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

WEVERTON LUIS DA COSTA CORDEIRO

ChangeLedge: Change Design and Planning in Networked Systems based on Reuse of Knowledge and Automation

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

Prof. Dr. Luciano Paschoal Gaspary
Advisor

Porto Alegre, February 2009

CIP – CATALOGING-IN-PUBLICATION

Cordeiro, Weverton Luis da Costa

ChangeLedge: Change Design and Planning in Networked Systems based on Reuse of Knowledge and Automation / Weverton Luis da Costa Cordeiro. – Porto Alegre: PPGC da UFRGS, 2009.

104 f.: il.

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

1. Information Technology. 2. ITIL. 3. Change Management. 4. Change Templates. 5. Requests for Change. I. Gaspary, Luciano Paschoal. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Pedro Cezar Dutra Fonseca
Pró-Reitora de Pós-Graduação: Prof. Aldo Bolten Lucion
Diretor do Instituto de Informática: Prof. Flávio Rech Wagner
Coordenador do PPGC: Prof. Álvaro Freitas Moreira
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

"You'll never get to heaven
if you're scared of getting high. "
— Kylie Minogue

"With men this is impossible: but
with God all things are possible. "
— Matthew 19:26

ACKNOWLEDGEMENTS

First of all, I thank God for all the blessing He has given me, and for all the support I have received throughout my journey. All honor and glory to our Lord Jesus Christ.

To my parents, Weliton and Ana Margarete da Costa Cordeiro, for the support I have received through all my life. I want you to know that all my achievements are a result of the education and lessons you have taught me. Also thanks to my brother, Werley, and my sister, Lis Marina. And my special regards to my aunt Marinês Vieira da Costa, who was very important in the beginning of my journey in Belém, Brazil.

Also thanks to Aldete das Graças Serra da Costa and Ladislau Cavalheiro da Costa, who have adopted me as a son during my stay in Belém. Also my best regards to Anderson Jorge, Ana Carolina, and Maria Auxiliadora da Silva Tavares.

Thanks to Profs. Antônio Abelém, Carla Reis, and Rodrigo Reis, for the support and lessons that will last for a lifetime. My regards to my colleagues and friends from the Federal University of Pará, in special to Adailton Lima, Breno França, and Rafael Esteves, for the great friendship and help you have given to me.

It is also a great pleasure for me to remind the people who have made my stay in Porto Alegre a very funny experience: Vânia Rodenas, Kênia Lilian, Neila Rezende, William Bruno, Flávio “Cockroach”, and the folks from former Eraldo’s shared house.

My special thanks to Profs. Luciano Gaspary, Lisandro Granville, and Marinho Barcellos, for all the lessons taught and the holy patience in putting me up throughout my masters. The good news is that this period is finally over. Bad news is that there are four more years ahead!

I cannot forget the members of the II/UFRGS Network Group. I hope you have learned something good from me, as I have learned valuable lessons from you: Ewerton, Raniery, Clarissa, Giovane, Juliano, Roben, Jeferson, Flávio, Alan, Cristiano, and Fabrício. To Guilherme, my sincere apologies. Not being sufficient to put up with me in Brazil, he had to tolerate me in Europe as well! Guilherme, great success to you in your new journey in Switzerland, “are you plugged in”?

Thanks also to David Trastour, Claudio Bartolini, Abdel Boulmakoul, and Robert Fink, for the cooperation, barbecues, the muqueca party, and talks that have made my stay in Bristol, UK a very pleasant period.

Finally, a very special thanks to my girlfriend (and future wife, if God wish) Sarita Nunes Loureiro. Thank you very much for your tenderness, support, and comprehension. Soon we will overcome this distance that separates us! I love you, my darling!

AGRADECIMENTOS

Agradeço primeiramente à Deus, por todas as oportunidades que Ele tem me concedido, e por Ele ter sempre me apoioado na minha caminhada. Toda honra e toda glória seja dada ao Nosso Senhor Jesus Cristo.

Aos meus pais, Weliton e Ana Margarete da Costa Cordeiro, pelo apoio que eu sempre recebi em todos os momentos da minha vida. Quero que vocês saibam que todas as minhas conquistas são resultado da educação e dos ensinamentos que vocês me transmitiram. Meus agradecimentos também aos meus irmãos Werley e Lis Marina. Minha lembrança especial à minha tia Marinês Vieira da Costa, a qual foi fundamental no início da minha jornada em Belém.

Agradeço também à Aldete das Graças Serra da Costa e ao Ladislau Cavalheiro da Costa, os quais me adotaram como um filho durante a minha estadia em Belém. Minhas lembranças também ao Anderson Jorge, Ana Carolina, e à Maria Auxiliadora da Silva Tavares.

Aos Profs. Antônio Abelém, Carla Reis e Rodrigo Reis, pelo apoio e ensinamentos que ficarão para uma vida inteira. Minhas lembranças também aos colegas e amigos do Curso de Computação da UFPA, em especial ao Adailton Lima, Breno França, e Rafael Esteves, pela grande amizade e ajuda que sempre me foi prestada.

É um grande prazer também lembrar aqui as pessoas que tornaram a minha estadia em Porto Alegre super divertida: Vânia Rodenas, Kênia Lilian, Neila Rezende, William Bruno, o Flávio "Barata", e toda a galera da ex-pensão do Eraldo.

Meu agradecimento todo especial aos Profs. Luciano Gaspary, Lisandro Granville e Marinho Barcellos, pelos ensinamentos e pela santa paciência em terem conseguido me aturar durante o mestrado. A boa notícia para vocês é que esse período finalmente acabou. A má notícia é que agora tem mais quatro anos pela frente!

Não posso esquecer também os integrantes do Grupo de Redes do II/UFRGS. Espero que vocês tenham aprendido algo de positivo comigo, assim como aprendi lições importantes com vocês: Ewerton, Raniery, Clarissa, Giovane, Juliano, Roben, Jéferson, Flávio, Alan, Cristiano e Fabrício. Ao Guilherme, minhas sinceras desculpas. Não bastava ter que me aturar no Brasil, teve que me aguentar na Europa também! Guilherme, sucesso lá na tua nova empreitada na Suíça, tá ligado?

Agradecimentos também ao David Trastour, Claudio Bartolini, Abdel Boulmakoul e Robert Fink, pela cooperação, churrascos, muqueca e conversas que tornaram a estadia em Bristol (UK) super agradável.

Para finalizar, um especial agradecimento à minha namorada (e se Deus quiser, futura esposa) Sarita Nunes Loureiro. Muito obrigado pelo seu carinho, apoio e compreensão. Logo logo haveremos de superar a distância que nos separa! Eu te amo, meu amor!

TABLE OF CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
AMN	Activity Modeling Notation
API	Application Programming Interface
BPEL	Business Process Execution Language
CAB	Change Advisory Board
CHAMPS	Change Management with Planning and Scheduling
CI	Configuration Item
CIM	Common Information Model
CM	Causality Metric
CMDB	Configuration Management Database
CMS	Configuration Management System
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
GM	Global Metric
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ITIL	Information Technology Infrastructure Library
IT	Information Technology
ITSM	Information Technology Service Management
LM	Local Metric
OASIS	Advancing Open Standards for the Information Society

OGC	Office of Government Commerce
OS	Operating System
RFC	Requests for Change
SLA	Service Level Agreement
StAX	Streaming API for XML
URL	Uniform Resource Locator
WfMC	Workflow Management Coalition
XML	Extended Markup Language

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Proper management of Information Technology (IT) resources and services has become imperative for the success of modern organizations. The IT Infrastructure Library (ITIL) represents, in this context, the most widely accepted framework to help achieve this end. In order to deal with IT changes, ITIL defines the change management process, whose main goal is to ensure that standardized methods and procedures are used for the efficient and prompt handling of these changes. To meet this goal, it is of paramount importance reusing the experience acquired from previous changes in the design of subsequent ones. ITIL suggests the use of change models as a mean of enabling the reuse of such experience across recorrent, similar changes. The creation of change models can be done considering two distinct approaches. In a top-down approach, IT operators may manually design models based on the knowledge owned/acquired in the past. In contrast, in a bottom-up perspective, these models could be discovered from past execution traces gathered from IT provisioning tools. In practice, however, changes have been usually described and documented in an *ad hoc* fashion, due to the lack of proper mechanisms to both support the change design process. This hampers knowledge acquired when specifying, planning, and carrying out previous changes to be reused in subsequent requests. To address this problem, in this thesis we propose (*i*) the concept of change templates as a mechanism to formalize, preserve, and (re)use knowledge in the specification of (recurrent and similar) IT changes, (*ii*) an algorithm for the automated refinement of change plans into actionable workflows, (*iii*) a mechanism to discover change templates from traces of past changes, and (*iv*) an end-to-end solution, supported by a real system, to allow planning and implementation of IT changes to be designed and executed. To prove concept and technical feasibility of the proposed solution, we have developed a prototypical implementation of a change management system called ChangeLedge and used it to carry out a set of experiments, considering typical IT changes. The results obtained indicate the effectiveness of the solution and efficiency of the system, which is able to generate accurate and actionable change plans in substantially less time than would be spent by a skilled human operator, and to extract templates that accurately describe IT change procedures previously executed in the organization.

Keywords: Information Technology, ITIL, Change Management, Change Templates, Requests for Change.

ChangeLedge: Projeto e Planejamento de Mudanças em Sistemas de Rede com base no Reuso de Conhecimento e Automação

RESUMO

A gerência adequada de recursos e serviços de Tecnologia da Informação (TI) se tornou imperativa para o sucesso de organizações modernas. A Biblioteca de Infraestrutura de Tecnologia da Informação (*Information Technology Infrastructure Library*, ITIL) representa, nesse contexto, o *framework* mais popular para ajudar a alcançar esse fim. Para lidar com mudanças em TI, a ITIL define o processo de gerência de mudanças (*change management*), cujo principal objetivo é garantir que métodos e procedimentos padronizados são utilizados para o tratamento imediato e eficiente dessas mudanças. Para alcançar esse objetivo, é fundamental reutilizar a experiência adquirida com mudanças passadas no projeto de requisições futuras. A ITIL sugere o uso de modelos de mudanças (*change models*) como uma forma para permitir o reuso de tal experiência em mudanças recorrentes e similares. A criação de modelos de mudanças pode ser concretizada considerando duas abordagens distintas. Em uma abordagem *top-down*, operadores de TI podem projetar os modelos manualmente, com base no conhecimento adquirido no passado. Em uma perspectiva alternativa, *bottom-up*, esses modelos poderiam ser extraídos a partir de traços de mudanças passadas obtidos com orquestradores de mudanças. Na prática, no entanto, mudanças tem sido geralmente descritas e documentadas de forma *ad hoc*, devido à falta de mecanismos adequados para apoiar o projeto das mesmas. Isso impede que o conhecimento adquirido na especificação, planejamento e condução de mudanças passadas seja reutilizado em requisições futuras. Para abordar esse problema, nesta dissertação são propostos (*i*) o conceito de *templates* de mudança como um mecanismo para formalizar, preservar, e (re)usar conhecimento na especificação de mudanças recorrentes e similares, (*ii*) um algoritmo para o refinamento automatizado de planos de mudanças em *workflows* executáveis, (*iii*) um mecanismo para extrair *templates* de mudanças a partir de traços de execuções passadas, e (*iv*) uma solução fim-a-fim, apoiada por um sistema real, para permitir o planejamento e implantação de mudanças em TI. Para provar conceito e viabilidade técnica da solução proposta, foi realizada uma implementação prototípica de um sistema de gerência de mudanças chamado ChangeLedge, o qual foi utilizado para conduzir uma série de experimentos considerando mudanças típicas em TI. Os resultados alcançados indicam a efetividade da solução e eficiência do sistema, o qual é capaz de gerar planos de mudança executáveis e corretos em um período de tempo substancialmente menor que o que seria gasto por um operador humano experiente, e de extrair *templates* que descrevem com acurácia mudanças passadas executadas na organização.

Palavras-chave: Tecnologia da Informação, ITIL, Gerência de Mudanças, *Templates* de Mudança, Requisições de Mudança.

1 INTRODUCTION

The use and importance of networked, distributed systems to support the business of organizations has increased significantly in recent years. This scenario has made imperative the employment of effective approaches for the proper and efficient handling of these systems (also referred to as Information Technology systems in this thesis). The Information Technology Infrastructure Library (ITIL) (OGC, 2008) has become, in this context, one of the most widely accepted and adopted set of best practices and processes for IT service deployment and management (REBOUCAS et al., 2007), being of special importance to those organizations characterized by their large scale and rapidly changing, dynamic services.

In order to deal with changes in IT systems, frequently introduced in response to business needs, ITIL defines the change management process (LACY; MACFARLANE, 2007). This process has the goal of maximizing the value of changes to business, by means of minimizing change-related incidents and service-delivery disruption. To this end, change management recommends the use of standardized methods and procedures to deal with these changes.

As described in the ITIL Service Transition book (LACY; MACFARLANE, 2007), change management comprises from the specification of *Requests For Change* (RFC) documents, by a *change initiator*, to the generation, either by an *IT operator* or automatically, of *actionable change plans*. RFC documents, typically triggered by business users, express the requested changes in a high level of abstraction (for example, *Deploy New e-Commerce Service* and *Increase Data Link Capacity*). Actionable change plans, in turn, are workflows that comprise low level actions (for example, software installation, route table manipulation, and setting modification) to be deployed upon the managed IT infrastructure. If executed, these actionable change plans (also called *actionable workflows* throughout this thesis) should deploy the requested changes. The subsequent phases of a traditional change management process are *evaluation*, *test*, *authorization*, *scheduling*, *implementation*, and *record* of the executed actions in the *Configuration Management Database* (CMDB).

