which presents best practises and guidelines for service management: *Service Strategy*, *Service Design*, *Service Transition*, *Service Operation*, and *Continual Service Improvement*.

The ITIL Service Transition book relates to the delivery of services required by the business into live/operational use, and often cover the "project" side of IT rather than the whole picture of business (IT Infrastructure Library, 2007a). It includes the management of IT changes, in a subject called Change Management. Here, change means any modification made to the IT environment in different levels. From the high level, as to implement new services due to business requirements, to the finest level of granularity, as to install a new router firmware or configure a new system user.

To accomplish a change, ITIL recommends to follow a set of steps (IT Infrastructure Library, 2007a). These steps are explained in order as follows:

1. **Create RFC:** a *Request for Change* (RFC) is raised by a change initiator or operator, which is an individual who requires the change. As mentioned before (Subsection 2.2.1), one of most critical information recommended by ITIL for a successful and complete RFC creation is the existence of a *back-out* or *remediation plan*.

2. **Record RFC:** the newly requested change is recorded in the Configuration Management Database (CMDB). Some information must be recorded in the change creation process (e.g., change description, reason for the change, contact information of the person who proposed it, back-out or remediation plans), whereas other information are continuously created/updated throughout its lifecycle (e.g. executed actions).

3. **Review RFC:** the RFC is evaluated regarding the completeness, status (e.g., accepted, rejected, under consideration), and technical or financial feasibility. Changes that are not in accordance with the evaluation may be filtered and returned to the change initiator, as well as with a reject report.

4. **Assess and Evaluate Change:** the RFC document is evaluated by the Change Advisory Board (CAB). The CAB is a group of people capable of analyzing and assessing changes from a technical as well as from a business point of view.

5. **Authorize Change:** the CAB actually authorize the change to be deployed.

6. **Plan Updates:** the required actions needed to deploy such RFC are planned, updated, and tested.

7. **Co-ordinate Change Implementation:** the RFC is forwarded to a team who actually deploy the changes over the IT infrastructure.

8. **Review and Close Change Record:** by the end of the deployment, results should be reported for evaluation to the team responsible for managing changes. Incidents or other managed failures need to be explicitly included in the report for a future evaluation.

In the change management process, all change activities shoud be executed in concomitance with the Configuration Management. For that purpose, ITIL recommends the use of Configuration Management Systems (CMS) and Configuration Management

Databases (CMDB), to manage information about all configuration items involved with Service Management processes.

In the context of this thesis, ITIL recommendations are followed related to the Change Management area. However, other promising standards that cover IT management may be observed. The Control Objectives for Information and related Technology (COBIT), as the ITIL, is a set of best practices (framework) for IT management created by the Information Systems Audit and Control Association (ISACA), and the IT Governance Institute (ITGI) in 1992 (ISACA, 2009). Basically, COBIT is focused on the control, while the ITIL concentrates on the processes, the fact which most of times makes COBIT to be perceived as a kind of audit framework. Also, another COBIT characteristic is the use of IT metrics and critical success factors. However, COBIT's books *Control Practices*, *IT Assurance Guide*, *IT Governance Implementation Guide*, and *User's Guide for Service Managers* have grown to offer a credible alternative to ITIL (RIDLEY; YOUNG; CARROLL, 2004).

Despite that COBIT demonstrate to be a promising standard to IT management, ITIL best practises are followed in this thesis mainly due to: a) big companies such HP, IBM, and EDS uses ITIL recommendations over its products; b) ITIL describes processes, and it is important to be based on consolidated and well-known steps when defining recovery procedures.

3 ROLLBACK SOLUTION

In this chapter we present our solution for rollback support in IT change management systems. We first describe a general IT management architecture where the rollback support is introduced. We then discuss how atomic actions take place and how system administrators are able to group critical activities in atomic groups. Finally, we present the modeling of IT infrastructure classes to support the proposed approach and the algorithm used to generate rollback plans.

3.1 IT Change Management Architecture and Rollback Support Components

Although there is not a single, widely employed architecture to support IT change management, it is possible to identify a set of basic functional components that, grouped together, form a general architecture. We introduce in such an architecture complementary elements to explicitly support rollback in change plans. Figure 3.1 depicts the general IT change management architecture highlighting in black the required components for the rollback support.

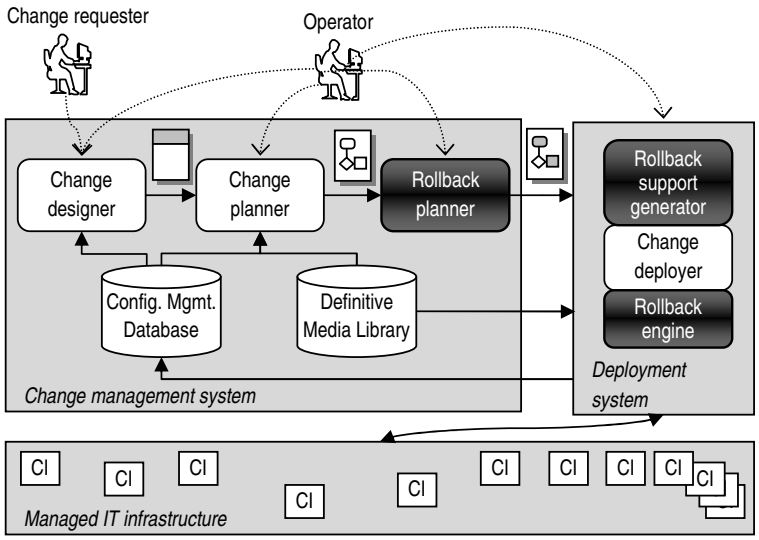

Figure 3.1: IT change management architecture

The specification of a new RFC begins when the *change requester* describes, in a high-level document, his/her change necessities. This is achieved by interacting with the *change designer*, which is a tool that helps the change requester to fulfill the RFC doc-

ument in a clear and consistent way. It is important to remember that an RFC express what is required, but not how to achieve that. This is initially defined when the *operator*, also interacting with the change designer, come up with a preliminary *change plan*. For example, consider the statement "install a new Web-based project management system on server A", which is typical of an RFC. Here, nothing defines whether a new Web server is needed or if an additional database must be created prior to the installation of the Web application; this is, however, informed by the operator that, consulting the *Configuration Management Database* (CMDB) is able to check whether Web servers and required databases are available.

The output of the change designer is then a preliminary change plan that needs to be further complemented. The *change planner* component is the one responsible for automatically computing an actionable workflow that defines the final change plan. The algorithm for such computation has been already addressed by other authors (KELLER et al., 2004) and is the subject of a complementary research of our group (COSTA CORDEIRO et al., 2008). The change planner, in its process of complementing the preliminary change plan, needs to consult two other components: the *Configuration Management Database* (CMDB) and the *Definitive Media Library* (DML). CMDB stores updated information about the managed IT infrastructure, enabling the *change planner* to discover which elements must be manipulated in order to fulfill the original RFC. The *Definitive Media Library*, on its turn, maintains information about software dependencies, which is consulted during the installation/uninstallation process. Actually, DML is a secure place that stores and protects definitive authorized versions of all media for *Configuration Items* (CIs). Therefore, DML does not only store software, but also hardware information and their dependencies. For example, CMDB lists the software already available on server A, but it is the DML that informs, during the installation process, that the Web-based management system requires a pair of Web server and database to work properly.

In a change system with no rollback support, the workflow computed by the change planner would be ready to be submitted, upon the operator's order, to the *deployment system*. It then executes, using off-the-shelf solutions, the changes over the IT infrastructure (according to the change plan). One central point of this thesis is that, in this case, if any fault occurs during the change plan deployment, the system will possibly enter in an inconsistent state, because no reactions to faults are usually defined. In our approach, we address this issue by complementing the change plan with rollback information. That is accomplished through the *rollback planner* component. As can be seen in Figure 3.1, it takes as input an actionable workflow as well as additional marks informed by the operator. These additional marks allow the rollback planner to create an enhanced version of the original change plan, that now includes marks to support rollback actions if fails occur.

Internally, the deployment system is composed of: *rollback support generator*, *change deployer*, and *rollback engine*. The rollback-enabled change plan is submitted to the rollback support generator that creates internal structures to support rollback actions (explained with more details in Section 4.2), while the change deployer performs changes over *Configuration Items* of the IT environment. If any fail occurs in the change process, the *rollback engine* is invoked and executes such rollback plan for the failed activity, following the marks provided in the rollback-enabled change plan.

Whether a change plan has been successfully deployed or a fail triggered a rollback procedure, the *deployment system* must update CMDB at the end of the process. This ensures that the configuration database will provide an updated view of the IT infrastructure,

which is required by the change planner to compute future change plans.

3.2 Marking Rollback-enabled Change Plans

As mentioned before, the operator is responsible, in a rollback-enabled IT change management system, to mark the original change plan in order to complement it with rollback information. To present how these marks are defined, we first need to observe how RFCs and change plans are internally organized.

A single RFC is composed of one or more *operations*. Each operation is an independent element that must be executed to accomplish the change requested in the RFC. Since operations are independent from each other, two different operations of the same RFC can be executed in parallel. Internally to each operation, a single change plan is found, which means that one change plan is associated to each operation. In fact, all change plans from the same RFC could be merged into a single one to optimize the deployment process, but for the sake of simplicity we assume here that RFCs with more than one operation will present one change plan for each operation. Finally, each change plan is composed of a set of activities that are chained to form the final actionable workflow.

In order to support rollback-enabled RFCs, we define that some elements (RFCs, operations, or activities) can be marked as *atomic elements* using *atomicity marks*. The simplest case is the one where the whole RFC is marked as atomic. It means that, if any problem happens during the execution of any of its change plans, all activities must go backward, moving the IT infrastructure to the state previous to the RFC deployment. If no mark is defined at all, any failure will abort all change plans in execution, but no rollback action will be performed, leaving to the system administrator the responsibility of leading the infrastructure back to a consistent state.

Limiting the atomicity marks at the RFC level may be too restrictive in some scenarios. Consider, for example, that one does not want to uninstall a Web server in the event of a failure in the process of installing an associated database. We thus further define that besides marking an RFC as atomic, one can mark each operation of an RFC as individually atomic as well. In this case, only the change plan of a failing operation rolls back, leaving the change plans of other operations untouched.

Finally, atomicity can be defined at the activity level. Here, if an activity defined as atomic fails, it will only rollback itself, but the next subsequent activities will be executed. If a failing activity is not atomic, no rollback is executed, but the associated change plan is aborted at all. An additional mechanism is incorporated at the activity level: we define that a group of activities that are closely related to each other may form an *atomic group*. Once any activity of such a group fails, all other activities in the same group must rollback. Using atomic groups, one can define an atomic operation in two different ways: (a) by marking, at the operation level, the operation as atomic, or (b) by grouping all activities inside of an operation in a single atomic group. The option (a) is obviously easier to choose, but the fact that (a) and (b) are a possible indicative that, in fact, an atomic operation is a particular case of an atomic group composed of all activities from a single change plan.