To meet the change management goal of maximizing the value of changes to business, it is of paramount importance reusing the experience acquired from the execution of previous changes in the design of subsequent ones. ITIL suggests the use of change models as a mean of enabling the reuse of such experience across recurrent, similar change requests. Ideally, the creation of these models can be done considering two distinct approaches. In a top-down approach, IT operators may design models based on the knowledge owned/acquired in past changes. In contrast, in a bottom-up perspective, these models could be discovered from execution logs gathered from IT provisioning tools (or from any general-purpose workflow orchestration engine).

In practice, however, RFCs and change plans are traditionally modeled in an *ad hoc* fashion (REBOUCAS et al., 2007), where natural language is often employed to express *what*, *why*, *when*, *where*, and *how* changes should be executed. The lack of a common standard widely accepted and adopted by business users and IT operators to assist the design of changes often leads, for example, to documents specified with either too much or not enough information than actually required, and to erroneous interpretation of the produced change-related documents. More importantly – and the focus of the investigation presented in this thesis – it hampers knowledge owned/acquired by the personnel responsible for specifying, planning, and carrying out changes to be reused in recurrent or similar requests.

We highlight two reasons that justify the importance of reusing the experience of operators in the implementation of IT changes. First, as change plans are recurrently instantiated, they become more stable and, therefore, their use may result in fewer incidents (upon change execution). Second, the time consumed by the IT personnel in the specification and planning of changes tends to be reduced, given that they will seldom be generated "from scratch".

Since the inception of ITIL, there has been substantial research on IT change management issues. For example, important steps have been taken towards exploring parallelism opportunities in the execution of change plans (KELLER et al., 2004), optimizing both the scheduling of changes to maintenance windows and the assignment of technicians to execute them (TRASTOUR et al., 2007), and guiding the scheduling of changes as to satisfy business objectives/restrictions (REBOUCAS et al., 2007). However, despite the potential benefits of reusing knowledge behind IT changes, this topic has been barely addressed in previous investigations (KELLER et al., 2004; LACY; MACFARLANE, 2007; KELLER, 2005) (this will be further discussed in Chapter 2).

To bridge this gap, in this thesis we present a solution, based on the guidelines for change management proposed by ITIL, to support the design and planning of changes in IT systems. The contributions of this thesis are fourfold. First, we introduce the concept of *change templates* as a mechanism to formalize, preserve, and reuse the experience accumulated within organizations in relation to IT changes. These templates may exist at different abstraction levels of a change management system. We characterize *request templates* as those used by the change initiator to specify RFC documents. Complementarily, *plan templates* comprise large-grained activities required to accomplish the RFC objectives. Second, we propose an algorithm for the automated refinement of large-grained activities present in plan templates into finer-grained ones, in order to produce detailed, actionable change plans. Our algorithm focuses on the impact that already computed, preceding actions will cause on the IT infrastructure, when computing the subsequent ones. As a consequence, the actionable plans generated using our algorithm are aware of the runtime constraints imposed by the target IT environment (*e.g.*, disk space and memory constraints). Third, we present a mechanism, inspired on process mining techniques (MARUSTER et al., 2002), to discover change templates from execution traces generated by IT provisioning tools. In contrast to the already established workflow mining techniques, our mechanism concentrates on obtaining templates that capture the essence of previously executed change processes. The discovered templates, in turn, enable the reuse and applicability of such processes in scenarios having settings diverse from the original ones (*e.g.*, different dependency relationships among affected IT components, or specific system requirements). And fourth, in order to support the reuse of knowledge through templates, the automated refinement of change plans, and the discovery of change templates from historic data, we introduce an end-to-end solution, supported by a real system, to allow planning and implementation of IT changes to be designed and executed.

The use of change templates, the refinement of preliminary plans into actionable workflows, and the discovery of templates from execution traces have been evaluated through a prototypical implementation of a change management system called ChangeLedge. In addition to allowing templates to be (manually) designed, (automatically) discovered, and (re)used in the specification of new change documents, the system is able to compute actionable change plans by correlating instantiated RFCs/sketched plans with dependency information available in IT-related repositories. The resulting actionable workflows can, in turn, be straightforwardly translated into a workflow language and executed by any off-the-shelf deployment system. To prove concept and technical feasibility of our solution, we have conducted several experiments exploring the potentialities of employing templates as a resource for preserving and reusing the expertise acquired with IT changes.

The remainder of this thesis is organized as follows. Chapter 2 discusses related work on IT change management and some of the concepts that form the basis of the investigation presented herein. Chapter 3 introduces our conceptual solution for IT change management. Chapter 4 details how the concept of templates is tailored to knowledge reuse in the context of IT change management. Chapter 5 presents the algorithm for the automated refinement of preliminary change plans into actionable workflows. Chapter 6 introduces our mechanism for extracting change templates from execution traces. Chapter 7 emphasizes implementation aspects of the ChangeLedge system and discusses the results obtained with the experimental evaluation. Finally, Chapter 8 concludes the thesis with final remarks and perspectives on both research directions and trends in the area.

2 BACKGROUND

In this chapter we briefly present key concepts in the field of IT change management (Section 2.1), and discuss the most relevant research work related to this thesis (Section 2.2).

2.1 ITSM, ITIL, and Change Management

Information Technology Service Management (ITSM) (BON; JONG et al., 2007) is a discipline for managing IT systems, philosophically centered on the customer's perspective of IT's contribution to the business. ITSM is not concerned with the details of how to use a particular vendor's product, or necessarily with the technical details of the systems under management. Instead, it focuses upon providing a framework to structure IT-related activities and the interactions of IT technical personnel with business customers and users. Because of the ever-growing importance of IT systems to corporate activity, ITSM has been the object of concentrated study over the past decade (SAUVÉ et al., 2007). As a result, best practice collections for ITSM – such as the Control Objectives for Information and related Technologies (COBIT) (ISACA, 2008) and the Information Technology Infrastructure Library (ITIL) – are becoming popular, and have been employed and validated across a diverse set of environments and situations (LACY; MACFARLANE, 2007). In this thesis, we focus our investigation on the ITIL framework.

The ITIL publication provides detailed but generic guidance to organizations, and has the following components (OGC, 2008): 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. The ITIL Core is composed of five publications, which together provide guidelines necessary for an integrated approach for service management: *Service Strategy*, *Service Design*, *Service Transition*, *Service Operation*, and *Continual Service Improvement*. Each publication addresses capabilities that have a direct impact on a service provider's performance. Figure 2.1 illustrates the structure of the ITIL Core.

Within the ITIL framework, Service Transition (LACY; MACFARLANE, 2007) "provides guidance for the development and improvement of capabilities for transitioning new and changed services into operations". This includes, as a major task, the management of IT changes. In this context, the term *change* comprises any modification to entities (CIs) involved in the operation of IT services. Therefore, change management covers minor tuning, such as granting access rights or rebooting

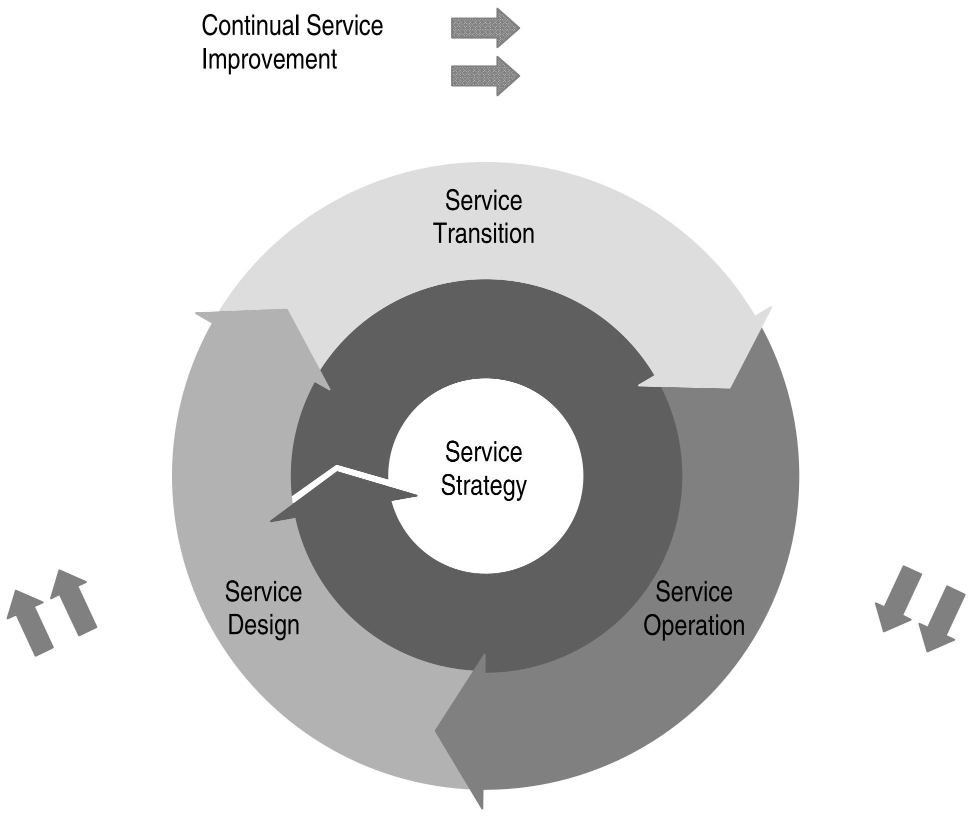


Figure 2.1: ITIL Core (LACY; MACFARLANE, 2007)

devices, as well as high level tasks, such as design and implementation of new services due to emerging business needs (FINK, 2009; LACY; MACFARLANE, 2007).

Figure 2.2 presents the typical activities of a traditional change management process. In the first activity, *Create RFC*, an RFC document is raised by a request from the initiator – the individual (*e.g.*, business user) or organizational group (*e.g.*, maintenance staff) that requires the change. For major changes having reasonable impact on business continuity or significant financial implications, a *Change Proposal* may be required, which should contain a full description of the change, along with technical/business/financial justification.

In the subsequent activity, *Change Record*, the newly requested change is recorded in the *Configuration Management Database* (CMDB). Some information must be recorded upon change creation (*e.g.*, change description, reason for the change, and contact information of the person proposing it), whereas other information are continuously created/updated throughout its lifecycle (*e.g.* executed actions). The degree of detail recorded depends on the impact and size of the change. The information that should be recorded for an RFC document are:

- Unique identification number;
- Trigger of the change (*e.g.*, problem report number, business need, legislation);
- Description;
- Identity of the item(s) to be changed, along with a description of the desired change;

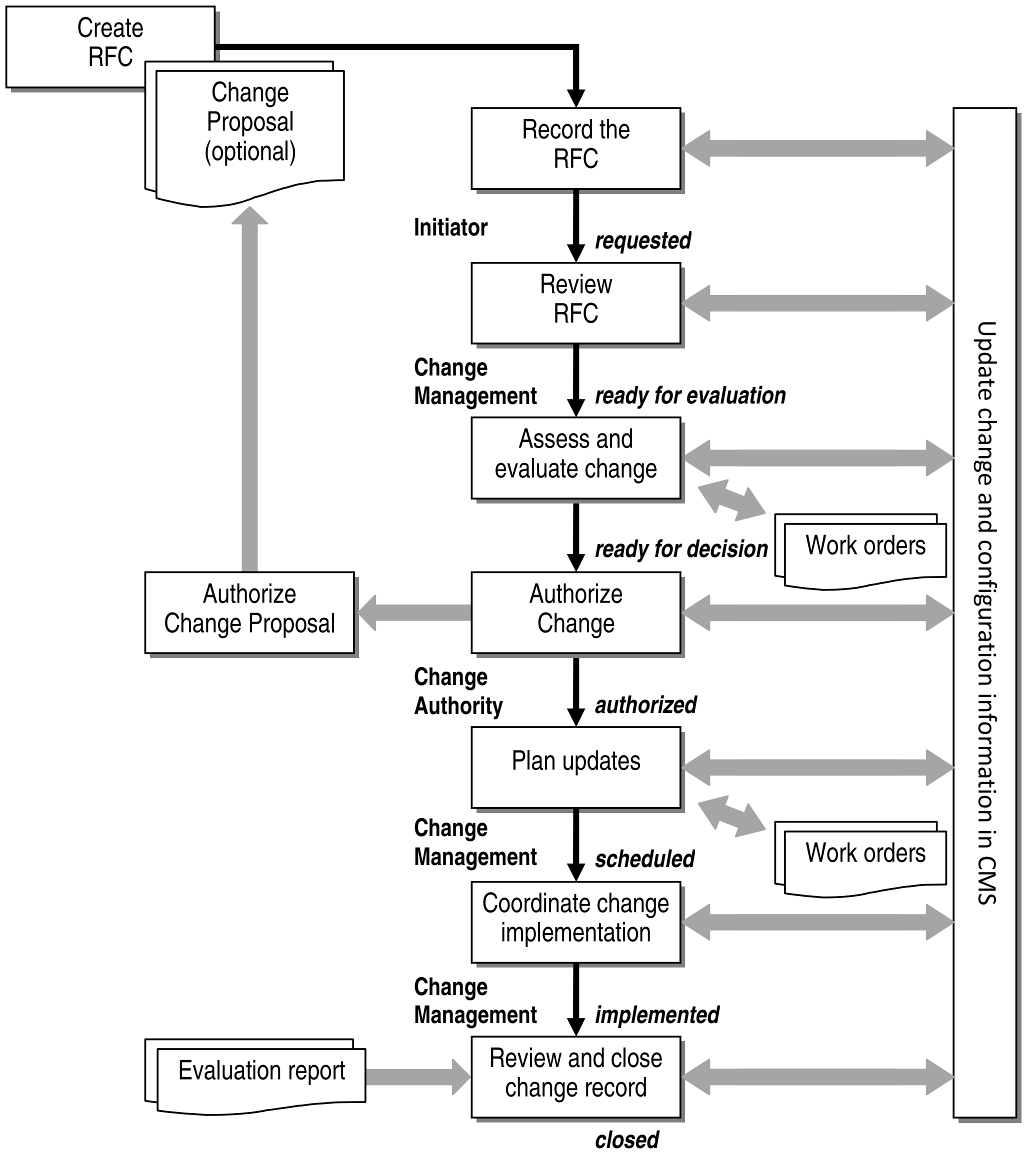


Figure 2.2: ITIL Change Management Process (LACY; MACFARLANE, 2007)

- Reason;
- (Negative) effects of not implementing the change;
- Configuration item(s) that should be primarily affected;
- Contact and details of the person proposing the change;
- Date and time the change was proposed;
- Category of the change (*e.g.*, minor, significant, major);
- Change priority (*e.g.*, low, medium, high, immediate);
- Risk assessment and risk management plan;
- Back-out or remediation plan;

- Impact assessment on business continuity;
- Decision and recommendations accompaining the change;
- Authorization for the deployment of the change (could be electronic);
- Authorization date and time;
- Scheduled implementation date and time;
- Detailed steps for change implementation (*i.e.*, change plan) ;
- Actual implementation date and time;
- Change implementation details;
- Review date;
- Review results (including reference to new RFC, when necessary);
- Closure.

In the next activity, *Review RFC*, submitted RFCs are evaluated in regard to their completeness, status (*e.g.*, accepted, rejected, under consideration), and technical/financial feasibility. Changes that do not comply with required standards or that are totally impractical, for example, should be filtered and returned to the initiator, along with a justification for the rejection.

In the *Assess and Evaluate Change* activity, the RFC document is then 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. After the RFC is authorized (activity *Authorize Change*), the required actions to perform it are planned, updated, and tested (activity *Plan Updates*). Subsequently, the RFC is passed to the team responsible for deployment upon the target IT infrastructure (activity *Co-ordinate Change Implementation*). ITIL suggests the deployment of changes to be performed using *Work Orders*, in order to enable the trackability of the RFC under implementation. Finally, on completion of the change, the results are reported for evaluation to those responsible for managing changes (activity *Review and Close Change Record*). At this stage, incidents observed during the change deployment should be reported. Moreover, business users and customers involved in the change should be contacted, in order to check whether the change met its desired results or not.

As indicated by Figure 2.2 and explicitly stated in the ITIL Service Transition publication (LACY; MACFARLANE, 2007), all activities comprising the change management process should be executed in close conjunction 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 in ITSM processes.

2.2 Related Work

In recent years, several research efforts on IT change management have been carried out within the network/service operations and management community. In parallel, the use of historic data to extract process models has been largely investigated in the areas of Software Engineeering and Data Mining. In this section, we cover some of the most prominent investigations concerning these topics. Subsection 2.2.1 presents related work in the field of IT management. Subsection 2.2.2 briefly describes some process mining techniques that support our work. Finally, Subsection 2.2.3 reviews some additional research efforts on the field of IT change management.

2.2.1 IT Change Design and Planning

The ITIL Service Transition book (LACY; MACFARLANE, 2007) recommends the use of *change models* to both define a library of recurrent changes and foresee the impact associated to them, once performed. However, since ITIL concentrates on documenting generally applicable industry best practices, it is out of its scope to propose *how* to materialize such models.

Keller *et al.* (2004) have proposed CHAMPS, a system for automating the generation of change plans that explore a high degree of parallelism. The topic of knowledge reuse was not the focus of that work. Even though the system enables some degree of *reuse* through *abstract workflows*, the scope of the reuse is limited. That is so because these workflows cannot be generalized, once they are tied to a given combination of software packages. Refinements, on the other hand, are restricted to "*editing/saving as*" the already existing workflows. The lack of specialization/generalization capabilities hampers knowledge to be better structured and systematically reused in future changes.

In regard to change planning and scheduling aspects of CHAMPS, they are approached as an optimization problem. Although the system is able to evaluate technical constraints in the planning and scheduling of changes, the scope is limited to *Service Level Agreements* (SLAs) and *policies*. Since it does not perform a fine-grained control of resource constraints, modifications on the infrastructure produced by the already processed tasks of the plan under refinement are not taken into account when computing the subsequent ones. As a consequence, the resulting change plans may not be executable in practice.

Aware of the importance of formalizing IT change documents, Keller (KELLER, 2005) has introduced, in a subsequent work, the concept of *electronic contracts*. Four types of contracts are proposed: *Requests for Change*, *Deployment Descriptors*, *Policies and Best Practices*, and *Service Level Agreements*. In respect to the specification of RFCs – central to our investigation – the author only enumerates the parameters to be supplied in an RFC, but does not propose a more robust information model to express them.

2.2.2 Process Mining

The idea of extracting workflow processes from execution logs has been explored in the field of software engineering and workflow management (ROZINAT et al., 2007). Seminal techniques proposed, however, were not able to mine workflows having and/or branches/joins (MARUSTER et al., 2002). As a consequence, the resulting workflows were limited to a purely sequential behavior.

To tackle this limitation, Maruster *et al.* (2002) have proposed a mechanism to mine workflows from event logs, which explicitly distinguishes decision points that route the workflow enactment. In a subsequent work (MEDEIROS et al., 2004), the authors have extended the class of workflows their technique can mine, by enabling their algorithm to mine short loops from event logs. More recently (ROZINAT; AALST, 2006), the authors have proposed a supplementary mechanism that enables understanding how environment data influences in the routing of a workflow execution. The proposed mechanism, called *Decision Miner*, was implemented and evaluated in the context of a process mining framework called *ProM*. The solution for change template mining, proposed in this thesis, adopts some principles and techniques introduced in the previously mentioned related work (this will be further discussed in Chapter 6).

2.2.3 Management of IT Systems

Despite not directly related with the problem addressed in this thesis, some additional research efforts on change management published in recent years merit attention. Shankar *et al.* (2006) have proposed an enhanced policy-based solution to execute changes on the IT infrastructure, triggered by the managed system and not by a human operator, in response to the occurrence of events. Dumitras *et al.* (2007) have presented *Ecotopia*, a framework for change management that schedules change operations with the goal of minimizing service-delivery disruptions. Trastour *et al.* (2007) have formulated a solution to assign changes to maintenance windows and to allocate change activities to technicians.

Looking at IT changes from a business perspective, Rebouças *et al.* (2007) have investigated approaches for the planning and scheduling of changes aiming at minimizing costs (*e.g.*, labor costs and financial loss due to SLA disruption). Sauvé *et al.* (2007) have gone a step further, by proposing a method to automatically assign priorities to changes, considering the individual exposure of each requested change to risks as its execution is postponed. Finally, in a previous work (MACHADO et al., 2008), we have introduced the concept of atomic groups in the design of change plans with the purpose of providing an end-to-end solution for change management with rollback support.

There are additional relevant research efforts published in other fields, *e.g.* Software Engineering and Knowledge Management, which explore similar aspects of our work. Nevertheless, due the fact that they are not related to neither the management of information technology systems nor the scope of our investigation, they are not addressed in this thesis.

3 CONCEPTUAL SOLUTION FOR CHANGE MANAGEMENT

As mentioned in the previous chapter, little has been done to facilitate change design and implementation by capturing, organizing, and reusing the knowledge acquired within IT organizations. To tackle this issue, we propose a conceptual solution to allow planning and implementation of IT changes to be, respectively, designed and executed. In contrast to previous investigations, our solution focuses on providing a systematic way to design changes – from RFCs to resulting change plan documents – by reusing the knowledge accumulated in the past.

In this chapter, we first characterize the issues addressed in this thesis using an example (Section 3.1). Subsequently, we introduce our solution for IT change management (Section 3.2), followed by the presentation of its building blocks (Section 3.3).

3.1 A Typical IT Change Process

Before introducing the issues associated to a typical change process, consider the environment of a research & development department of an organization. It is often composed of a set of dedicated servers (Virtual Machine servers, database servers, Web servers, *etc.*), users' workstations, and other network facilities (routers, switches, repeaters, *etc.*). In the daily operations of this department, users request changes to be performed, for example to deploy new Web applications, or to upgrade the set of software packages installed on their workstations. These requests are materialized, respectively, by RFCs such as *Deploy New Web Application* and *Upgrade Software Installed at Workstation*. Once the demands are assessed, these RFCs are *translated* into changes, and subsequently executed by the IT operator (*e.g.*, the maintenance staff). To illustrate, a possible translation of the first RFC into an actionable change could result in the following high level activities: *Install Operating System*, *Install Web Server*, *Install Web App*, *Configure Database Server*, *Configure Web Server*, *Configure Firewall*, *Publish Web App at DNS Server*, *Start Web Server*, and *Start Web Application*.

During the translation of requests into a sequence of activities (*i.e.*, an actionable change plan), the operator must consider two important information: the current state of the IT infrastructure in place, and the parameters specified in the request. Also, he/she must design an strategy to deploy the change plan, depending on the previously mentioned information. Focusing on the first RFC, the Web application can be installed on a physical or a virtual host, which may have an operating system and other software already installed (or not). Considering the current state of the host, dependencies for the installation of the Web application (*e.g.*, the prior installation of an operating system and a Web server) may also need to be identified. Furthermore, the operator must decide in which subnetwork (assuming there is more than one in the organization) the Web application will be available. This decision, in turn, may result in different activities to be performed onto the network's firewall and DNS server.

There are five major issues related to such *ad hoc* approach to deploy this and other requested changes. First, the steps necessary to conduct the change may be scattered among IT operators and paper procedures. These steps, along with output generated (in the form of change logs) from previous and similar changes, may be disregarded during the change design, due to the lack of a proper mechanism to express and process such information. Second, the operator must know the state of the IT infrastructure prior to the planning and deployment of the change. This is often not the case, specially for organizations having a large number of elements belonging to their infrastructure, and a complex network of relationship among them. Third, the sequence of actions designed by the operator might be incomplete (*e.g.*, due to missed dependency information) or have more actions than actually needed (*e.g.*, due to optional steps). This may translate into incomplete/faulty changes, or changes that take longer or demand more effort than necessary to be completed. Fourth, the sequence of actions may be erroneousy interpreted, if distributedly executed by several operators (*e.g.*, actions may be executed in a different order than originally envisaged, which may disrupt their execution). Fifth, knowledge is owned/acquired only by the operator(s) orquestrating the change. As a consequence, the dissemination of such knowledge to other operators may be hampered, and the organization itself may be prejudiced if the operator owning this knowledge quits from his/her position. These issues underscore the importance of a process-oriented, systematic approach (taking advantage of automation when possible) to conduct IT changes in organizations.

3.2 Conceptual Architecture

In this section we present our solution to support the different phases of the change process. In our solution, the design of an RFC and preliminary change plans represents the initial phase for the conduction of a change management process, being followed by the refinement of preliminary plans into actionable changes, the deployment of actionable workflows into the target IT infrastructure, and the discovery of change templates from execution traces. Figure 3.1 depicts the basis of our conceptual architecture for change management, highlighting its main conceptual components, personnel involved, and their interactions.

The *Change Initiator* starts a change process by interacting with the component *Change Designer* in order to specify an RFC (flow 1 in Figure 3.1). The *Configuration Management System* (CMS) provides the change initiator with an updated view of the IT infrastructure and services (flow 2) so that he/she is able to precisely identify the hardware, software elements, and services (Configuration Items – CIs) involved in the desired change. The creation of a new RFC document can be done "from scratch" or instantiated from a *request template* stored in the *Change Templates* repository (3). The intent of a request template is to specify the set of high level objectives that must be met by a change. Request templates will be created for routine and recurrent changes and will be persisted in the repository (3).

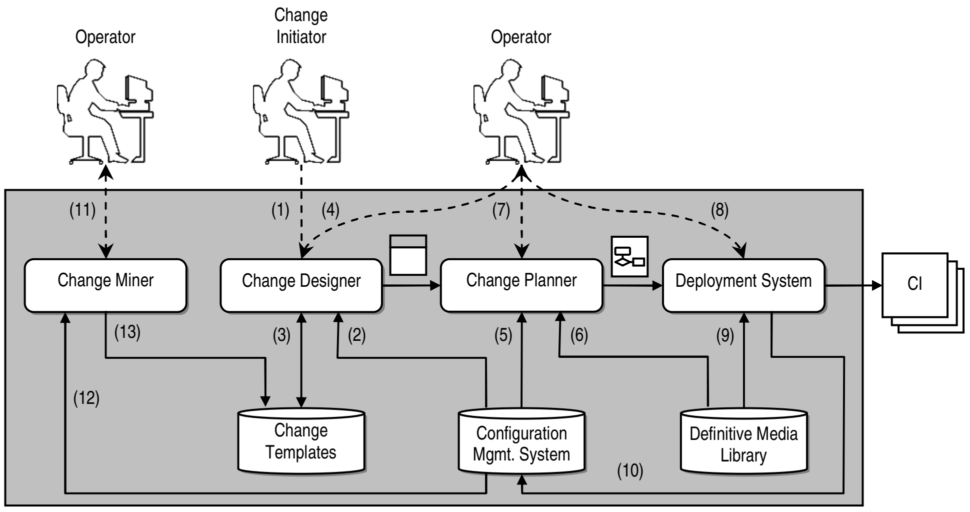


Figure 3.1: Elements of the proposed solution and interactions

After an RFC is instantiated/designed, an *Operator*, who is responsible for modeling the change procedure, interacts with the component *Change Designer* (4), now to sketch a preliminary change plan. In this stage, the operator specifies large-grained steps required to fulfill the RFC objectives, possibly taking advantage of *plan templates* available in the *Change Templates* repository (3). Plan templates encode composable workflows of change procedures and represent the knowledge gained from past experience in an IT department. Such templates are also stored in the *Change Templates* repository (3). Request and plan templates will be described in more detail in Chapter 4.