In order to exemplify the use of atomicity marks, Figure 3.2 depicts how atomicity is defined at the aforementioned three levels, *i.e.*, at RFC, operation, and activity levels. Figure 3.2-a shows an RFC composed of three operations: OP1, OP2, and OP3. In this case, the whole RFC is marked as atomic, implying that any problem on any operation will revert all actions executed. Figure 3.2-b presents the same RFC and operations, but

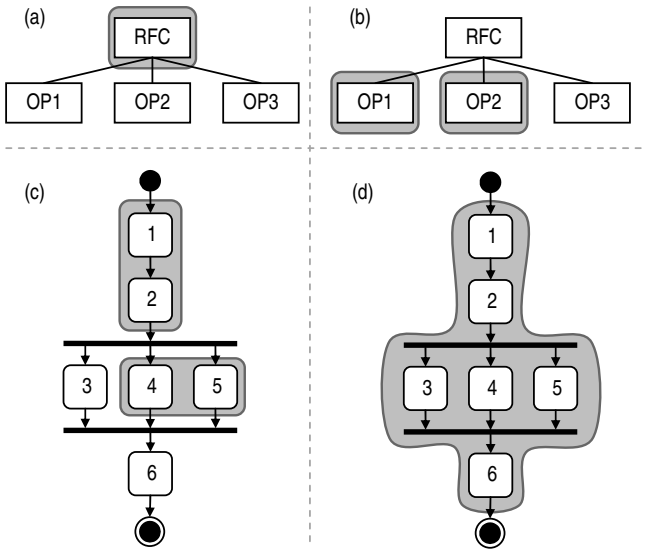

Figure 3.2: Examples of atomicity marks and atomic groups

atomicity is defined differently now. If OP1 fails, its internal actions will be undone, but it will not affect other operations of the RFC. The same happens with OP2. Since OP3 is not marked, any fail in its actions will abort the associated change plan without executing any rollback procedure. Figure 3.2-c presents marks at the activity level. In this case, a change plan composed of activities numbered from 1 to 6 has two atomic groups: one formed by activities 1 and 2, and another formed by activities 4 and 5. Note that, in the first group, the activities are executed sequentially. In this case, if activity 2 fails, activity 2 itself and activity 1 are reversed. After that, the workflow continues evolving to the execution, in parallel, of activities 3, 4, and 5. In the second atomic group, if activity 4 or 5 fail, both will be undone. In this case, activity 3 is not affected at all. Once activity 3 finishes, and activities 4 and 5 rollback or complete successfully, activity 6 will be ready to be executed. In the event of a failure in activity 3, it is important to emphasize that the whole change plan is aborted, skipping the execution of activity 6. Finally, Figure 3.2-d shows the case where the whole change plan is made atomic. The same effect can be achieved by defining that the operation that generated this change plan is atomic, in this case without requiring the definition of an atomic group of all operations' activities.

3.3 Rollback Model

In order to enable change plans to include rollback support, it is necessary to model change plan information with rollback in mind. As mentioned before, a change plan consists in a workflow of activities that when executed lead the managed infrastructure to a new consistent state. Therefore, for our solution, a model for change plan information must thus express actionable workflows that include rollback support. We design our solution thus defining a *Requests for Change and Change Plan Model*. Our model is strongly based on the change management guidelines presented in the ITIL Service Transition book (IT Infrastructure Library, 2007a), and on the approach to specify workflows defined by the Workflow Management Coalition (WfMC) (The Workflow Management Coalition Specification, 2007). It is not our goal here to stress which information is required to fully express workflows. Rather, we are interested in modeling the information

required for rollback support in a change management system. Considering these assumptions, Figure 3.3 presents a partial view of the defined model, highlighting the elements required for the rollback support, while omitting others that are not related to rollback.

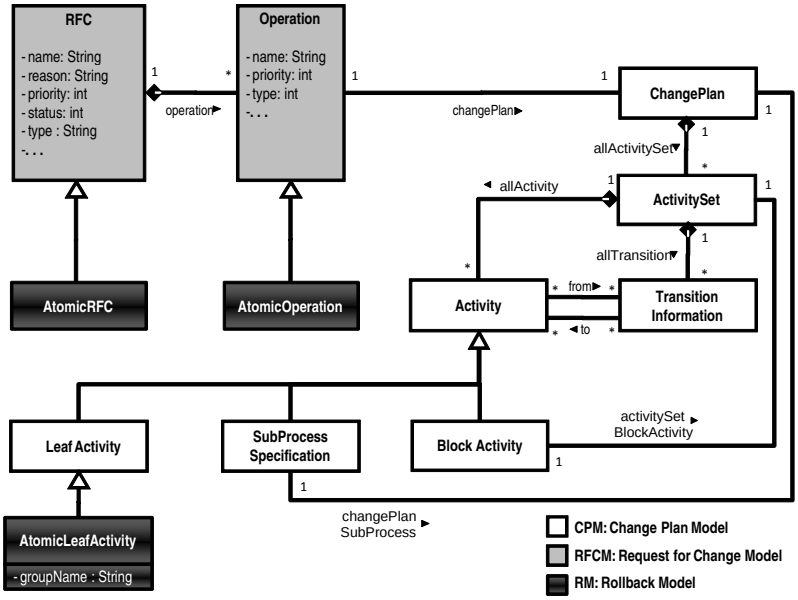

Figure 3.3: RFC and change plan model with rollback support

An `RFC` is composed of `Operations` that, in turn, are composed of one or more `ChangePlans`. `RFC` and `Operation` hold more abstract information of a change request, and thus form the *Requests for Change part* of our model. An `AtomicRFC` is a specialized RFC whose final actions must be treated as a single transaction by the deployment system, *i.e.*, one marks an RFC as atomic by using the `AtomicRFC` class. In the same way, an `AtomicOperation` is an `Operation` whose associated actions will rollback in the event of a problem in their deployment.

Each operation in an RFC has a change plan composed of `ActivitySets`, which are groups of one or more activities intended to implement a change plan. An `Activity` can be either a low-level, non refinable activity (`LeafActivity`), which is the lowest level of granularity that an action can represent, or grouped activities (`SubProcess Definition` and `BlockActivity`), which are activities composed of another activity set or by a new change plan. The `TransitionInformation` class models how activities are chained in the final workflow. The classes `ChangePlan`, `ActivitySet`, `Activity`, `LeafActivity`, `TransitionInformation`, `SubProcess Definition`, and `BlockActivity` form the Change Plan part of our model.

In order to define an atomic activity, one should mark it using the `AtomicLeafActivity` class. Since several atomic activities can be grouped together in atomic groups, each atomic activity needs to provide the identification, in a string, of the atomic group it belongs to. If no name is provided, the activity will belong to a group formed solely by itself. Notice that it is not possible, for a single activity, to be part of more than one atomic group. If that was possible, the activity would work as a mechanism to merge the rollback behavior of those groups. That is so because if one atomic group rolls back, the common activity of both groups should rollback too, leading the activities of the second atomic group to rollback as well.

It is important to note that the design of rollback actions is made by dealing with low-level activities (`LeafActivity` class). As mentioned before, at this level, activities should carry the information of what action should be performed and where. This information is expressed using the Activity Modeling Notation (AMN) proposed by Cordeiro *et al.* (COSTA CORDEIRO et al., 2008), and is filled in as a sentence into the attribute *description* at the `LeafActivity` class. In Some examples that AMN can handle are *uninstall SoftwareElement <X> at ComputerSystem <Y> with <parameters>*, *configure ManagedSystemElement <X> at ComputerSystem <Y> using Setting <Z> with <parameters>*, and *stop Service <X> with <parameters>*. The AMN can easily be extended to express almost any kind of actions performed over an IT infrastructure, just being necessary to define the correct interface between the deployment system and managed elements.

3.4 Producing Actionable Rollback Plan Workflows from Marked Change Plans

In order to produce actionable workflows to deploy an RFC with rollback support, it is necessary an algorithm to generate such rollback plans. In the proposed solution, atomic marks are attached to workflows in order to associate activities to atomic groups. However, plans to undo such previously marked activities must be generated. Therefore, in this section we will present the proposed algorithm that takes as input a marked change plan, and generate as output a set of rollback plans.

In order to explain the rollback plan generation process, we define an example that is used until the end of this section. After the process of marking change plans with atomic information, it is possible to assume that the final change plan result is shown in Figure 3.4-a, *i.e.*, a workflow having eight activities and two atomic groups named AG1 and AG2.

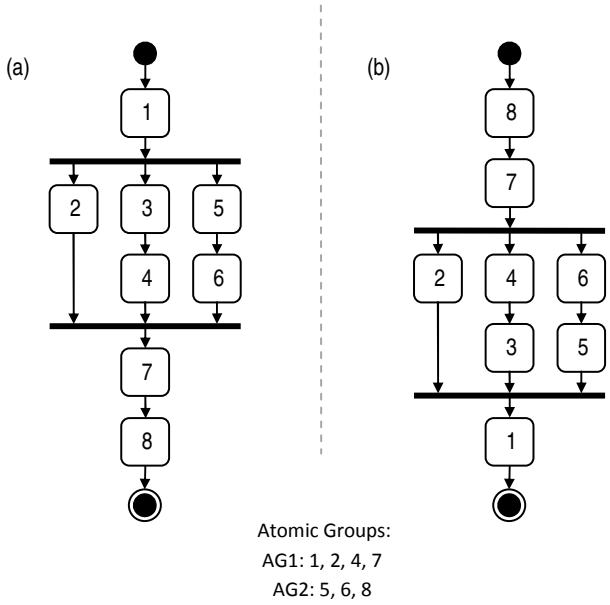

Figure 3.4: Change plan workflow example with atomic groups and the reversed form

The first step to generate rollback plans is to reverse the change plan workflow. The

reversion process is an important step towards the generation of rollback plans, since it preserves workflow transitions and the dependency order. It means that if activity 8 fails (Figure 3.4-a), the generated rollback plan will first undo activity 8, following the rollback of other activities respecting each dependency relationship. The reversion process is made by using the information contained in `TransitionInformation` class (presented in Section 3.3). In fact, the `TransitionInformation` represents transitions between activities, including attributes as *from* and *to*, which are, respectively, source and destination. In this way, to reverse the whole change plan workflow, the algorithm needs to change the order of the mentioned attributes. Figure 3.4-b shows the result of the reversion process following the introduced change plan example (Figure 3.4-a).

The identification of existing atomic groups is made in the reversion process. In fact, the reversion process walks through the workflow, visiting all activities and building a new reversed change plan. Therefore, at this point it is possible to retrieve all the atomic information to save processing. Such atomic information is retrieved from the classes named `AtomicRFC`, `AtomicOperation`, and `AtomicLeafActivity`. During the reversion process, the algorithm will only collect data from *groupName* attribute (`AtomicLeafActivity`), but just if such activities do not belong to any `AtomicRFC` or `AtomicOperation` class. That occurs because there are different ways to express atomicity in higher levels (*i.e.*, RFC and operation levels). For example, if an operation is marked as atomic, all activities will logically belong to the same atomic group, being unnecessary to walk through it. On the other hand, the algorithm will collect data from *groupName* if such activities do not belong to any atomic operation or atomic RFC.

The second step to generate rollback plans consists in analyzing the existing atomic groups. The analysis is made by recursively removing activities following a set of premises. To achieve that, the algorithm takes each activity that belongs to an atomic group and, following such premises, builds a new workflow which is, in fact, the rollback plan. It is important to note that a different rollback plan is generated for each activity, using a reversed workflow copy to the algorithm execution. Thus, the algorithm removes activities that match the following premises:

- Activities that do not belong to the same atomic group;
- Activities that do not have atomic group, and consequently are represented by the `LeafActivity` class – and not by `AtomicLeafActivity`;
- Activities that belong to the same atomic group, but succeed the activity which is taken to generate the rollback plan.

The first premise discards the execution of any rollback action related to other atomic group. In other words, the first premise has the function to maintain the atomic group concept: if a failure occurs in any atomic group participant, only activities that belong to that group will have rollback actions. The second premise discards any activity that was not marked as atomic in the change plan. As a consequence, such activities will not have any rollback actions, and being unprotected from any type of failure. Finally, the last premise guarantees that activities which have not been performed in the change plan, will not be affected by rollback actions triggered by any rollback plan. For example, if the activity 4 in the change plan workflow fails (Figure 3.4-a), even belonging to the same atomic group AG1, the algorithm will generate rollback actions to activities 4, 2, and 1, not adding rollback actions to the activity 7, which has not been executed yet. Figure 3.5 shows rollback plans for activities 7 (Figure 3.5-a), 8 (Figure 3.5-b), and 4 (Figure 3.5-c).

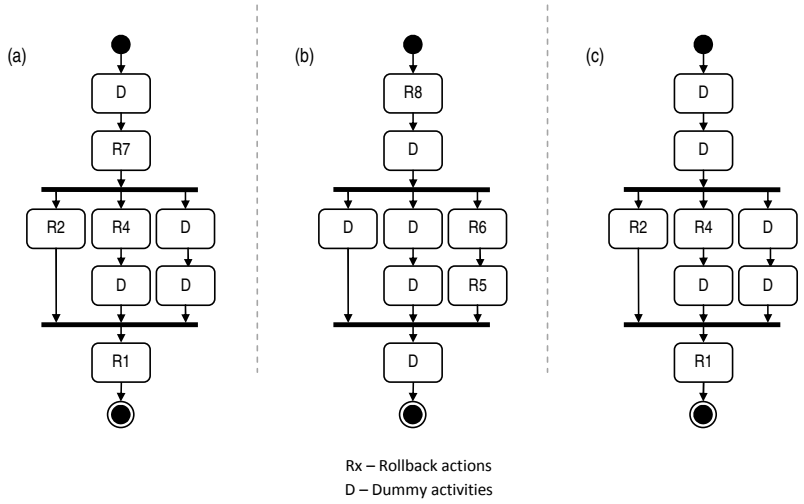

Figure 3.5: Rollback plan workflows for activities 7, 8 and 4, respectively

It is possible to note that discarded activities appear as “dummy activities" in the generated rollback plan workflows. A dummy activity is characterized by the lack of any kind of action, despite of having the representative activity structure in the rollback plan workflow. This approach has been used due to the high processing in rebuilding transitions, if an activity is completely removed from the workflow. In other words, supposing the use of a complete activity removal, the rollback plan shown in Figure 3.5-a will be the following: R7 with a split transition (parallel) to R2 and R4, and then a join transition from R2 and R4 to R1. In order to generate the cited workflow without dummy activities (removing activitities completely), the algorithm would need to iterate again over each transition, finding out which activities are in parallel or in sequence. In this way, the algorithm would need to rebuild the workflow, wasting processing time. Therefore, the use of dummy activities recycles the change plan workflow structure, leading the algorithm to walk through the workflow only once to match the presented premises.

Finally, it is important to note that the second step (analyzing atomic groups) is executed over a reversed workflow copy for each activity belonging to an atomic group. It means that, for each atomic group participant, the algorithm will generate a new reversed workflow copy, and consequently produce a rollback plan for that activity. However, rollback plans can be generated in an identical manner for two or more activities, depending on where the activity is located in the change plan workflow. An example to demonstrate the occurrence of this fact can be observed in the rollback plan for the activity 4 (Figure 3.5-c), that is identical to the activity 2 rollback plan. Detecting the occurrence of identical rollback plans, for a possible optimization, is a subject to be addressed in future work.

4 PROTOTYPE IMPLEMENTATION

In order to prove concept, we have developed a prototype system called CHANGELEDGE that implements our rollback proposed solution. Our implementation is based on Web services technologies and standards, mainly due to the following reasons:

- Web services is a widely accepted solution for interprocess communication over the Internet;
- Several organizations in the industry use Web services for internal communication, and have been increasingly adopting them for the communication between services with other different organizations;
- Network and service management can be implemented using Web services, and some important standards to support that (*e.g.*, WS-Notification, WS-Management, etc.) seems to have a good approval by both academia and industry;
- Standards for Web services composition, such as the BPEL, are gaining popularity in the industry as well, and can thus be used to coordinate distributed actions over an IT infrastructure.

In this context, our implementation is based on the following Web services solutions:

- At the final *Configuration Items* side, we assume that target elements that need to be managed (*e.g.*, hosts, servers, clusters, storage) implement a management interface as a Web service. It can, for example, be materialized following the Configuration Description, Deployment, and Lifecycle Management (CDDLM) specification (The GGF CDDLM working group, 2008). In our assumption, such management interface needs to be able not only to perform actions over the CIs, but also to detect eventual errors. Once such management interface detects any failure during the deployment, it should generate an exception through the Web service for whom invoked the execution of the action. Associated with the Web service management interface, we assume that a Web Service Description Language (WSDL) (W3C Note, 2009) document is also available, describing the management interface itself;
- In order to deploy the required changes over an IT infrastructure, actionable workflows are described in BPEL documents that can be read and executed by a BPEL engine, such as ActiveBPEL (Active Endpoints, 2009). The BPEL engine operates as the deployment system of the previously presented architecture. The communications with the BPEL engine are accomplished using Web services as well;

- The change management system is implemented as a simple Web application accessed by both change requester and operator, and works as the Web service client of the deployment system.

In addition, the prototype interface uses Flash technologies that enable an easy interaction for building and marking change plans, for example. The interface was developed in Flex[1], and is supported by the Apache/Tomcat. As a consequence, for each interaction with the Web interface, Tomcat invokes a service through Java, the language employed to implement the system's kernel. The interface allows defining atomic marks in the three RFC levels, also having the possibility to view the generated rollback plans. The use of the prototype with the interface is presented in the first evaluation case study scenario (Subsection 5.1.1), where the scenario is constructed using the implemented system.

The following subsections present BPEL constructions used in the rollback plans that are expressed in BPEL documents, and how the deployment system is internally implemented.

4.1 BPEL Constructions to Support Rollback

While executing a change plan, the deployment system must keep track of the organization in which actions will be performed. To orchestrate the system such as expressed in a given workflow, four BPEL constructions have been used: `sequence`, `flow`, `if`, and `links`. The `sequence` is a BPEL construction which arranges and executes all of its enclosed activities in an ordered list. `Flow` is a construction that executes all activities inside it in parallel. It means that two or more activities can be defined to start at the same time. `Flows` are complete only when all the activities in the constructions have also completed. They can be used in two situations: to explore parallelism by executing multiple activities simultaneously in one machine, or to invoke actions (without dependencies) distributed in different machines. The `if` construction, as the name suggests, implements the conditional branch, which is common in most programming languages. Finally, `links` is a basic BPEL construction to create links between activities, representing transitions in a workflow. BPEL *link* construction is the main controller of the executing order of the actionable workflow.

The BPEL `invoke` activity has been used to represent a workflow activity, allowing to call a remote Web service to perform the given task. It can thus be used to perform remote operations using two-way (request-response) or one-way messages. In a one-way communication, the invoker sends a message and does not wait for any response. It can, however, be used in a two-way fashion if an explicitly provided `receive` activity is placed to retrieve the result of a remote call, thus implementing an asynchronous communication. In other words, the `receive` activity waits for an input (remote webservice call) just after that `invoke` has called a Web service. In the request-response communication, a message is sent by the BPEL engine and the processing of the workflow is blocked until a response arrives. In this case, the communication is synchronous and timeout-related issues must be taken into account. Our prototype supports both synchronous and asynchronous communications combining `sequence`, `flow`, `if` and `links` to guide the execution of workflows.

Finally, in order to detect configuration problems and thus trigger rollback actions, additional BPEL constructions have been employed. For example, an `invoke` activity

[1] Adobe Flex version 2.0. More information is available at http://www.adobe.com/products/flex/.

can include fault handlers to deal with errors associated with the invoked service. In this case, the `invoke` activity must be added to a *scope* and errors can be caught by a `catch all` construction at execution time, deviating the normal execution flow to a different one that handles the failure. Since we assume in the solution that rollback actions do not fail, no additional fault handlers are attached to a *catch all* construction.

4.2 Deployment System

The deployment system is implemented in Java and organized internally in three blocks already introduced in Figure 3.1: the rollback support generator, the change deployer, and the rollback engine. The complete diagram for the deployment system is presented in Figure 4.1.

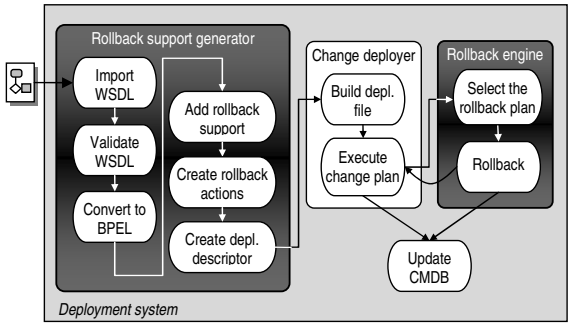

Figure 4.1: Deployment system

First, the rollback support generator receives a marked change plan, and after reading the internal information, imports the set of WSDL files from all the endpoints that will be affected by the change plan. The WSDL files are then validated to guarantee that all required resources and operations are available in the managed elements. In this step, a verification is also done in order to determine what kind of communication will be used to perform each activity (synchronous or asynchronous).