The final refinement of the preliminary change plan into an actionable workflow is then performed, without human intervention, by the component *Change Planner*. This refinement is computed using a *task refiner algorithm* (further detailed in Chapter 5), based on both factual information about the IT infrastructure (5) and information about acceptable configurations and dependencies among software packages, available in the *Definitive Media Library* (DML) (6). Thereafter, the resulting change plan can be modified by the operator as to precisely reflect his/her needs (7) and finally translated to a workflow language.

In the last step of the change process, the operator may invoke the actual execution of the workflow by interacting with a *Deployment System* (a generic abstraction for IT provisioning tools) (8). To carry out some of the activities contained in the workflow, such a system may consume software packages available in the DML (9). After executing the change plan, the deployment system is responsible for updating the CMS with the changes performed on every target CI (10).

We extend the traditional change management process by introducing the *change mining* phase, in which change templates are discovered from execution traces and persisted in the *Change Templates* repository. The operator starts the change mining by interacting with component *Change Miner* (11). In this step, he/she indicates the traces that will be consumed for analysis and subsequent template extraction. The execution traces consumed in this process – available from the CMS (12) – are typically recorded by the component *Deployment System* (10), during the execution of previous changes over the IT infrastructure. The component *Change Miner* generates plan templates that capture the high-level actions involved in the changes associated to the analyzed traces (the extraction of change templates will be further discussed in Chapter 6). Thereafter, the resulting plan templates can be modified by the operator as to precisely reflect his/her needs (11), and finally stored in the *Change Templates* repository (13). The extracted change templates may then be later (re)used in the design of future RFCs, solicited by change initiators, and of preliminary change plans, by operators, as previously described in our conceptual solution for change management.

3.3 Building Blocks of the Proposed Solution

Having presented a general view of our solution for change management, we now focus on the formalization of change-related documents and other information used by the solution. More specifically, we introduce three "building blocks", namely: the model for the management and persistence of IT information (Subsection 3.3.1), the model to design change-related documents (Subsection 3.3.2), and the notation for the modeling of activities (Subsection 3.3.3).

3.3.1 IT Infrastructure Model

We chose to base our *IT Infrastructure Model* on a subset of the Common Information Model (CIM), defined by the Distributed Management Task Force (DMTF) (DMTF, 2007). It allows the representation of computing and business entities comprising an organization, as well as the relationships among them. Figure 3.2 shows a partial view of the model. The root class of the model is *ManagedElement*. Through specialization, it is possible to represent any CI (*e.g.*, physical devices, computer and application systems, and services), as well as IT personnel. Relationships such as associations, compositions, and aggregations, most of them omitted in the figure for the sake of legibility, map the dependencies among the elements comprising the infrastructure. Some classes are described below.

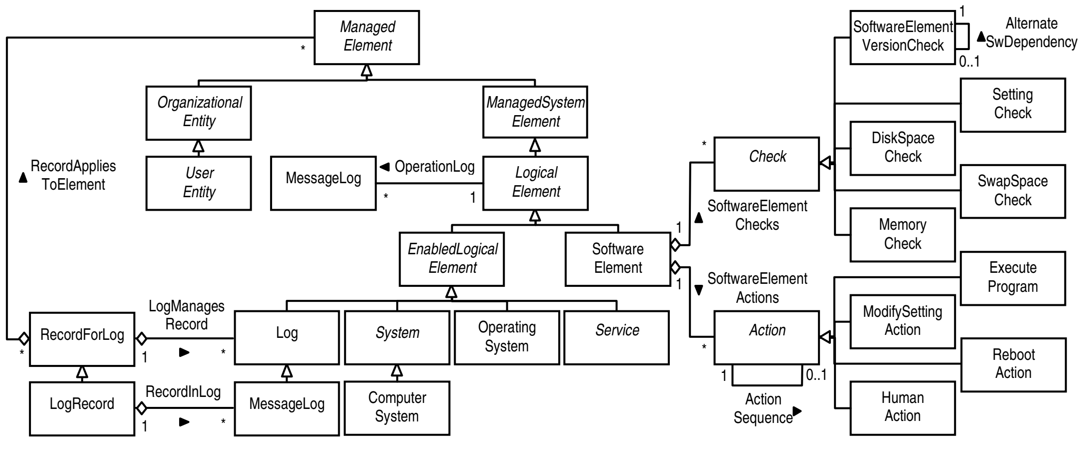


Figure 3.2: Partial view of the IT Infrastructure model

Instances of class *UserEntity* allow representing personnel belonging to the IT staff of an organization. Class *ComputerSystem* enables the representation of the several aggregations of computational entities (*e.g.*, computers or clusters) present in the managed IT infrastructure. Instances of *ApplicationSystem* allow to represent applications or software packages that support particular business functions (*e.g.*, Microsoft Word^{TM}). The several features that are provided by applications are represented by instances of *SoftwareFeature* (*e.g.*, Microsoft Word features include spell check and macro definition/execution). The software binaries (*e.g.*, system libraries or executables) that provide a specific feature are represented by instances of *SoftwareElement*. Finally, class *Service* permits to model functionalities provided by managed elements. Concrete examples of services are authentication facilities, name resolution, and mail delivery, which are provided by an authentication system, a domain name server, and a mail server, respectively.

The model also incorporates classes such as *Check* and *Action*, which represent relevant information required for managing the lifecycle of CIs (*e.g.*, software upgrade and application system installation/uninstallation). In Figure 3.2, we illustrate some specializations of classes *Check* and *Action* for the manipulation of a *SoftwareElement*.

An instance of class *Check* defines a condition to be met or characteristic required by the associated *SoftwareElement* for it to evolve to a new state (*e.g.*, *deployable*, *installable*, *executable*, or *running*). Possible checks include verification of software dependencies, available disk space and memory, and required environment settings. Each instance of class *Action*, in its turn, represents an operation of a process to change the state of the associated *SoftwareElement* (*e.g.*, from *installable* to *executable*). Examples of actions are invocation of a software installer/uninstaller, manipulation of files and directories, and modification of configuration files.

In regard to the classes that support the persistence of execution traces, *LogicalElement* enables the representation of IT provisioning systems that are employed for the implementation of changes over the managed CIs. Instances of class *MessageLog* represent, in the context of this thesis, the several changes executed over these CIs. Class *LogRecord*, in turn, holds information about tasks executed during the change process (in other words, it enables the representation of execution traces). These traces may be associated to one or more *ManagedElement*, in this case meaning that the associated element was involved in the generation of that trace.

In addition to being used to represent the current IT infrastructure, the same model is also employed to specify new hardware and software packages (along with their dependencies) that may be required throughout a change process. Our conceptual solution proposes these two uses of the same information model to be materialized, respectively, in the *Configuration Management System* and in the *Definitive Media Library*, as previously presented in Figure 3.1.

3.3.2 Requests for Change & Change Plan Model

Our conceptual solution for template-based change management includes the proposal of a *Requests for Change & Change Plan Model*, to be employed in the design of change-related documents. The model relies on both (a) guidelines presented in the ITIL Service Transition book (LACY; MACFARLANE, 2007) in regard to the change management process and (b) the workflow process definition, proposed by the Workflow Management Coalition (WfMC) (MARIN; NORIN; SHAPIRO, 2008).

Figure 3.3 presents a partial view of the model. It is structured in two connected parts. The first (whose background is highlighted in gray) enables the modeling of an RFC, while the second (with white background) provides classes for expressing the corresponding change plan. Each one is detailed below.

An RFC (represented by class *RFC* in the model) is composed of operations (class *Operation*) that indicate, in a high level of abstraction, what changes the change initiator (class *UserEntity* of the IT Infrastructure model) would like to perform. It is also important to notice that each operation has an explicit reference to the CIs over which the change operation is supposed to be executed (class *ManagedElement*). Other classes related to an RFC are: *ProblemReport* (problem that motivated the change request), *CABRecommendation* (RFC impact and resource requirement assessments), *RFCReview* (modifications to the RFC after its creation), and *RFCAuthorization* (approval of the RFC by a Change Authority).

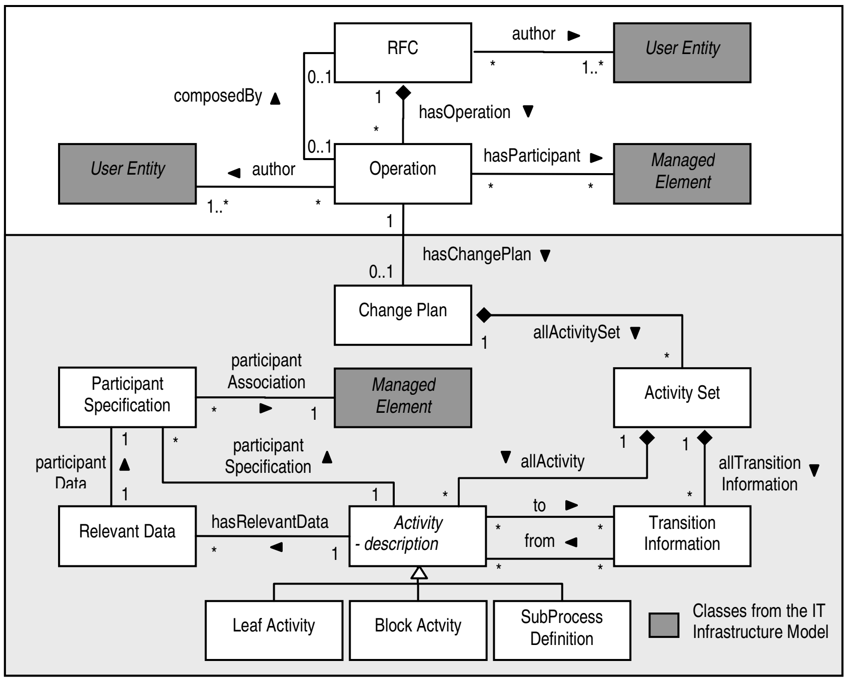


Figure 3.3: Partial view of the RFC & Change Plan model

Every operation of an RFC has an associated change plan (class *ChangePlan*) that consists of a network of activities (class *Activity*) and their relationships. These activities may represent either low-level, actionable tasks (*LeafActivity*) or large-grained ones (*BlockActivity*/*SubProcessDefinition*), being the latter subject to further levels of refinement. In general, an activity manipulates one or more CIs (class *ManagedElement*). The class *ParticipantSpecification* describes the resources which perform a given activity, for example a human, a software package, or an automated machine. Transitions between activities are represented by instances of the class *TransitionInformation*, which may be of type *branch* (with conditions or not) or *join*. Finally, the class *RelevantData* allows the specification of data consumed and produced by each activity. We would like to point out that change plans modeled using the aforementioned classes are vendor neutral and, therefore, can be easily mapped to any workflow language, such as the Business Process Execution Language (BPEL) (OASIS, 2007).

3.3.3 Activity Modeling Notation

For the refinement process to be automatically computable, the large-grained tasks must be expressed unambiguously and understood by the refinement algorithm. To address this issue, the class *Activity* introduced in the previous section presents an attribute called *description*, which is filled in with a sentence written in accordance to the *Activity Modeling Notation* (AMN), proposed in the scope of this research. Table 3.1 enumerates AMN constructions that are currently supported by our solution. In the subsequent paragraphs we present some examples of using AMN to model activities.

For illustration, consider an activity that consists of installing a Web application X on a computer Y. Its specification using AMN, according to Table 3.1, is *install SoftwareElement X at ComputerSystem Y*. In this example, X and Y are references to objects from the *Definitive Media Library* and the *Configuration Management System*, respectively.

The automated refinement of the activity *install* mentioned above is started by looking up pre-requisites (instances of class *Check*) to be met and actions (instances of *Action*) to be executed to move application X from state *installable* to state *executable* (*i.e.*, an installation procedure). This process is recursively executed (*e.g.*, for new instances of *SoftwareElement* identified as dependencies) until all pre-requisites are solved. The result of this refinement, as mentioned earlier, is a subworkflow of actions necessary to materialize the activity goal.

Consider also an activity that consists of uninstalling a Database Management System (DBMS) Z from computer W. According to AMN, its specification is *uninstall SoftwareElement Z from ComputerSystem W*. In this case, the system does not only lookup pre-requisites and actions to uninstall Z (*i.e.*, to move Z from state *executable* to state *installable*), but also consider dependency relationships within the IT infrastructure that need to be solved. For example, Z may provide database services to web pages running on a certain Web server (*i.e.*, they may need to query data from Z in order to display data to clients). As one valid approach to temporarily tackle this issue, the refined change plan could contain activities to unpublish these web pages. This process is also recursively executed, as to identify all the CIs that will be affected by the uninstallation of Z and include the necessary countermeasures in the refined workflow (the refinement of preliminary plans into actionable workflows is detailed later on in Chapter 5).