Next, the workflow translation is used to convert the original marked workflow into a BPEL workflow. In our implementation, the change plan file is already a BPEL document that contains no rollback support but only atomicity marks. In order to transform these marks in BPEL rollback structures, the *add rollback support* component is issued. In fact, the *add rollback support* component is where the rollback algorithm is implemented (Section 3.4), producing not only the rollback-enabled change plan that is able to catch activity failures, but also rollback plans for each activity.

Finally, with all information ready to be delivered for execution, the *create deployment descriptor* component creates a complementary file called Process Deployment Descriptor (PDD) required by the ActiveBPEL engine (which we also used in our implementation) to execute a whole actionable workflow. Therefore, the complete set of files is then forwarded to the change deployer. Just to mention, the ActiveBPEL engine is a robust runtime environment capable of executing process definitions created for the Business Process Execution Language (BPEL) standard. Besides that, it is an open source engine, allowing to extend its functionalities if needed.

The *change deployer* expands the BPEL engine functionality by adapting its resources to the deployment system intention. Primary, it builds a file packing of all previously

generated files (WSDL, BPEL, and PDD), called Business Process Archive File, in order to turn the change plan fully available for execution. Finally, ActiveBPEL engine is called to execute the change plan effectively. In fact, the ActiveBPEL engine is responsible for executing the change plan and handling the faults. Once a fail is detected, the normal flow of the change plan is stopped and the *rollback engine* is called. The first thing is to select the correct rollback plan, depending of the failure source. After that, the *rollback engine* actually performs all rollback actions that belong to the selected rollback plan. In other words, if a failure occurs in the *install apache* activity, for example, the *change deployer* will handle the failure using already mentioned *catch all* BPEL construction, and will invoke the *rollback engine*. Afterwards, the *rollback engine* performs all rollback actions accordingly to the fail source by selecting the generated rollback plan. Therefore, the *rollback engine* is able to correctly perform the rollback related to the *install apache* activity failure.

In this thesis we assume that such rollback actions are operations that undo the failed activities. For example: if an original activity was an `install` instruction, the reverse of it will be an `uninstall` instruction. In addition, assuming that the configurable items interfaces are implemented in Web services, we also assume that for each remote action there will be another action able to undo the first one. If that does not happen, however, the rollback procedure itself may lead the managed system to another unpredicted state. It is thus crucial that not only the rollback support works properly, but that the endpoints present reversible operations as well. These reversible operations are aligned to Recovery-Oriented Computing (ROC), where the undo/redo is one proposed recovery technique that is able to heal a high percentage of failures (CANDEA et al., 2004). Figure 4.2 shows that such technique is the closest to the manual repair. However, the ad hoc manner may be not an opportune solution when the time of recovery is considered, and as a consequence being a costly solution. The system undo/redo is also costly (in terms of operations), but less than ad hoc manual technique, and with the advantage to be automated.

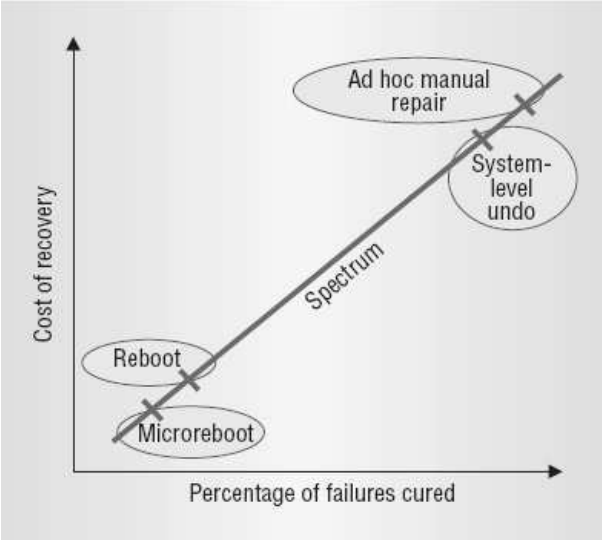

Figure 4.2: Cost of recovery X Percentage of failures cured, according to ROC

As a matter of fact, and being more generic, the proposed solution uses a kind of log-based recovery scheme as a discussed technique in the Section 2.1. The main characteristic of such technique is that a process can recreate its pre-failure state even if this

state has not been checkpointed, by logging and replaying the nondeterministic events in their exact original order. Such characteristic reflect in part the CHANGELEDGE implementation and the whole solution as well. The deployment system executes a plan where activities may be performed over different nodes, or in the context of this thesis, configuration items (CIs). If a fail occur, the deployment system detects such failure and, following a rollback plan, replay such actions in the reverse order respecting the exact original workflow. The difference is that the deployment system will not reach the pre-failure state by replaying activities occured in past changes. The system will return to the pre-failure state by replaying the activities of the current change plan, undoing what was done over the IT infrastructure.

In actual IT scenarios, a variety of operations/changes over IT environments may keep some side effects during the recovery process of systems. This side effects are, for example, configuration files that may remain after an uninstall, other services that may be affected if the DNS daemon is stopped, or entries that are created in the system registry during an install process. However, since this research assumes the existence of the CMDB, which stores updated information about the IT infrastructure, it is possible to map exactly what is modified and by which action some managed element may be modified. This assumption is completely aligned to ITIL recommendations (as well as the whole solution proposed), and turns possible the construction of consistent rollback plans respecting dependencies and also considering effects.

After performing a rollback, the execution flow may return to the original change plan, or then evolve directly to its end, depending on how the atomic activities and groups have been defined by the system operator. The last action, as already mentioned, is to update CMDB (configuration management database) to reflect the new state of the managed IT infrastructure.

5 EXPERIMENTAL EVALUATION & ANALYSIS

To evaluate our proposal, we have conducted two experimental deployments considering real-life scenarios. The first scenario (Section 5.1) is characterized as a more qualitative evaluation, involving a high-level environment and company's strategy. In addition, we show the RFC creation process expressing atomic elements using the CHANGELEDGE interface. The second scenario (Section 5.2) aims to present a more quantitative evaluation, in order to analyze the algorithm that produces rollback plans.

5.1 Case Study #1: Improve the Emergency Load Threshold and Company's Resources

The case study is based on a telecommunication company that provides some management services to its customers using the Internet. To illustrate, these services are basically bill payments, access to the voice mailbox, send free online text messages to mobile phones, and others. Therefore, this case is focused on how the rollback solution is applied, taking into account the company's business strategy. As follows, Subsection 5.1.1 depicts the study case scenario while in Subsection 5.1.2 some analyses regarding the change deployment are presented.

5.1.1 Scenario

In order to cope with the heavy demands of the provided services, a given company employs a high-performance cluster composed of 10 nodes. Each node is equipped with a Dual Core Xeon processor and 2GB of RAM memory. In addition to the cluster, an authentication server responsible for customer's authentication is available as well. Finally, an HP 9000 server, configured using optimal performance options, hosts a MySQL database used to persist the information manipulated by the provided services. This IT environment is responsible for handling users' request, always respecting the Emergency Load Threshold (ELT). The ELT is an appropriated performance threshold that describes the difference between the maximum system load capability and the system load level in rush times. For the mentioned infrastructure, the ELT was calculated in 50%. This difference may appear too high, but since the whole IT environment is a big part of their business, abrupt variations must be taken into account not to compromise the entire system. The ELT is monthly calculated and measured considering the average number of customers' accesses, and typically varies when the company releases a new service. Once the ELT decrease, the availability of the provided services may be severely compromised.

As soon as the company intends to release a new service, an increase in the access load is expected. The company quantify and qualify such increase through a poll estimative.

It was verified that, with such new service, the ELT will significantly decrease to 30% – which breaks the company ELT policy. The company policy says that the ELT value must not fall below 45%. In order to support the load increase and maintain the system health, the company decides to upgrade its IT infrastructure, and consequentially have an ELT increase from 30% to 55%, having a comfort edge with additional 10%. It is important to observe that the IT infrastructure was improved reflecting a strategic action to release a new service, since the ELT could not remain in the same old value. Considering this IT upgrade strategy, the previous decisions are materialized in the IT infrastructure accordingly to the RFC document presented in Figure 5.1. Using the CHANGELEDGE prototype, such RFC is created through the user interface that is shown in Figure 5.2-a.

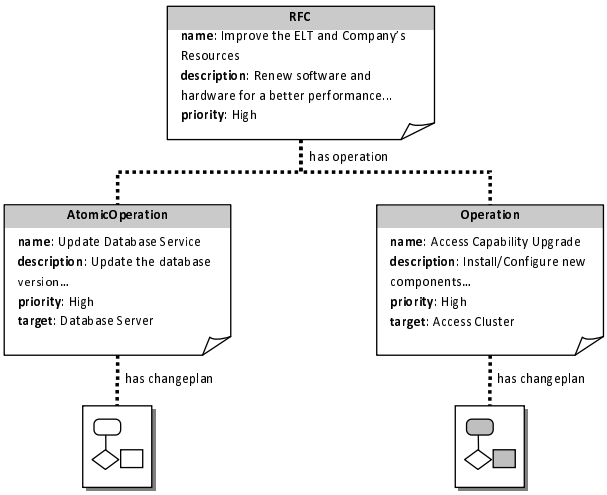

Figure 5.1: Case study RFC showed in a general and hierarchical view

In this RFC, two operations are defined: (a) a database service update, and (b) an access capability upgrade. The first operation intends to improve the database reliability by updating MySQL 4.1 to the Enterprise Edition (EE). The second operation installs a new machine in the access cluster, and tunes the configuration of older machines in order to increase their performance. After being created by the CHANGELEDGE (Figure 5.2-b and Figure 5.2-d) and processed by the *change planner*, the operations generate the change plans depicted in Figure 5.3.

Considering the first operation as a critical one, the change requester marks this operation as an atomic operation. He creates and marks the operation using the CHANGELEDGE interface as showed in Figure 5.2-b. The second operation, however, is not marked (Figure 5.2-d); in fact, the change manager delegates that to the operator, which must define, among the internal operation's activities, those that must be grouped in atomic groups. The operator, accessing the interface using the *operator role*, will mark activities as shown in Figure 5.2-e which is part of the CHANGELEDGE workflow editor. Also it is important to note that, in the first operation, since it is marked as atomic, the CHANGELEDGE disabled the possibility of creating non-atomic activities, as shown in Figure 5.2-c.

The change plan for the second operation is composed of the installation of a new machine with better hardware resources, and performance improvements on each old machine by the installation of additional memory. The first action of the change plan is to physically install and configure the hardware for the new access cluster, including a RAM installation for each old machine. This action is performed by humans, and we assume, in

Figure 5.2: Creating the case study RFC using the CHANGELEDGE interface

this example, that no fail occurs. In this case, such activity can increase the ELT in 5% if it succeed. As the change plan evolves, it executes the software installation/configuration in parallel. The technical conditions are an important *red*factor to define which activities will participate in the atomic groups, which are set in our case study in a way to prevent the system of being unavailable.

Considering this scenario, the operator evidenced that to guarantee the access cluster availability and the new user access demands, the process of marking the change plan with atomic elements must consider the following: the activity related to the new machine must succeed (install and configure software), *and* the activities related to the old machines (*i.e.*, a configuration in performance options) may either succeed *or* fail. In other words, the new machine must be successfully deployed to minimally ensure the access cluster availability and the new release of the service, even having a failure in the performance options related to old machines' activities (independently). The result of the performance options configuration (old machines) is not critical, because even with a failure, the access cluster will remain available – whether or not a rollback action is performed. In this case, the performance configuration will lead to an increase of 5% in ELT, which is not solely sufficient to the new services release, but it does not affect the current company's services in case of failure. However, if all activities of both cases succeed, the release of the new service is ensured. These evidences were taken by consulting the company's IT Service

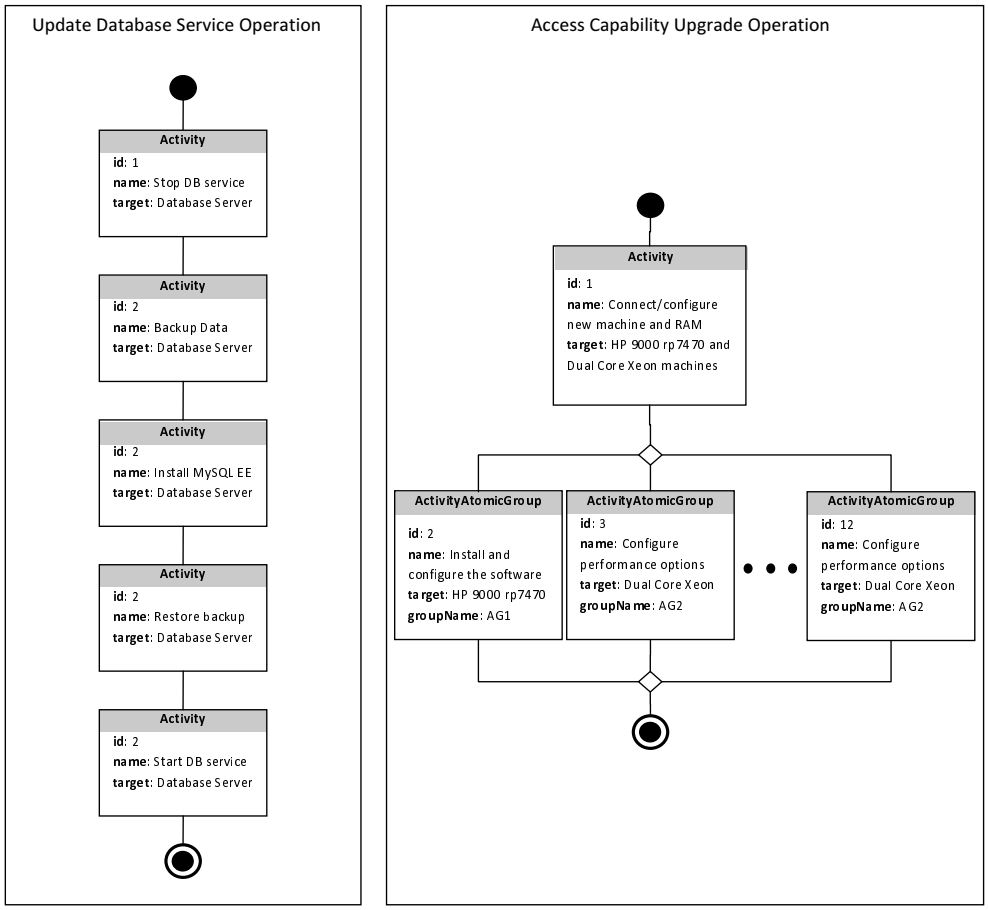

Figure 5.3: Parallel change plans

Continuity plans produced by the IT Service Continuity Management (ITSCM). One of the ITSCM main objectives is to provide advice and guidance to all other areas of the business and IT on all continuity and recovery-related issues (IT Infrastructure Library, 2007b). Therefore, to justify operator's choices of marking the change plan with atomicity information, some conditions are described as follows:

- If one activity related to an old machine fails, the system must rollback all other old machines to the previous consistent and working configuration. In this case, one machine configured differently from others may turn customer's services unavailable. Performing the rollback just for old machines, in case of one fail, guarantees at least the cluster functionality with a better memory specification (due to the activity with Id number 1 in Figure 5.3). In other words, having a independent view of this case, the ELT will be maximally increased in 5%;

- If the new machine installation/configuration fails, the system must identify and rollback its actions. In this case, the change plan will not be interrupted and the access cluster will work normally as before. Therefore, the confort ELT edge of 55% for customer's access demand, which was accorded to the new service release, cannot be assured. On the other hand, the success of this activity guarantees the company's new service release, since the new machine configuration can elevate ELT to 45% (additional 10%).

For these reasons, expressing consistent atomic marks in the activity level can guarantee at least the access cluster availability. In this case, the single activity related to the new machine are set as participant of “Atomic Group 1" (AG1), and all activities related to old machines, in turn, are set as “Atomic Group 2" (AG2).

5.1.2 Analysis

In the first operation performed by the change deployer, which is a database service update (Figure 5.1), we suppose a failure in the MySQL Enterprise Edition installation. Therefore, the change deployer invokes the rollback engine, immediately executing the following actions:

1. Request the database server to rollback the installation: wipe out all installation files related to the MySQL Enterprise Edition, resulting in the previous untouched MySQL server;
2. Request the database server to undo the backup process;
3. Request the database server to start MySQL again.

Note that the *backup data* activity (Figure 5.1) does not affect the system for the rollback procedure, meaning that whether undoing the backup data or keeping it will make no difference.

The second change plan operation, which has the goal of increasing the access cluster capability, has activities marked in atomic groups as presented in Figure 5.1. Supposing a situation where the configuration of the first old machine fails (activity with *id* number 3), the change deployer identifies the failure inside the BPEL constructs, and invokes the rollback engine, then ordering the atomic group to rollback by remotely invoking rollback operations in all 10 old machines. The rollback requests happen in parallel, since the change plan activities were also originally expressed in parallel.

Supposing that both the first old machine and the new machine installation/configuration fail, the change plan will not succeed at all and the final result will not be sufficient to allow the release of the new service. Despite of that, the failing requested change does not lead to the unavailability of all other services previously available because the access cluster is still active. The same case occurs if the new machine installation/configuration fails, but the old machines performance configuration succeeds: the access cluster is still active, but ELT was not increased to release the new service. Finally, supposing that the new machine installation/configuration succeeds, the result of the performance options configuration at old machines will not affect the new service release, since it can increase ELT in 45%.

5.2 Case Study #2: Installing a project management web-based application

In order to look into a more quantitative evaluation, we have constructed a real-life scenario focusing in a low-level view for a better comprehension. The quantitative evaluation consists in the measurement of rollback plan generation times. Such measurement allows analyzing the feasibility of the proposed solution to generate rollback plans for emergency changes, as an example. Moreover, it is important to observe that the comparison related to the quantitative evaluation is made among two cases using our proposed

algorithm. This fact occurs since no other existing solution is available to construct a comparison table with times and atomic elements. Therefore, Subsection 5.2.1 presents the RFC environment, while in Subsection 5.2.2 presents times related to the rollback plan generation algorithm with the proper analysis.

5.2.1 Scenario

As a result to administrative requirements in a company, a change requester describes an RFC with the following description: install the *dotproject* application into a new dedicated server. Among these administrative requirements we can highlight: the project management team discontentment regarding the lack of several special features in the current project management platform, and the high cost for maintaining a non open source software that does not have all features needed. Therefore, it was decided to test a new project management platform called *dotProject*, which has an open source license and can be easily extended to company's needs. Such RFC has one operation, and the generated change plan has 16 low-level activities. The change plan workflow is shown in Figure 5.4, and is generated by the change planner component respecting dependencies inquired at the Definitive Media Library. Also, the described change plan contains marked activities and its atomic groups, defined by the operator in the *rollback planner* component. Since the main objective is to analyze the algorithm, the current case study will focus on how plans are composed and less on company's business strategy or constraints. In other words, we are not concerned about operator reasons to define such atomic groups, or about the importance of each activity in the change plan workflow. In fact, the concept of atomic groups can be used in any way, being the operator the responsible for unfavorable results regarding the misuse of rollback techniques. However, it is important to highlight that ITSCM can guide the operator in such decision, as mentioned in Section 5.1.1.

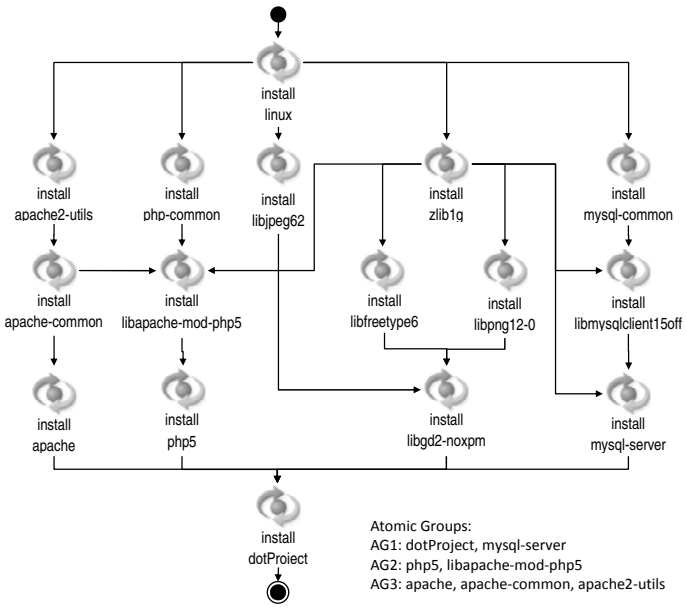

Figure 5.4: Change plan workflow with marked atomic groups

In this case study scenario, the operator defines three atomic groups that are covered by rollback in any failure event. However, activities as *install linux*, *install mysql-common* and others are susceptible to lead the infrastructure to an inconsistent state. The proposed

algorithm will generate 7 rollback plans – for all atomic group participants.