A procedure typically conducted after the installation of a Web server is making it accessible either to the local Intranet or to the entire Internet. One of the steps performed to this effect is publishing, in the network's Domain Name Server (DNS), the *hostname* of the computer in which the Web server is installed. For this particular case, the set of actions that should compose the preliminary plan is: (*i*) *add HostName www.example.com at ComputerSystem C*; (*ii*) *modify HostName www.example.com at ComputerSystem C set Type to "ADDRESS"*; and (*iii*) *modify HostName www.example.com at ComputerSystem C set IP to x.y.z.w*. In this example, C is the network's Domain Name Server (DNS) and *x.y.z.w* is the IP address of the network interface in which the Web server is listening to incoming connections.

AMN also supports the specification of activities that must be manually executed by a human operator, in order to permit that their effects can be taken into account during the refinement process. For example, an operator may wish to specify, within the preliminary plan, that more physical memory must be added to a computer B

Table 3.1: Activity modeling using AMN

Abstract Activity	*Abstract Specification*
Start a service	start Service ⟨Service⟩ at ComputerSystem ⟨Host⟩
Stop a service	stop Service ⟨Service⟩ at ComputerSystem ⟨Host⟩
Reload service settings	reload Service ⟨Service⟩ at ComputerSystem ⟨Host⟩
Restart a service	restart Service ⟨Service⟩ at ComputerSystem ⟨Host⟩
Install a software	install SoftwareElement ⟨Software⟩ at ComputerSystem ⟨Host⟩
Uninstall a software	uninstall SoftwareElement ⟨Software⟩ from ComputerSystem ⟨Host⟩
Backup software settings	backup Setting from SoftwareElement ⟨Software⟩ at ComputerSystem ⟨Host⟩ to DataFile ⟨File⟩ at ComputerSystem ⟨Backup Server⟩
Restore software settings	restore Setting for SoftwareElement ⟨Software⟩ at ComputerSystem ⟨Host⟩ from DataFile ⟨File⟩ at ComputerSystem ⟨Backup Server⟩
Copy specific setting from software	copy Setting ⟨Setting⟩ from SoftwareElement ⟨Software⟩ at ComputerSystem ⟨Host⟩ to SoftwareElement ⟨Software⟩ at ComputerSystem ⟨Host⟩
Copy settings from software	copy Setting from SoftwareElement ⟨Software⟩ at ComputerSystem ⟨Host⟩ to SoftwareElement ⟨Software⟩ at ComputerSystem ⟨Host⟩
Add a new user	add UserEntity ⟨User⟩ at ComputerSystem ⟨Host⟩
Remove an existing user	del UserEntity ⟨User⟩ from ComputerSystem ⟨Host⟩
Modify user attributes	modify UserEntity ⟨User⟩ at ComputerSystem ⟨Host⟩ set ⟨Attribute⟩ to value ⟨New Value⟩
Add a new DNS entry	add HostName ⟨Host⟩ at ComputerSystem ⟨DNS Host⟩
Delete an existing DNS entry	delete HostName ⟨Host⟩ at ComputerSystem ⟨DNS Host⟩
Modify an existing DNS entry	modify HostName ⟨Host⟩ at ComputerSystem ⟨DNS Host⟩ set ⟨Attribute⟩ to ⟨New Value⟩
Create a new file	create DataFile ⟨File⟩ in Directory ⟨Dir⟩ at ComputerSystem ⟨Host⟩
Delete an existing file	delete DataFile ⟨File⟩ in Directory ⟨Dir⟩ from ComputerSystem ⟨Host⟩
Rename file	rename DataFile ⟨File⟩ to ⟨New File⟩ in Directory ⟨Dir⟩ at ComputerSystem ⟨Host⟩
Copy File to Location	copy DataFile ⟨File⟩ in Directory ⟨Target Directory⟩ at ComputerSystem ⟨Host⟩ to Directory ⟨Target Directory⟩ at ComputerSystem ⟨Host⟩
Move File to Location	move DataFile ⟨File⟩ in Directory ⟨Target Directory⟩ at ComputerSystem ⟨Host⟩ to Directory ⟨Target Directory⟩ at ComputerSystem ⟨Host⟩
Modify file attributes	modify DataFile ⟨File⟩ in Directory ⟨Dir⟩ set ⟨Attribute⟩ to value ⟨New Value⟩
Configure a generic software/device	configure ManagedSystemElement ⟨Element⟩ of type ⟨Element Type⟩ at ComputerSystem ⟨ComputerSystem⟩ using Setting ⟨Setting⟩
Modify attributes of a generic software/device	modify ManagedSystemElement ⟨Element⟩ of type ⟨Element Type⟩ at ComputerSystem ⟨ComputerSystem⟩ set ⟨Attribute⟩ to ⟨New Value⟩
Dump database schema to file	dump schema from DatabaseSystem ⟨Database⟩ at ComputerSystem ⟨Host⟩ to DataFile ⟨File⟩ in Directory ⟨Dir⟩ at ComputerSystem ⟨Host⟩
Load database schema to file	load schema from DataFile ⟨File⟩ in Directory ⟨Dir⟩ at ComputerSystem ⟨Host⟩ to DatabaseSystem ⟨Database⟩ at ComputerSystem ⟨Host⟩

prior to the installation of a new software package. According to AMN, the specification of this activity is *modify ManagedSystemElement A of type OperatingSystem at ComputerSystem B set TotalVisibleMemorySize to 2048M*. By computing the effect of this manual activity, the refinement process will be aware that computer *B* will have 2 GB of physical memory available for operating system *A* (according to the CIM model, each operating system has its own memory settings).

4 DESIGN OF PRELIMINARY PLANS THROUGH CHANGE TEMPLATES

The conceptual solution presented in Chapter 3 is the core basis upon which the notion of IT change templates is built. In this chapter we explore it in more detail. Subsection 4.1 characterizes request and plan templates. The association between such templates, during the design of changes, is explained in Subsection 4.2. Finally, Subsection 4.3 discusses composition, specialization, and generalization of templates.

4.1 Request & Plan Templates

In the context of this work, we define templates as parameterized building blocks that can be reused in the design of changes similar to previously executed ones (CORDEIRO et al., 2008). Explored in different levels of the change process, they are categorized as request or plan templates. Each one is detailed below.

A *request template*, defined by means of the upper part of the model illustrated in Figure 3.3, defines a set of prototypical operations that need to be performed on the IT infrastructure. The person producing such a template (*e.g.*, change initiator or operator) is free to specify which attributes/references will have fixed values and which will be left to be provided during the instantiation of a new RFC. As an example, Figure 4.1 (background highlighted in gray) presents a request template to deploy a hosting service in a dedicated server. This template is composed of two operations: *Build Dedicated Web Hosting Service* and *Lease Dedicated Data Circuit*. In the case of the first operation, the references *what* to be installed and *where* are defined to be of type *WebApplication* and *ComputerSystem*, respectively, thus specifying the type of CIs that may be affected.

The request template just described could be used, for example, to instantiate two distinct RFCs: one to install a J2EE Web application on a RISC computer and one to install an ASP.NET Web application on a 64-bit computer. For that to be possible, J2EE and ASP.NET apps should be present in the DML (as *WebApplication* objects), while RISC and 64-bit computers should appear (as *ComputerSystem* objects) in the CMS.

Going down to the level of change plans, *plan templates* have a central role in the process of promoting the reuse of the experience accumulated by operators. Specified using the classes illustrated in the lower part of Figure 3.3, these templates comprise steps necessary to materialize recurrent changes. The large-grained activities of a plan template are typically loosely coupled, learned and tuned after a multitude of attempts to implement the corresponding changes successfully. For this reason, they

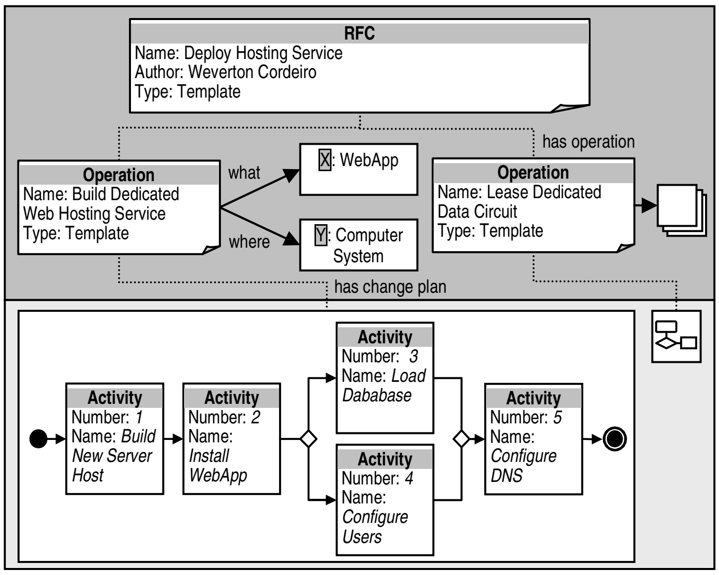


Figure 4.1: Examples of request and plan templates

are not automatically computable based on dependency information.

Figure 4.1 (background in white) depicts an example of a plan template for the installation of a new server, from scratch, and a Web application on top of it. *Activity 1* consists of building the new server into *ComputerSystem Y*. *Activity 2* installs *WebApplication X* into the newly built server. Next, two parallel tasks take place: *Activity 3*, which loads the default database for use by the *WebApplication X*, and *Activity 4*, which configures the users that will have remote access to the application. Finally, *Activity 5* comprises the instructions for publishing the URL of the Web application in the network's DNS server. This activity is defined to be executed in a final step because clients must not have access to the Web application until it is fully installed and operational.

4.2 Association Between Request and Plan Templates

The *association* between templates enables the creation of change documents as a combination of one request and one or more plan templates that may satisfy the demand of the RFC operation(s). The motivation for such combination is twofold. First, different changes specified in the level of request templates may be materialized by the same implementation procedure, hence the same plan template. Second, there may be more than one valid plan template that materializes the same operation of a request template.

The association of templates can be done in one of two ways: (1) gradually, after the instantiation of an RFC (taking advantage of a request template or not); or (2) at any point in time before an RFC instantiation. Regardless of the moment of such associations, they are achieved through the establishment of a relationship between the classes *Operation* (at the RFC level) and *ChangePlan* (at the change plan level).

When associating templates, the operator must specify the mapping between attributes/references from the RFC/request template and variables of the binded plan templates. It is expected that all attributes and/or references of the former are linked to one (or more) variable(s) of the latter. Variables of the plan template not set by this process must be assigned "*values*" by the operator in order to produce a consistent and complete preliminary change plan. In the example illustrated in Figure 4.1, one can observe the association between a request and a plan template. One of the operations, *Build Dedicated Web Hosting Service*, has been associated to the plan template depicted in the lower part of the figure (explained in the previous subsection).

We highlight, in the aforementioned association, the binding of the variables of *Activities 2* and *5* to the variables of the operation *Build Dedicated Web Hosting Service*. For *Activity 2*, one of the participant CIs is the Web application to be installed. In this case, the Web application is already present in the associated operation (variable named X, which is of type *WebApplication*). When the operator assigns a value for variable X in the operation *Build Dedicated Web Hosting Service*, the assigned CI will be automatically indicated as a participant CI in *Activity 2*. For *Activity 5*, however, one of the participant CIs, the DNS server, is not listed as a variable of the associated operation. Therefore, during the instantiation process, the operator will need to identify, among the servers registered in the *Configuration Management System*, what is the DNS server to be affected by *Activity 5*.

There are several scenarios in which association between templates is important. Looking at association between templates from a top-down perspective, consider the request template *Build Dedicated Web Hosting Service*. From the *Change Templates* repository, the operator may identify two distinct plan templates that materialize the change expressed in the request template: *Build Hosting Service Using GNU/Linux* and *Build Hosting Service Using Microsoft Windows XP*. The former contains the high level steps to install a Web Hosting Service in a GNU/Linux environment (by using Apache and/or Tomcat Web servers, among others), while the latter comprises steps to do the same, however in a Microsoft Windows environment (by using Internet Information Service or Apache for Windows, for example).

Now looking at association from a bottom-up perspective, consider the plan template *Migrate System to a New Host*. It comprises the detailed steps to migrate data files, installed software, and system and software settings, from a given computer to another one. This plan template could be used to materialize the following request templates: *Improve Quality of Service for Customer's Application* and *Upgrade Web Server Hardware*. Considering that the operator is aware that the best choice to accomplish both requests is migrating the system to a new host having higher computational capabilities, he/she could take advantage of the previously mentioned plan template to meet the objectives of these request templates.

4.3 Composition, Generalization, and Specialization of Templates

Reuse of knowledge in the context of IT changes is facilitated through the use of template *composition*, *specialization*, and *generalization*. These techniques are introduced next, considering both request and plan templates.

Template *composition* allows complex changes to be designed based on the experience formalized and preserved in previous processes. The basic principle is to use existing templates as "building blocks" in the design of new change documents. From the perspective of request templates, composition is possible, for example, by grouping operations that have been specified elsewhere in different request templates. This type of composition is supported by the association between the classes *RFC* and *Operation* of the model presented in Subsection 3.3.2.