5.2.2 Analysis

Before the RFC deployment, the *rollback support generator* component needs to analyze such atomicity information in order to generate rollback plans. After the aforementioned algorithmic process, all rollback plans are ready in the *rollback engine* component, waiting for any failure event. As an example, we can highlight the generated rollback plan for the *install dotProject* activity shown in Figure 5.5. We can observe that dummy structures (represented by brackets) are taking the most part of the workflow. However, the rollback algorithm maintained the workflow structure, just removing actions off. In other words, the algorithm just removed the semantic action from those activities, preventing, for example, to perform an uninstallation that do not participate from the same group of the *install dotProject* activity.

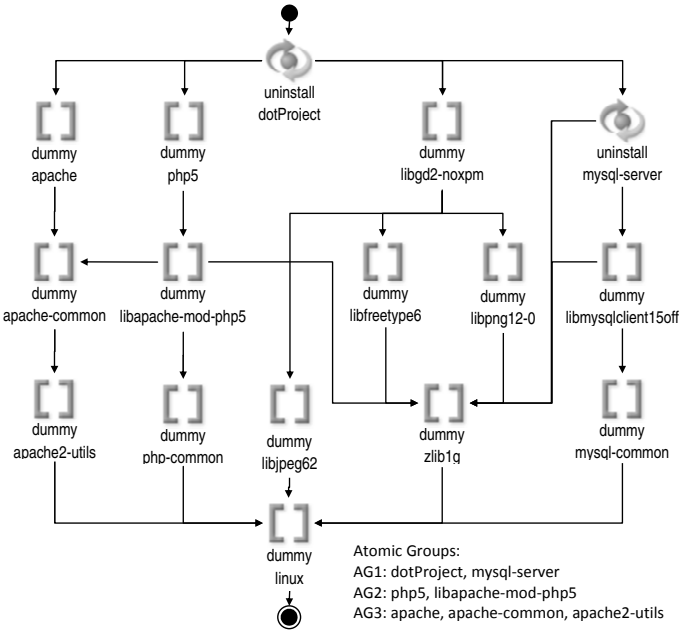

Figure 5.5: Rollback plan workflow for the *install dotProject* activity

Another critical element of this process is how long the algorithm takes to generate all rollback plans for a given RFC. Even though this thesis is not focused on optimizations, an optimized rollback plan algorithm is extremely important due to two aspects. First, the Change Advisory Board (CAB), which is encharged to approve RFCs, cannot authorize changes expressed in RFCs that do not have associated remediation plans (IT Infrastructure Library, 2007a). Second, but assuming that the first aspect is solved, RFCs can be scheduled immediately (as in an emergency change). As a result, both cases require an optimized rollback plan generation algorithm to minimize critical delays. This subject is also discussed in the Subsection 6.2.

To analyze these two aspects, we have executed two cases (using the proposed algorithm) based on the workflow described in the current case study (Figure 5.4). In fact, we just created one additional case scenario aiming to have comparison parameters. In the described case study change plan, the operator expressed atomicity information in the activity level, creating three distinct atomic groups. On the other hand, we just created a

parallel case to analyze how much time the rollback plan generation process can raise by expressing the whole RFC as atomic. Therefore, having executions with the same RFC (and thus with the same change plan), but with different atomicity marks, it is possible to compare time results. Below is shown both cases description:

- Atomic Groups case: it is the same change plan shown in Figure 5.4 with its atomic groups – unchanged;
- Atomic RFC case: the whole RFC marked as atomic in spite of marking only some activities. It means that all change plan activities are participating to the same atomic group – a particular case of marking the change plan in the activity level, as explained in Section 3.2. In this case, all sixteen activities shown in Figure 5.4 will have a rollback plan associated.

For each case, the algorithm was executed 30 times generating all rollback plans accordingly to its atomicity specification. Furthermore, it was calculated a confidence interval of 95% related to all repetitions. Table 5.2.2 shows the results for the described cases.

Table 5.1: Rollback plan generation results. Times represented in milliseconds.

		Cases	
		Atomic RFC	Atomic Groups
Number of activities		16	16
Number of atomic activities		16	7
Time Average		205.73	206.10
Standard deviation		2.03	7.48
Lowest generation time		204	203
Highest generation time		213	245
Total generation time		11923.53	1442.70
Confidence Interval	Lower bound	201.67	191.13
	Upper bound	209.79	221.07

The results evidenced that considering the original case study scenario (Atomic Groups case showed in Table 5.2.2), the RFC could be ready to the scheduling process – and consequently the deployment – with approximately 2 seconds of delay. This fact is observed by looking to the total rollback plan generation time, meaning that after the operator finishes its interaction, the system produces in 2 seconds an RFC ready to be approved by CAB and, afterwards, properly scheduled for deployment. Representing the worst case, when the whole RFC is marked as atomic (Atomic RFC case), the RFC could be scheduled with approximately 12 seconds of delay. The whole RFC marked as atomic is considered as being the worst case because the algorithm must generate rollback plans for all activities. However, the mentioned delays can be slightly higher, since that other factors like BPEL documents deployment and network communication must be taken into account.

We can also observe that in the Atomic Groups case the standard deviation value is considerable higher compared to the Atomic RFC case. The main cause to this fact is the highest generation time (245 milliseconds) that was performed to generate the rollback plan for the *install dotProject* activity. However, the highest generation time value

reached in the Atomic RFC case (213 milliseconds) also was generated by the same activity. Actually, the main difference consists in transforming activities in dummy. To generate the rollback plan for the *install dotProject* in the Atomic RFC case, the algorithm does not need to transform any activities to dummy: if any activity fails, all other activities must have rollback actions. On the other hand, within the Atomic Groups case the algorithm needs to walk through the workflow and match the premises presented in Section 3.4, taking more time to generate such rollback plan. Therefore, rollback plan generation times for cases where atomic groups are expressed tend to be scattered due to further algorithmic processing. Overall, the quantitative results show that the rollback plan generation algorithm not only generates complete and correct plans (as showed in Section 5.1), but also the rollback plan generation minimally interferes in the change scheduling process regarding to delays.

Finally, we can highlight two other important aspects for this case study analysis:

1. A factor that really matter for *service availability* – taking as an example – is the Mean Time to Recover (MTTR). However, in the scope of this thesis, using the MTTR to evaluate the effectiveness of rollback plans in different scenarios is not relevant at all. The recover time of such a failure really depends on the root of such failure: some can take hours to be repaired due to a complex uninstallation, and others can last just some minutes due to a simple device reboot. Thus, the analysis process of the root of failures is not taken into consideration in this work. On the other hand, evaluating that a reasonable fast rollback plan generation process is feasible – as made in this subsection – is possible to observe that the presented solution may minimize efforts and service unavailability as well.

2. The workflow showed in Figure 5.5 was generated by the rollback algorithm. Taking into consideration that one human operator could take the change plan workflow and generate the rollback plan manually, the total time for generating ONE rollback plan (showed in the Figure 5.5) is estimated in 670 seconds (11.16 minutes). The estimation takes into account the number of transitions, dummy activities, activities that really have associated actions, and technical BPEL details. These details are, in fact, time assumptions related to the BPEL technology: 30 seconds to create dummy activities, 5 seconds to create one transition (source and destination), and 60 seconds to create activities with actions. Such estimative is able to show that generating rollback plans using the proposed solution may be several times better compared to the crafted manner.

6 CONCLUSION

In this chapter we first present the considerations and main contributions of this thesis, also highlighting results and some analysis (Section 6.1). In a second moment, it is described some issues in the area of IT change management related to the treatment of failures (Section 6.2). Such issues had not been covered in this thesis but are promising subjects for future investigation.

6.1 Considerations and Contributions of this Thesis

In this thesis we have discussed how organizations currently implement their changes, and the importance of having a change management system able to identify fails and rollback the managed system. Most of organizations have a complex IT infrastructure where changes are deployed by humans, increasing the failure probability. For this reason, we have proposed a solution to express atomic activities in a change plan, informing the system which activities must rollback to a previous consistent state in a failure case. Also, in our solution, there are three ways to express atomicity: in RFC, operation, and activity levels. This can turn the system more specialized, making possible not only the operator to express atomicity in a low-level, but also the change manager to define which actions of a given RFC must rollback.

The obtained results shows to be coherent in the sense that they guarantee what the change requester or operator have expressed. The use of atomic groups showed that activities can be involved as a single transaction, not affecting activities of other atomic groups. Also, the algorithm that generates rollback plans demonstrated to work in compliance with the atomic groups specification, respecting the original change plan dependencies and actions.

Moreover, both case studies presented that the rollback solution helps change managers/operators to define atomicity in a more consistent way, also taking into consideration scheduling time related to the rollback plan generation. In the second case of study, even whether this thesis did not focused on optimizations, the rollback plan generation ran on the order of hundreds of milliseconds to dozens of seconds. This time is somehow lower than the time that would be spent by an experienced human operator to design the same rollback plan from scratch. Generating automatic rollback plans may help to speed up the entire change management process: from the scheduling phase to the actual change deployment.

6.2 Additional Issues related to the Treatment of Failures in IT Change Management Systems for Future Investigation

The previous sections have presented a solution to support rollback in change management systems as a manner to recover the IT environment from failures. However, other issues may represent further interesting research opportunities related to the treatment of failures in the field of IT change management.

Among the recommendations described in the ITIL Service Transition book (IT Infrastructure Library, 2007a), a consistent failure remediation plan is one of the essential parts that a change plan should contain. In this context, and considering the word definition itself, to *remediate* would be the act or process of correcting a fault or deficiency. Thus, a natural interpretation for the ITIL best practices regarding remediation, is that a change plan can either return to a previous state (rollback) or counterbalance failures trying to reach the RFC goal (compensation). In this thesis, we cover the rollback support in change management systems, which is an important method to remediate failures in a change. However, a compensation technique may also be extremely important when combined together with rollback, since both have complementary functions: the latter returns the infrastructure back to a consistent state, while the former performs some predefined actions to reach the RFC goal even with failures. The construction of a consistent compensation technique is considered an issue to the field of IT change management. Since compensation actions performed during the change may modify the original flow, it can be necessary to recalculate in which workflow point the execution must continue after the compensation is done.

Another issue that can be considered is the optimization of the rollback plan generation process. Inside this context, there are three types of optimization that can be observed: (a) the rollback plan generation time, (b) how the algorithm generates such plan in terms of organization (actions, parallelism, etc), aiming to minimize the rollback time execution, and (c) the rollback plan generation taking into account available resources (*e.g.*, human power, machine power, and others).

In the first type of optimization, reducing the rollback plan generation time can be very important due to two aforementioned aspects. The first one is when the Change Advisory Board (CAB) must approve the RFC. In this case, the generation of rollback plans could not take a long period of time since the CAB just authorize changes expressed in RFCs that do have associated remediation plans. For example, if the algorithm that generates such plan takes 4 hours, the CAB will just have the meeting, whether to decide if the plan is feasible or not, after that period. Second, and combined with the first aspect, a possible delay can be even worse if the RFC needs to be scheduled immediately. Therefore, the rollback plan generation time must keep to a minimum in order to avoid critical delays.

In the second type of optimization, having a rollback plan with a better degree of paralelism could, for example, minimize the rollback time execution. In fact, this is an important optimization since the IT environment should not remain in an inconsistent state for a long period of time. This result may be reached considering that the rollback process is not only based on the *system undo/redo* technique, where the rollback plan originally undo failed actions as proposed in this research. The algorithm that generates the rollback plan may also include other rollback actions depending on the context of the failure. For example, if a rollback plan is to undo activities A, B, and C sequentially, an optimized rollback plan for activity A would be: execute in parallel activity Z (new one), the reverse of B, and the reverse of C. In other words, depending on how the activities

that compose the original change plan are organized, the rollback plan would have new activities or even a better degree of parallelism if such optimization keep the effectiveness of the rollback process.

Finally, in the area of optimization, the generation of rollback plans could take into consideration available resources. The rollback plan was generated aligned to human and/or machine resources? Can it be executed if the company just have three available humans (in spite of ten) to perform manual rollback actions? And if the server processor does not support hundreads of parallel rollback actions? There is another way to generate such plans? The answer to these questions are not trivial and needs a consistent information model to represent such resource availability as the base to the rollback plan generation algorithm.

In the conclusion of this thesis was also identified a new trend in the area of treatment of failures in IT change management. The rollback technique presented can be classified as reactive, since rollback plans are invoked by the deployment system when failures happen. However, a proactive manner, combined with the reactive (rollback), may be more effective when dealing with uncertainty. In this case, quantifying risks in change plans may be faced as a proactive method in order to predict some possible problems before the deployment of changes.

REFERENCES

Active Endpoints. **ActiveBPEL Open Source Engine**. [S.l.: s.n.], 2009. Available at: http://www.activebpel.org. Visited on: Jan. 2009.

ALVES, R. S.; GRANVILLE, L. Z.; ALMEIDA, M. J. B.; TAROUCO, L. M. R. A Protocol for Atomic Deployment of Management Policies in QoS-Enabled Networks. In: IEEE INTERNATIONAL WORKSHOP ON IP OPERATIONS AND MANAGEMENT (IPOM 2006), 6. **Anais...** Springer, 2006. p.132–143. (Lecture Notes in Computer Science, v.4268).

American Heritage Dictionary. 3.ed. Boston: Houghton Mifflin, 1996.

ANDRZEJAK, A.; HERMANN, U.; SAHAI, A. FEEDBACKFLOW-An Adaptive Workflow Generator for Systems Management. In: INTERNATIONAL CONFERENCE ON AUTOMATIC COMPUTING (ICAC 2006), 2. **Anais...** IEEE Computer Society, 2005. p.335–336.

BARTOLINI, C.; SAUVE, J.; TRASTOUR, D. IT Service Management driven by Business Objectives - An Application to Incident Management. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS 2006), 11., Vancouver, Canada. **Anais...** [S.l.: s.n.], 2006. p.45–55.

CANDEA, G.; BROWN, A. B.; FOX, A.; PATTERSON, D. A. Recovery-Oriented Computing: building multitier dependability. **IEEE Computer**, [S.l.], v.37, n.11, p.60–67, 2004.

COSTA CORDEIRO, W. L. da; MACHADO, G. S.; DAITX, F. F.; BOTH, C. B.; GASPARY, L. P.; GRANVILLE, L. Z.; SAIKOSKI, K.; SAHAI, A.; BARTOLINI, C.; TRASTOUR, D. A Template-based Solution to Support Knowledge Reuse in IT Change Design. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS 2008), 11., Salvador, Brazil. **Anais...** [S.l.: s.n.], 2008. p.355–362.

ELNOZAHY, M.; ALVISI, L.; WANG, Y. min; JOHNSON, D. B. **A survey of rollback-recovery protocols in message-passing systems**. [S.l.]: ACM Computing Surveys, 1996.

ENNS, R. **NETCONF Configuration Protocol**. 2006. n.4741. (Request for Comments).

FTSE Group. [S.l.]: Stock Market Indices Provider, 2009. Available at: http://www.ftse.com. Visited on: Mar. 2009.

HEWLETT-PACKARD News Website. [S.l.]: Survey: Business Continuity and Availability Solutions, a High Priority for Corporate Spending in 2007, 2007. Available at: http://www.hp.com/hpinfo/newsroom/press/2007/070326a.html. Visited on: Mar. 2009.

ISACA. **Control Objectives for Information and related Technologies (COBIT)**. [S.l.: s.n.], 2009. Available at: http://www.isaca.org/cobit. Visited on: Jan. 2009.

IT Infrastructure Library. **ITIL Service Transition Version 3.0**. [S.l.]: Office of Government Commerce, 2007.

IT Infrastructure Library. **ITIL Service Design Version 3.0**. [S.l.]: Office of Government Commerce, 2007.

ITIL. **Information Technology Infrastructure Library (ITIL)**. [S.l.]: Office of Government Commerce (OGC), 2009. Available at: http://www.itil.co.uk/. Visited on: Jan. 2009.

KELLER, A.; HELLERSTEIN, J. L.; WOLF, J. L.; WU, K.-L.; KRISHNAN, V. The CHAMPS System: change management with planning and scheduling. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS 2004), 9., Seoul, Korea. **Anais...** [S.l.: s.n.], 2004. p.395–408.

OASIS Standards. **Business Process Execution Language Version 2.0**. [S.l.: s.n.], 2007. Available at: http://docs.oasis-open.org/wsbpel/2.0/. Visited on: Apr. 2007.

PATTERSON, D.; BROWN, A.; BROADWELL, P.; CANDEA, G.; CHEN, M.; CUTLER, J.; ENRIQUEZ, P.; FOX, A.; KICIMAN, E.; MERZBACHER, M.; OPPENHEIMER, D.; SASTRY, N.; TETZLAFF, W.; TRAUPAMN, J.; TREUHAFT, N. **Recovery Oriented Computing (ROC)**: motivation, definition, techniques, and case studies. [S.l.]: UC Berkeley, 2002.

REBOUÇAS, R.; SAUVE, J.; MOURA, A.; BARTOLINI, C.; TRASTOUR, D. A decision support tool to optimize scheduling of IT changes. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT (IM 2007), 10., Munich, Germany. **Anais...** [S.l.: s.n.], 2007. p.343–352.

RIDLEY, G.; YOUNG, J.; CARROLL, P. COBIT and Its Utilization: a framework from the literature. In: HICSS '04: PROCEEDINGS OF THE PROCEEDINGS OF THE 37TH ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (HICSS'04) - TRACK 8, Washington, DC, USA. **Anais...** IEEE Computer Society, 2004. p.80233.

The GGF CDDLM working group. **Configuration Description, Deployment, and Lifecycle Management**. [S.l.: s.n.], 2008. Available at: https://forge.gridforum.org/projects/cddlm-wg. Visited on: Sept. 2008.

The Workflow Management Coalition Specification. **Workflow Process Definition Interface - XML Process Definition Language**. [S.l.: s.n.], 2007. Available at: http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf. Visited on: Sept. 2007.

W3C Note. **Web Services Description Language 1.1 (WSDL)**. [S.l.: s.n.], 2009. Available at: http://www.w3.org/TR/wsdl. Visited on: Jan. 2009.

WHITEPAPER: IT Operations Management Solutions. [S.l.]: Dimension Data Holdings, 2005. Available at: http://www.coleman-parkes.co.uk/material/Whitepaper-v3.pdf. Visited on: Mar. 2009.

APPENDIX A RESUMO ESTENDIDO DA DISSERTAÇÃO

Atualmente, empresas e organizações de grande porte não podem oferecer serviços de qualidade sem empregar uma sofisticada infra-estrutura de TI para suportar seus negócios. Por sua vez, infra-estruturas de TI geralmente possuem complexidade de gerência considerável, trazendo assim custos elevados para sua manutenção. Neste contexto, adotar políticas de gerência racionais para infra-estruturas de TI torna-se um ponto crítico para as organizações. Para alcançar uma gerência apropriada – e assim reduzir custos – o ITIL (*Information Technology Infrastructure Library*) (ITIL, 2009) compilou um conjunto de processos e boas práticas que ajudam as organizações à manter suas infra-estruturas de maneira adequada.

Entre os processos propostos pelo ITIL, o gerenciamento de mudanças (IT Infrastructure Library, 2007a) é aquele que define como mudanças em TI devem ser planejadas, agendadas, implementadas e avaliadas. A importância da gerência de mudanças reside no fato de que as mudanças em uma infra-estrutura de TI devem ser executadas de forma que não levem o sistema gerenciado a um estado desconhecido ou inconsistente. Assim, mudanças em infra-estruturas de TI são expressas primeiramente em documentos intitulados requisições de mudança (*Request for Change* - RFC), que definem quais mudanças são necessárias, mas não especifica porém como elas devem ser executadas. A definição de uma RFC é o primeiro passo do processo que irá gerar um plano de mudança (*change plan*), que essencialmente é um *workflow* composto por atividades concretas e de um nível de abstração mais baixo. O papel do plano de mudança, uma vez executado, é o de levar o sistema gerenciado para um novo estado de execução consistente que reflita as mudanças solicitadas na RFC original.

Apesar de a área de gerenciamento de mudanças ser nova e ainda pouco explorada, alguns problemas em potencial já foram investigados (KELLER et al., 2004) (BARTOLINI; SAUVE; TRASTOUR, 2006) (REBOUÇAS et al., 2007). Devido à complexidade intrínseca do assunto, as pesquisas desenvolvidas até o momento se basearam em algumas premissas que permitiram chegar a diversas conclusões importantes sobre vários aspectos em gerenciamento de TI. Uma destas premissas é a de que uma vez aprovadas, as atividades de uma RFC irão sempre ser implantadas com sucesso, levando a infra-estrutura de TI para o próximo estado consistente. Na verdade, essa suposição não reflete a realidade dos ambientes de TI, já que falhas durante a execução das mudanças efetivamente ocorrem, e assim não podem ser ignoradas.