An example of composition in this level is the use of the templates *Upgrade Web Server Hardware* and *Install New Leased Data Line Circuit*. Since one of their objectives is improving the computational environment for software having high computational, storage, memory, and bandwidth demands, they could be used to compose the request template *Improve Quality of Service for Customer's Application*, mentioned in the previous subsection.

It is in the context of plan templates, though, that composition can be better taken advantage of. In this case, large-grained activities found in a plan template can be materialized by other plan templates, possibly forming a hierarchy of *nested templates*. Figure 4.2 illustrates a template for installing an operating system that is nested to *Activity 1*, *Build New Server Host*, of the plan template presented in Figure 4.1. Such nesting is made possible by using the class *SubProcessDefinition*.

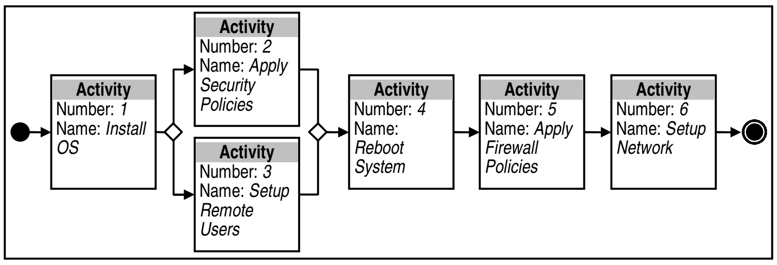


Figure 4.2: A template for installing an operating system

Composition in the level of plan templates also enables the modeling of changes by joining together (pieces of) templates already present in the *Change Templates* repository. This approach permits specifying complex change plans in less time than would be required to design the whole preliminary plan from scratch. For example, the plan template presented in Figure 4.2 may be built by composing several other plan templates. As shown in Figure 4.3, *Activities 5* and *6* may be materialized by other plan templates, which describe, in a more detailed perspective, the steps involved in the configuration of the system's firewall and the configuration of the network, respectively.

Specialization enables the creation/modification of change templates to address non-ordinary changes, whose activities usually deviates slightly from those found in typical procedures. It also permits the operator to think globally about a problem, model a template to solve it, and then adapt the composed plan to the specific needs or nuances of each target environment. For request templates, specialization may be useful, for instance, to both extend an existing one with new operations and/or restrict the scope of the CIs that may be affected by them. For a concrete example, consider the request template shown in Figure 4.1. It could be specialized so that the affected CI, labeled as *what* in operation *Build Dedicated Web Hosting Service*, was restricted to J2EE Web applications, instead of any Web application, as originally defined.

One can also use specialization as a surrogate to model customized plan templates. For example, a template that dictates how an operating system should be installed within an organization may have different specializations to address the specific needs of each department. In this example, class *BlockActivity* could be em-

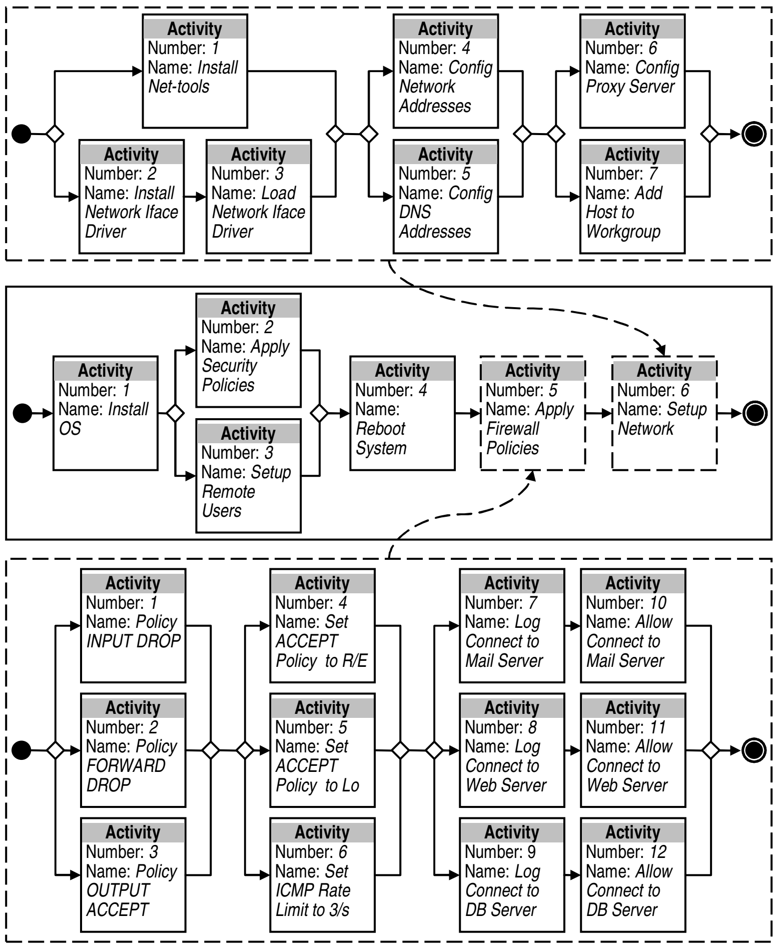

Activity
Number: 1
Name: Install Net-tools
Activity
Number: 2
Name: Install Network Iface Driver
Activity
Number: 3
Name: Load Network Iface Driver
Activity
Number: 4
Name: Config Network Addresses
Activity
Number: 5
Name: Config DNS Addresses
Activity
Number: 6
Name: Config Proxy Server
Activity
Number: 7
Name: Add Host to Workgroup
Activity
Number: 1
Name: Install OS
Activity
Number: 2
Name: Apply Security Policies
Activity
Number: 3
Name: Setup Remote Users
Activity
Number: 4
Name: Reboot System
Activity
Number: 5
Name: Apply Firewall Policies
Activity
Number: 6
Name: Setup Network
Activity
Number: 1
Name: Policy INPUT DROP
Activity
Number: 2
Name: Policy FORWARD DROP
Activity
Number: 3
Name: Policy OUTPUT ACCEPT
Activity
Number: 4
Name: Set ACCEPT Policy to R/E
Activity
Number: 5
Name: Set ACCEPT Policy to Lo
Activity
Number: 6
Name: Set ICMP Rate Limit to 3/s
Activity
Number: 7
Name: Log Connect to Mail Server
Activity
Number: 8
Name: Log Connect to Web Server
Activity
Number: 9
Name: Log Connect to DB Server
Activity
Number: 10
Name: Allow Connect to Mail Server
Activity
Number: 11
Name: Allow Connect to Web Server
Activity
Number: 12
Name: Allow Connect to DB Server

Figure 4.3: Illustration of *composition* in the level of plan templates

ployed to refine higher level, standard actions into more detailed, lower level steps, in order to accommodate those specific requirements.

The specialization of plan templates may alternatively serve the purpose of allowing the operator to guide/influence the refinement of change plans into actionable workflows. For example, consider the plan template depicted in Figure 4.1. Being aware that the Web application requires a Data Base Management System (DBMS), the operator may indicate, in the design of the template, which DBMS should be installed, instead of leaving the decision to the *Change Planner*.

Finally, *generalization* lets the operator to start thinking of a specific template (either request or plan) and, after that, generalize it in order to make it suitable for more general contexts. For example, an operator may design a template to perform some change considering the IT scenario of a particular department. The essence of the change may be later captured and applied to other departments as well, by means of generalization.

For request templates, the generalization could be achieved, for example, by changing the affected CI of an *install* operation, from *WebApplication* to simply *SoftwareElement*. In the case of plan templates, finer-grained activities, for example defined within a *BlockActivity*, could be replaced with a more general, larger-grained activity. As a practical example, suppose than an operator has designed a change template for the installation of an authentication system for Department X of an

Organization Y. Through generalization, this template could be adapted in order to become the standard installation of the same system in every department of the same organization.

5 FROM PRELIMINARY PLANS TO ACTIONABLE WORKFLOWS THROUGH AUTOMATED PLANNING

As previously mentioned in Chapter 3, the refinement of preliminary change plans into actionable workflows is computed by means of a *task refiner* algorithm. It generates actionable change plans that are aware of the restrictions imposed by the IT environment (*e.g.*, disk space and memory constraints). As a consequence, the resulting change plans are less prone to be prematurely aborted (*e.g.*, due to lack of resources), reducing the chances of incidents and service-delivery disruption caused by failed changes (CORDEIRO et al., 2008).

The key concepts of the algorithm for the constraint-aware refinement of change plans proposed in the scope of this work are the *backtracking technique* (CORMEN et al., 2001) and the *snapshots* of the IT infrastructure. Backtracking is a sistematic method to iterate through all possible configurations in the space of refinements for the activities that compose a preliminary change plan. Snapshots, in turn, represent the differences between the current state of the IT infrastructure and the state it would reach after the execution of the activities being refined. These differences include, for example, newly installed (or removed) software, disk space and memory consumed (or freed), modified settings, and created (or deleted) files and directories. Used in conjunction with the backtracking technique, snapshots prune the search space, indicating which of the possible refinements violate IT resource constraints. Please refer to a previous work (CORDEIRO et al., 2008) for an in-depth analysis on the dynamics of snapshots.

Algorithm 1 presents the pseudo-code that implements the refinement process. It receives three parameters as input: C, I, and R, where C is the preliminary change plan; I represents the state of the IT infrastructure as in the instant in which the preliminary plan C is submitted for refinement; and R represents the *Definitive Media Library* (DML). The output of the refinement process is an actionable workflow that meets the resource constraints imposed by the target environment.

As described in Algorithm 1, function *Refine* initiates the refinement process by copying all activities from the preliminary plan C to A – the set of activities that require further refinement. Subsequently, it creates an initial (and empty) snapshot of the IT infrastructure, s_0. Despite not describing any differences from the current state of the IT infrastructure, this snapshot serves as basis for the recursive refinement that will be conducted afterwards.

The actual refinement of the preliminary plan C is performed by function *Backtrack*. Upon each invokation of this funcion, it verifies if the set of activities that require further refinement is empty (or not). An empty set A received as parameter

```
ChangePlan Refine($C$, $I$, $R$);
begin
    $A \leftarrow$ set of activities from $C$;
    $s_0 \leftarrow$ initial snapshot of $I$;
    Backtrack($C$, $I$, $R$, $A$, $s_0$);
    if all activities in $C$ are refined, given $I$ then
        return ChangePlan $C$;
    else
        return false;
    end
end
ChangePlan Backtrack($C$, $I$, $R$, $A$, $s_i$);
begin
    if $A$ is empty then
        return ChangePlan $C$;
    else
        extract an activity $a_i$ from $A$;
        if all dependencies for $a_i$ are satisfied, given $I$, $R$, and $s_i$ then
            Backtrack($C$, $I$, $R$, $A$, $s_i$);
        else
            compute candidate activity dependency groups $Y_j$ to satisfy $a_i$;
            foreach set of activities $Y_j$ do
                add activities in $Y_j$ to the change plan $C$;
                add activities in $Y_j$ to the set of activities $A$;
                $s_{i+1} \leftarrow$ new snapshot of $I$, given $I$, $R$, and $s_i$;
                Backtrack($C$, $I$, $R$, $A$, $s_{i+1}$);
            end
        end
    end
end
```
Algorithm 1: Runtime constraint-aware refinement of change plans

indicates that all activities originally present in C (and the ones created and added to this set during the refinement process) have been already processed. If empty, the recursive refinement returns, and the plan C received as parameter is passed back to function *Refine* for further verification.

In case A is not empty, an activity a_i is extracted. Subsequently, it is verified whether a_i has computed dependencies. An activity a is said to have computable dependencies if: (*i*) the *Configuration Item* (CI) modified by a has instances of checks (*SoftwareElementChecks*) mapped in the Definitive Media Library and/or relationships in the IT repository (*e.g.*, shutting down *Service*$_1$ requires shutting down *Service*$_2$ and bringing up *Service*$_3$), and (*ii*) the previously described dependencies (or checks) are not yet fulfilled considering snapshot s_i and the current state of the IT infrastructure I. For example, suppose that a_i is expressed, according to AMN, as *install SoftwareElement WebApplication at ComputerSystem Server*$_1$. Suppose also that from R (the Definitive Media Library) *WebApplication* is known to require the prior installation of a Web server and a Database Management System (DBMS). In this case, there are two dependencies that need to be fulfilled to make a_i executable. If all the dependencies for executing a_i are already satisfied (considering s_i and I), the refinement continues recursively, now to process another activity from A (lines 18-19).

If a_i has dependencies not yet satisfied, all the groups Y of candidate activities to satisfy them are computed, given s_i, I, and R (line 21). Considering the previous example, suppose that *WebApplication* requires one of two different Web servers (*Apache* or *Internet Information Server*), and one of two different database servers (*MySQL* or *SQL Server 2005*). Suppose also that *SQL Server 2005* is known to require more disk space than $Server_1$ has available. The processing of this information, obtained from the DML and represented through instances of class *Checks*, yields the generation of the following groups of candidate activities: $Y_1 = \{$ *Install Apache*, *Install MySQL* $\}$ and $Y_2 = \{$ *Install IIS*, *Install MySQL* $\}$. Each of these activities represent a *BlockActivity* derived from the set of actions (instances of class *Action* from the model of Figure 3.2) necessary to install their respective software. The reader may note that *Install SQL Server 2005* is not regarded as a candidate since the requirements for the installation of *SQL Server 2005* violates the disk space constraint imposed by $Server_1$.

After the groups of candidate activities are computed, *Backtrack* searches for a group Y_j that leads to a refined plan. Although more than one Y_j may lead to a solution, the first Y_j whose *recursive* dependencies (*i.e.*, the dependencies of the activities in Y_j) satisfy the imposed restrictions will compose the refined plan. As described in Algorithm 1 (lines 22-27), for each group Y_j, the activities in it are added to the plan C and to A (since these activities may also require further refinement). Subsequently, the effects of executing the activities in Y_j are persisted at snapshot s_{i+1}. Note that s_{i+1} holds the effects of executing the activities in Y_j plus the effects already recorded in s_i. Finally, *Backtrack* is invoked recursively, now to process another activity present in A.

It is important to mention that, if activity a_i has dependencies to be satisfied, but none of these dependencies are executable (*e.g.*, due to violation of constraints), no group of candidate dependencies is created (line 21). In this case, the recursive processing fails, indicating that one group of candidate dependencies Y_j selected in a previous recursion does not lead to a refined plan. Therefore, another group Y_{i+1} is tested (if there is any). This process continues until a refined plan that meets resource constraints is found, or all the space of refinements for a given plan is explored.

As previously mentioned, once all activities added to A are processed, the plan C is returned to function *Refine*. This plan will be given as a solution to the operator, if refined (line 6). We consider a change plan C as refined if and only if the dependencies of all activities composing the plan are satisfied either by any other activity in C or by the current state of the IT infrastructure. In case the plan C is not refined, the operator will receive a negative feedback (line 9). This feedback will mean that an executable workflow (for the preliminary plan C submitted) could not be achieved. Having this feedback, the operator could reformulate and resubmit the preliminary plan, therefore starting the refinement process over again.

6 FROM EXECUTION TRACES TO CHANGE TEMPLATES THROUGH TEMPLATE MINING

The notion of change templates introduced in Chapter 4 forms the fundamental basis for the template mining mechanism presented in this thesis. Starting from past execution traces, the proposed mechanism extracts the workflows behind the implemented changes and, afterwards, transforms them into change templates. In a first iteration at this problem, it was focused the extraction of templates that comprise simple execution routing structures, i.e., composed of and/or branches/joins (CORDEIRO et al., 2009). In the following sections, the template mining mechanism is depicted in more detail. Section 6.1 describes, in more detail, the intermediate, grained-level phases of the change template discovery process. Section 6.2 presents how actionable change plans are extracted from execution traces. Finally, Section 6.3 details how the extracted plans are transformed into change templates.

6.1 Template Discovery Process Overview

The mechanism for template mining, encapsulated in the component *Change Miner* (illustrated in Figure 3.1), comprises three distinct, well-defined steps. These steps are materialized by subcomponents *Workflow Miner*, *Decision Miner*, and *Template Converter*, illustrated in Figure 6.1. The relationship between these subcomponents and with external entities is described next.

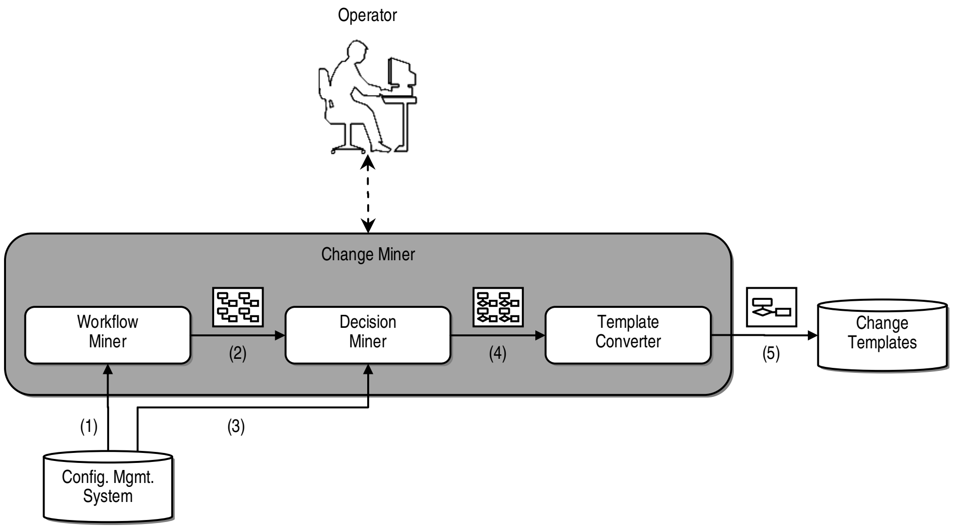


Figure 6.1: Subcomponents of the change mining mechamism

Once the change mining process is started by the operator (as described in Section 3.2), the component *Change Miner* triggers the execution of the subcomponent *Workflow Miner*. The purpose of this subcomponent is to extract workflows that reflect the behavior of similar changes executed in the past. To this end, it consumes, from IT-related repositories (flow 1 in Figure 6.1), execution traces generated by IT provisioning tools. By analyzing these traces, the *Workflow Miner* is able to discover the low-level activities (and their relationship) that comprised the past changes. This subcomponent produces typical workflows of activities having *and/or branches/joins*.

The workflows generated in the previous step are then processed by the subcomponent *Decision Miner* (flow 2). Also exploring historical information present in the *Configuration Management System* (3), this subcomponent identifies how data has influenced the execution routing. Subsequently, it populates the *and/or branches/joins* present in the workflows with the conditions that needed to be satisfied for the transitions to take place. Finally, the resulting workflows are consumed by the subcomponent *Template Converter* (4). This subcomponent will abstract the change procedures described in the workflows as to enable their applicability in several, diverse change contexts. Thereafter, as previously mentioned in Section 3.2, the resulting templates may be modified by the operator and stored in the *Change Templates* repository (5) for future (re)use.

6.2 Discovering Change Processes from Logs

As presented in the previous section, subcomponent *Workflow Miner* is responsible for extracting the actionable change plans (or actionable workflows) executed for the implementation of past changes. To materialize the conceptual functionality of this subcomponent, we have tailored some concepts from already established process mining techniques (MARUSTER et al., 2002; MEDEIROS et al., 2004) to the context of change management.

There are simplifying assumptions adopted by process mining techniques which are also applicable in our work. First, they assume that each event present in an execution trace refers to an activity (*i.e.*, a well defined step in the workflow). Second, each event refers to a case (*i.e.*, a workflow instance). Finally, they assume that events present in the trace are totally ordered. Researchers in the field of process mining argue that, in fact, any information system using transactional systems will offer this information in some form (MARUSTER et al., 2002). In the context of our work, activity maps to change action (the smallest unit of a change present in an RFC document), cases represent instances of execution of RFCs onto the target IT environment, and IT provisioning tools are examples of transactional systems.

In order to enable the applicability of process mining concepts in the field of change management, we denote a_i $0 \leq i \leq n$ as being a change action (smallest unit of a change) performed over the managed IT infrastructure. Sequences of change actions are represented by change traces. Finally, a change log L is a set of all traces related to a same process. We assume that L may be created by grouping together, from the log repository, traces that share a same attribute value (*e.g.*, the same change ID). To facilitate the understanding of the process, we illustrate each step considering the example of change log presented in Table 6.1. In this example, there are six change traces, each composed by nine executed tasks. The process for the

Table 6.1: Example of execution trace

Change trace number	Executed change actions
Trace 1	$a_1, a_3, a_5, a_6, a_4, a_9$
Trace 2	$a_1, a_3, a_4, a_5, a_6, a_9$
Trace 3	$a_2, a_5, a_6, a_3, a_4, a_9$
Trace 4	$a_1, a_7, a_3, a_8, a_4, a_9$
Trace 5	$a_2, a_3, a_7, a_4, a_8, a_9$
Trace 6	$a_2, a_3, a_5, a_4, a_6, a_9$

extraction of change plans from execution logs is comprised of four high-level steps, which are described below.

1) *Creation of the Dependency/Frequency Table*: The Dependency/Frequency (D/F) table (MARUSTER et al., 2002) contains information about the frequency of occurrence of events registered in the change log (*e.g.*, number of executions of each change action, and number of succession relationships that occur between them). The information contained in the D/F table are (*i*) the identifier of task a_i and a_j, (*ii*) the overall frequency of task a_i, (*iii*) the overall frequency of task a_j, (*iv*) the frequency in which task a_i directly succeeds another task a_j ($a_j > ai$), (*v*) the frequency in which task a_j directly succeeds another task a_i ($a_i > a_j$), (*vi*) the frequency in which a_i directly or indirectly succeeds another task a_j, but before the next occurrence of a_j ($a_j \ggg a_i$), and (*vii*) the frequency in which a_j directly or indirectly succeeds another task a_i, but before the next occurrence of a_j ($a_i \ggg a_j$). Table 6.2 shows an excerpt of D/F table for the change log illustrated in Table 6.1. We highlight the number of times in which $a_3 > a_4$ holds, suggesting that a_3 and a_4 are likely to be connected in the workflow.

Table 6.2: Dependency/Frequency table

a_i a_j	# a_i	# a_j	(a_j >a_i)	(a_i >a_j)	($a_j \ggg a_i$)	($a_i \ggg a_j$)
a_3 a_1	500	404	236	0	404	0
a_3 a_2	500	96	56	0	96	0
a_3 a_3	500	500	0	0	0	0
a_3 a_4	500	500	0	160	0	500
a_3 a_5	500	339	114	147	0	339
a_3 a_6	500	339	27	86	0	339

2) *Calculation of the Local Metric, Global Metric, and Causality Metric*: The Local Metric (LM) expresses the tendency of succession relation by comparing the magnitude of ($a_i > a_j$) versus ($a_j > a_i$). The Global Metric (GM), in contrast, determines the likelihood of succession between two change actions a_i and a_j considering the overall frequencies of these actions. Finally, the Causality Metric (CM) indicates the chance that action a_i causes the occurrence of action a_j. Table 6.3 presents the values for metrics LM, GM, and CM computed for the pairs of activities presented in the first field of Table 6.2. For example, the high values of LM, GM, and CM for the pair (a_3, a_4) reinforce the observation done for the data in Table 6.2.

Table 6.3: Values for metrics LM, GM, and CM

a_i a_j	LM	GM	CM
a_3 a_1	0	0	0
a_3 a_2	0	0	0
a_3 a_3	0	0	0
a_3 a_4	0.98	0.32	1.00
a_3 a_5	0.50	0.09	0.32
a_3 a_6	0.67	0.17	0.13

3) *Computation of the Existence of Direct Succession Between Tasks*: In this step, the three metrics LM, GM, and CM, computed in the previous step, are combined using a *Logistic Regression Model* (MEDEIROS et al., 2004). The output of this combination is the probability π that two events a_i and a_j, present in the log, are in a direct succession relationship.

4) *Generation of the Actionable Change Plan*: This last step consists in identifying and/or branches/joins between tasks, considering the succession relationship information present in the D/F table. After identified, they are explicitly incorporated into the workflow. Finally, the activities in the discovered plan are associated to the Configuration Items they affect. Figure 6.2 illustrates the change plan extracted from the change log partially illustrated in Table 6.1.

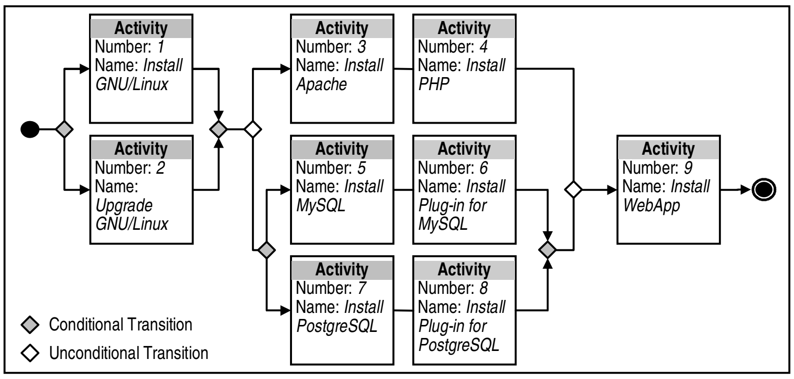


Figure 6.2: Change plan extracted from an execution trace

6.3 Converting Discovered Change Processes into Templates

Our mechanism for the conversion of an actionable change plan into a change template is materialized by a conceptual algorithm, illustrated in Algorithm 2. The conceptual algorithm, denoted by *ExtractTemplate*, takes as input the change plan C to be converted, the current state of the IT infrastructure I, and the *Definitive Media Library* (DML) R. The output is a change template T that comprises the high-level steps necessary to implement the same change described in C in a broader range of scenarios.