Esta dissertação aborda a necessidade de tratar falhas durante a execução de um plano de mudança, evitando assim que a infra-estrutura gerenciada evolua para um estado inconsistente e desconhecido. Para atacar este problema, esta pesquisa propõe uma solução que garanta que depois da implantação de uma RFC, a infra-estrutura gerenciada evoluirá para um novo estado consistente, ou então retornará ao estado imediatamente anterior à

RFC. Em outras palavras, objetiva-se que a implantação de uma RFC seja tratada como uma transação atômica. Para suportar este comportamento, é proposto um modelo de *rollback* para suportar falhas em planos de mudanças. Quando uma falha ocorrer durante a execução de um plano de mudança, um procedimento de *rollback* é chamado para desfazer as mudanças executadas até então. Na verdade, pode-se observar que para alguns cenários de TI pode ser muito restritivo se só considerarmos uma RFC como o unico possível elemento atômico. Então, é proposto que outros elementos atômicos possam ser complementarmente definidos em uma menor granularidade do que uma RFC (por exemplo, a nível do plano de mudança). Assim, o operador pode definir – usando o conceito de atomicidade – quais as ações de baixo nível em especifico irão possuir um procedimento de *rollback*.

Para fornecer suporte à transações atômicas no contexto de gerenciamento de TI, empregamos um conjunto de técnicas em um protótipo desenvolvido para avaliar nossa solução. Em particular, exploramos alguns mecanismos relacionados ao disparo de exceções definidos na *Business Process Execution Language* (BPEL) (OASIS Standards, 2007). No protótipo implementado, transações atômicas no nível de RFC e planos de mudança associados, definidos por operadores do sistema, são traduzidos em construções BPEL por um algoritmo de mapeamento. Este algoritmo de mapeamento é de extrema importância no processo, pois adiciona construções BPEL que identificam falhas na execução, invocando então ações de *rollback*. As ações de *rollback* por sua vez, são geradas automaticamente por um algoritmo que leva em consideração a especificação de atomicidade dos planos de mudança. Assim, um conjunto de experimentos é apresentado para observar o impacto da proposta em um sistema de gerenciamento de mudanças sobre uma infra-estrutura de TI. Esses experimentos são focados em cenários para a obtenção de uma análise tanto qualitativa quanto quantitativa.

Considerando a proposta como um todo, essa dissertação apresenta uma solução fim a fim para habilitar *rollback* em sistemas de gerenciamento de TI. Essencialmente, a solução fim a fim é composta por todos os passos contidas na especificação de planos de mudanças com suporte a rollback: desde a construção de planos de mudança com capacidade de recuperação de falhas, até a implantação da mudança invocando planos de *rollback*.

A.1 Resumo das Principais Contribuições

As principais contribuições desse trabalho estão sucintamente descritos em itens a seguir:

- Apesar de não existir uma única arquitetura amplamente utilizada para gerenciamento de mudanças em TI, este trabalho propõe uma arquitetura genérica com os componentes básicos para que a solução de suporte a *rollback* seja funcional;

- Para que o suporte de *rollback* seja possível, é proposto que o conceito de atomicidade em planos de mudanças seja introduzido. Assim, o operador tem a possibilidade de expressar atomicidade em diferentes níveis de uma RFC: desde o mais alto nível, até a granularidade mais baixa que são as atividades finais de um plano de mudança.

- Um modelo de planos de mudanças com componentes para a solução de *rollback* é proposto. Nesse modelo é definido todas as classes para a representação conceitual de uma mudança com suporte a definição atômica de partes da mudança.

- Os principais passos da geração de um plano de rollback é descrito, formando assim o algoritmo que produz planos de rollback a partir da instanciação de classes do modelo proposto.

- A implementação do protótipo desenvolvido é abordado, bem como seus principais componentes e as estruturas BPEL que foram utilizadas para identificar falhas no processo de mudança.

A.2 Conclusões

Nesta dissertação, discutimos como organizações implantam suas mudanças e a importância de se ter um sistema de gerenciamento de mudanças com suporte a *rollback*. Muitas das organizações tem uma infra-estrutura de TI complexa, onde mudanças são implantadas por humanos, aumentando a probabilidade de ocorrerem falhas. Por esta razão, com base em trabalhos passados desenvolvidos por este grupo de pesquisa, propomos um algoritmo de geração de ações de *rollback* que se encaixa em uma arquitetura de gerenciamento de mudanças. Assim, nesta arquitetura, um administrador/operador possui a possibilidade de marcar diversas partes de uma RFC como atômica. Caso haja alguma falha na implantação desta RFC, um *workflow* de ações de *rollback* previamente gerado será executado para levar o estado do sistema para um estado consistente.

Os resultados obtidos demonstram coerência entre a especificação de grupos atômicos em um *change plan* e o *workflow* de ações de *rollback*. Ou seja, seguindo as regras de especificação de grupos atômicos, e executando os passos do algoritmo proposto, pode-se chegar em um *workflow* que reverte falhas de um *change plan* real. Também foi possível observar que apesar do grande tamanho dos arquivos BPEL gerados para o *workflow*, o uso de atividades *dummy* não influenciou no processamento destes. Além disso, podemos salientar a reutilização das estruturas do *workflow* original no algoritmo de geração de ações de *rollback*, otimizando o seu processamento.

Ainda, ambos os casos de estudo apresentam que a solução proposta ajudam os operadores a definir atomicidade em planos de mudança de uma forma consistente, também levando em consideração o tempo de agendamento das mudanças relacionado ao tempo de geração de planos de *rollback*. No segundo caso de estudo, mesmo que este trabalho não seja focado em otimizações, a geração de planos de *rollback* foi obtido na ordem de centenas a dezenas de milisegundos. Esse tempo de certa forma é menor do que o tempo gasto por um operador humano em construir planos de *rollback*. A geração automática de planos ajuda a melhorar a velocidade do processo de gerenciamento de mudanças: desde a fase de agendamento, até a implantação das mudanças sobre a infra-estrutura.

A.3 Trabalhos Futuros

Esta tese apresentou uma solução para o suporte de *rollback* em sistemas de gerenciamento de mudanças como uma maneira de recuperar de falhas um ambiente de TI. Porém, outros problemas podem representar oportunidades de pesquisa relacionados ao tratamento de falhas no campo de gerenciamento de mudanças.

Entre as recomendações descritas pelo livro ITIL *Service Transition* (IT Infrastructure Library, 2007a), um plano de remediação consistente é uma das partes essenciais que um plano de mudança deve conter. Nesse contexto, e considerando a palavra remediação por si só, remediar pode ser encarado como o ato ou processo de corrigir uma falha ou

deficiência. Assim, uma interpretação natural para as boas práticas do ITIL relacionado à remediação, seria que um plano de mudança pode tanto voltar para um estado anterior (*rollback*) como contrabalancear falhas para atingir o objetivo de uma RFC (compensação). Nesta tese, é apresentado a solução para o suporte de *rollback* em sistemas de gerenciamento de mudanças, o que é um importante método para remediar falhas em mudanças. Porém, uma técnica de compensação pode também ser extremamente importante quando combinado com *rollback*, pois ambos possuem funções complementares: o último retorna a infra-estrutura para um estado consistente, enquanto o primeiro executa ações pré-definidas para atingir o objetivo da RFC mesmo havendo falhas. A construção de uma técnica de compensação consistente é considerado um problema para a área de gerenciamento de mudanças em TI. Dado que ações compensatórias executadas durante mudanças podem modificar o fluxo natural do plano, pode ser necessário recalcular em qual ponto do *workflow* a execução deverá continuar após que a compensação estiver concluída.

Outro problema que pode ser considerado é a otimização do processo da geração de planos de *rollback*. Nesse contexto há três tipos de otimizações que podem ser observados: (a) o tempo de geração de planos de *rollback*, (b) como o algoritmo gera esses planos em termos de organização (actions, paralelismo, etc.) objetivando minimizar o tempo de execução do *rollback*, e (c) a geração de planos de *rollback* levando em consideração recursos disponíveis (por exemplo, unidade de força humana de trabalho, unidade de força de máquinas, e outros).

No primeiro tipo de otimização, reduzir o tempo de geração dos planos de *rollback* pode ser bastante importante por dois aspectos. O primeiro é quando o *Change Advisory Board* (CAB) precisa aprovar a RFC. Nesse caso, a geração dos planos de *rollback* não pode levar um longo período de tempo se considerar que o CAB só autoriza mudanças expressadas em RFCs que tiverem um plano de remediação associado. Por exemplo, se o algoritmo que gera tal plano demorar 4 horas, o CAB só terá a reunião para decidir se o plano é factível ou não depois deste período de tempo. Além disso, e combinado com o primeiro aspecto, um possível atraso pode ser ainda pior se a RFC precisa ser agendada imediatamente. Então, o tempo de geração de planos de *rollback* precisa ser o mínimo possível para evitar atrasos altamente críticos.

No segundo tipo de otimização, ter um plano de *rollback* com um melhor grau de paralelismo pode, por exemplo, minimizar o tempo de execução de *rollback*. Na verdade, essa é uma importante otimização dado que o ambiente de TI não pode permanecer em um estado inconsistente por um longo período de tempo. Esse resultado poderia ser alcançado considerando que o processo de *rollback* não é somente baseado na técnica de *system undo/redo*, onde o plano de *rollback* originalmente desfaz as ações que falharam (como proposto nessa tese). O algoritmo que gera o plano de *rollback* poderia incluir outras ações dependendo do contexto da falha. Por exemplo, se um plano de *rollback* é desfazer as atividades A, B e C sequencialmente, um plano de *rollback* otimizado para a atividade A seria: executar em paralelo a atividade Z (new one), desfazer a B, e desfazer a C. Em outras palavras, dependendo de como as atividades que compõe o plano de mudança original estão organizados, o plano de *rollback* teria novas atividades ou até mesmo conter um melhor grau de paralelismo.

Por último na área de otimização, a geração de planos de *rollback* poderia levar em consideração recursos disponíveis. O plano foi gerado alinhado com recursos humanos e/ou tecnológicos? Pode ser executado se a determinada empresa tiver somente três humanos disponíveis (ao invés de dez) para executar ações de *rollback* de forma manual? E se o processador de um servidor não suportar centenas de ações de *rollback* parale-

lamente? Existe outra maneira de gerar esses planos? A resposta para essas perguntas não são triviais e precisam de um modelo de informação consistente para representar a disponibilidade de recursos servindo como base para o algoritmo de geração de planos de *rollback*.

Na conclusão desta tese também foi identificado um novo desafio na área de tratamento de falhas no gerenciamento de mudanças em TI. A técnica de *rollback* apresentada nesta tese pode ser classificada como reativa, dado que planos são invocados pelo *deployment system* quando falhas ocorrem. Porém, uma maneira pró-ativa, combinado com a reativa (*rollback*), pode ser mais eficaz quando estiver sendo lidado com a incerteza. Nesse caso, quantificar riscos em planos de mudança pode ser encarado como um método pró-ativo para predizer alguns possíveis problemas. Assim, falhas poderiam ser evitadas antes de ocorrer a implantação da mudança em um amibente de TI.