Figure 6.3 illustrates the functioning of *ExtractTemplate* using a simplified example. The change plan C presented in the figure – extracted using the change

```
1  PlanTemplate ExtractTemplate($C$, $I$, $R$);
2  begin
3      $A \leftarrow$ set of activities from $C$;
4      $S \leftarrow$ empty set of activities;
5      $T \leftarrow$ empty plan template;
6      foreach activity $a_i \in A$ do
7          $S \leftarrow S \cup \{$set of dependencies of $a_i$, given $I$ and $R\}$;
8      end
9      $T \leftarrow A - S$;
10     $B \leftarrow$ set of activities $a$ from the change template $T$;
11     foreach $a_i \in B$ do
12         foreach $b_j \in B$ do
13             if $a_i \neq b_j$ and there is a path from $a_i$ to $b_j$ in $C$ then
14                 cond $\leftarrow$ false logical expression;
15                 $P \leftarrow \{$set of paths $P_k$ that connect $a_i$ to $b_j$ in $C\}$;
16                 foreach transition $t_l$ arriving at $b_j$ do
17                     if $\exists P_k \in P \mid t_l \in P_k$ then
18                         cond $\leftarrow$ cond . (or (condition of transition $t_l$));
19                     end
20                 end
21                 connect $a_i$ to $b_j$ in $T$ using the logical expression cond;
22             end
23         end
24     end
25     $D \leftarrow$ empty set of variables $v$ of managed elements;
26     foreach activity $a_i \in B$ do
27         transform all managed elements $e$ affected by $a_i$ into variables $v$;
28         $D \leftarrow D \cup \{$list of variables $v$ present in activity $a_i\}$;
29     end
30     add $D$ to the list of variables of the change template $T$;
31     return PlanTemplate $T$;
32 end
```

Algorithm 2: Conceptual algorithm for converting change plans into templates

mining technique presented in the previous section – comprises the steps necessary to install a *Web Application* (*WebApp*).

As a first step towards the conversion of an actionable workflow into a change template, the set of activities that compose the change plan C is copied to A (line 3 in Algorithm 2). Next, an empty set of activities S (line 4), and an empty template T (line 5) are created. The newly created template T, in this case, will hold the result of the conversion process.

Subsequently, for each activity a_i that belongs to A (line 6), *ExtractTemplate* computes the set of dependencies that must be satisfied prior to its execution, considering information about the target IT infrastructure I and dependencies between software packages available in R. The result of this computation is added to S (line 7). At this point, S is composed only of activities that satisfy any dependency of any other activities of the change plan C. In the example illustrated in Figure 6.3,

S will be composed of activities *Install WebSrv*, *Install DBServer*, and *LibX*. In this case, *WebSrv* and *DBServer* are software that must be installed prior to the installation of *WebApp* (according to *WebApp*'s *SwChecks*, present in the DML). *LibX*, in turn, is a software dependency for both *WebServer* and *DBServer* (as both require its prior installation, according to their *SwChecks*, also shown in the DML). Since activity *Configure DNS* is not an explicitly dependency found in the DML, it is not added to S.

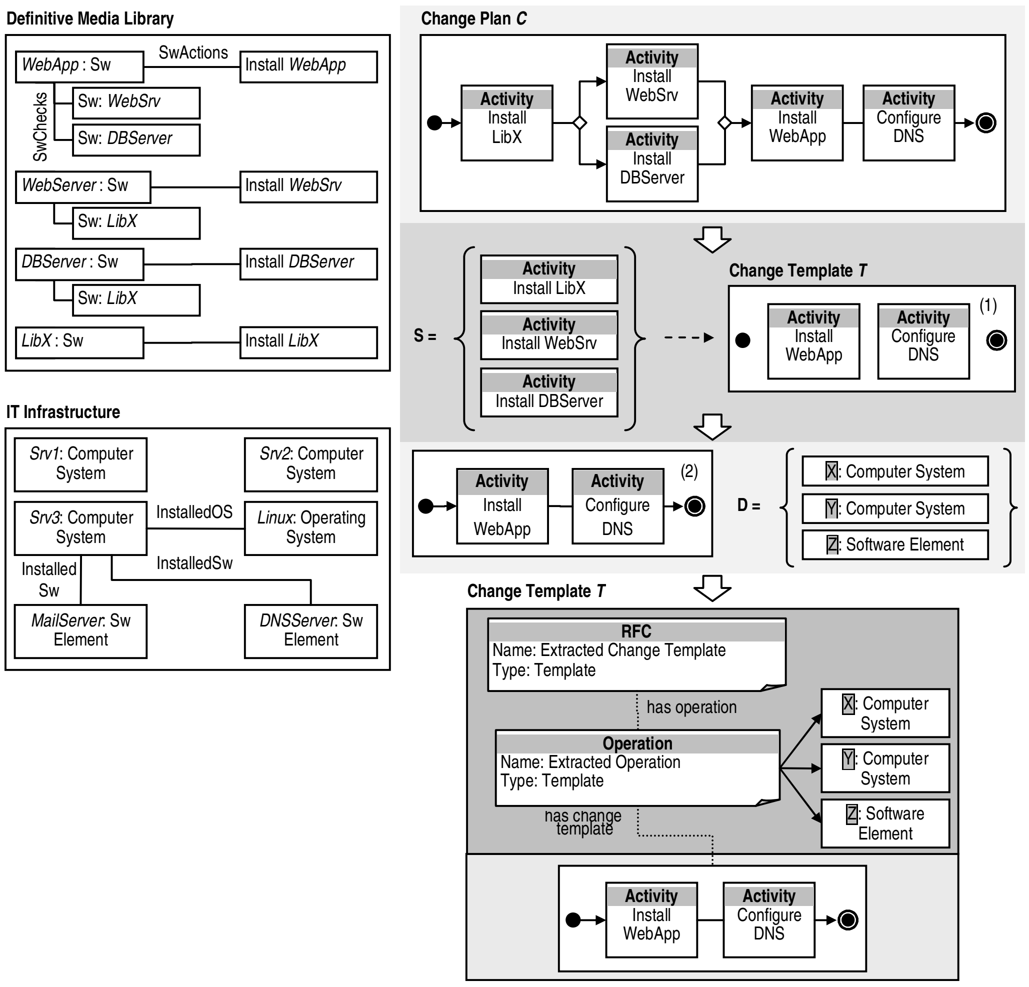


Figure 6.3: Conversion of a change plan into a plan template

After processing the activities in A, the computed dependencies (in S) are excluded from A, and the remaining ones are added to T (line 9). At this point, T is composed only of activities that cannot be deduced through automated analysis of dependencies between elements from IT-related repositories. In the example of Figure 6.3, the template T (box 1) is composed of activities *Install WebApp* and *Configure DNS*.

The activities that have been just added to T do not have any relationships to one another (*i.e.*, there are no conditional/unconditional transitions between them). This information must be obtained by observing how activities are linked to each other in the original change plan C. In order to compute this information, the set of activities contained in template T is copied to B (line 10). Subsequently, for each pair of ordered activities $(a_i,\ b_j) \in B$ (lines 11-12), if there is at least a path of activities starting in a_i and ending in b_j in the original change plan C (line 13), the following steps are performed. First, *cond* is initialized with a *false* logical expression (line 14). Subsequently, all different paths of activities P_k from a_i to b_j are stored in P (line 15) (P_k, in this case, is an ordered set of transitions that form a path of activities from a_i to b_j). Then, for each transition t_l that arrives at activity b_i (line 16), if this transition belongs to any of the paths in P (line 17), the condition of this transition is used to build a logical expression, which will be concatenated to *cond* (line 18). After all transitions that arrive to b_j are processed, a_i is connected to b_j in the change template T (line 21), using as transition condition the logical expression stored in *cond*. In Figure 6.3, the template T shown in box 1 was modified by the addition of the transitions between the activities that compose it, yielding the one presented in box 2.

Once the transitions between activities that compose the change template T are created, *ExtractTemplate* processes all managed elements affected by activities in T, converting references to them into template variables (which were previously described in Section 4.2). As a first step towards the processing of these variables, D is declared as a set of variables v of managed elements, *i.e.*, pointers to elements belonging to the IT infrastructure (line 25). Next, for each activity a_i of template T present in B (line 26), the managed elements affected by these activities are abstracted, *i.e.*, the activity will no longer be associated to a concrete managed element. Instead, it will reference variables v having the same class of the affected managed elements e (line 27). The variables resulted from this abstraction will be added to D (line 28).

In the example illustrated in Figure 6.3, activity *Install WebApp* affects two managed elements: the software to be installed (*WebApp*), and the computer in which it will be installed ($Server_2$). After the abstraction step (line 27), the software installed by *Install WebApp* becomes Z (and has type *SoftwareElement*), and the computer in which the software is installed becomes X (of type *ComputerSystem*). *Configure DNS*, in turn, modifies software settings of the network's domain name server, located in computer $Server_3$. After the abstraction, the element affected by this activity becomes Y (*ComputerSystem*). All these three variables are stored in the set of variables D, as illustrated in Figure 6.3.

As a last step, the variables present in D are added to the list of variables of the change template T (line 30) and returned to the change operator (line 31). In Figure 6.3, the change template resulted from this process has three variables, X, Y, and Z, and is composed of two activities, ordered according to the relationship information acquired from the previous change plan C. At this point, the change template T is ready for use in the instantiation of similar future changes.

7 PROTOTYPICAL IMPLEMENTATION AND EXPERIMENTAL EVALUATION

In this chapter, we first present the prototypical implementation of a change management system used as a proof-of-concept (Section 7.1). Subsequently, we discuss the experimental evaluation conducted using this system (Section 7.2).

7.1 The ChangeLedge System

The solution for template-based change management presented in the previous chapters is supported by the ChangeLedge system, a prototypical implementation developed as part of this work. In this section, we further detail how the system components are implemented (Subsection 7.1.1) and how change initiators and IT operators interact with the system to design changes (Subsection 7.1.2).

7.1.1 System Architecture and Technologies

The ChangeLedge system is implemented in layers, as shown in Figure 7.1. The modules of each layer either implement conceptual components presented in Figure 3.1 or provide services to other components in the system. These components are further described below.

The Web interface was implemented using the Flex programming language, and is supported by the Apache/Tomcat and by Granite Data Services (GRANITEDS, 2007), a Java to Flex package conversion library. 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.

In the layer underlying the Web interface are located the modules *Template Mining*, *Template Management*, *Change Specification* and *Change Deployment*, which compose the core layer of the ChangeLedge system. The module *Template Mining* provides the functionalities for the discovery of plan templates from execution traces. The *Template Management* incorporates the functionalities for the specification of request and plan templates. The *Change Specification* implements the functionalities related to the design of RFCs/preliminary change plans. Finally, the *Change Deployment* delivers the functionalities for the refinement of preliminary plans into actionable workflows (and their later invocation). Considering the conceptual solution presented in Chapter 3, the module *Template Mining* materializes the functionalities of the component *Change Miner*; *Template Management* and *Change Specification* materialize the *Change Designer*; and *Change Deployment* materializes the *Change Planner*.

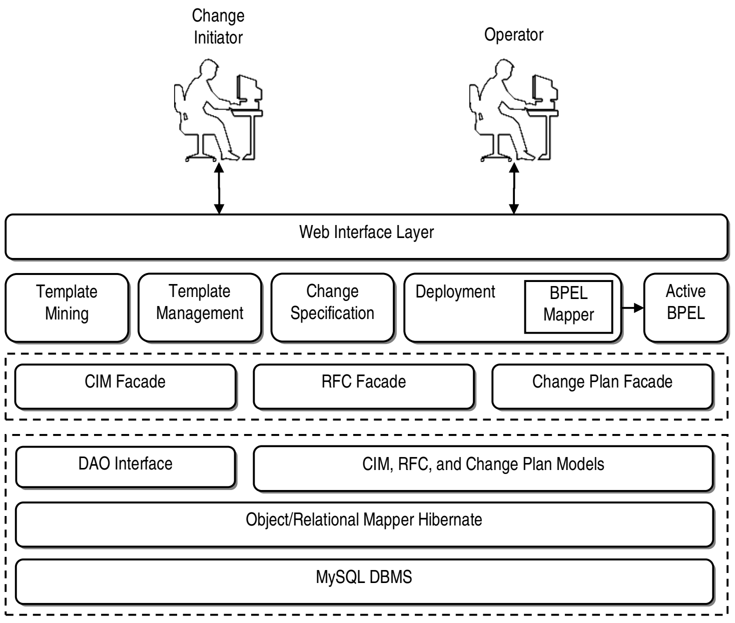


Figure 7.1: Implementation view of the ChangeLedge system

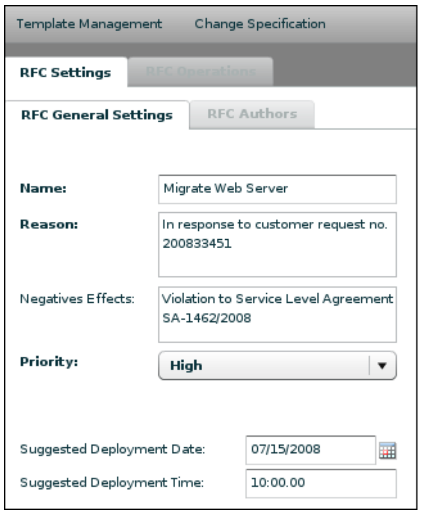
To fulfill their objectives, these modules employ the services provided by the layer underneath, which manipulates the data associated to the *IT infrastructure* and *RFC & Change Plan* models. These services are provided by the modules *CIM Facade*, *RFC Facade*, and *Change Plan Facade*. Below this layer, the persistence layer uses the Object/Relational Mapper *Hibernate* framework (REDHAT, 2007), and the Database Management System MySQL. The *Data Access Object* (DAO) design pattern is employed, in this context, to abstract and encapsulate all access to the IT-related repositories.

The actionable workflows generated by the system, more specifically by the module *Change Deployment*, are coded using the XML language, respecting the *Business Process Execution Language* (BPEL) (OASIS, 2007) standard. These documents are produced using the *Streaming API for XML* (StAX) (CODEHAUS, 2007) and executed through ActiveBPEL (ACTIVEENDPOINTS, 2007) (which materializes the component *Deployment System*, present in our conceptual solution). The choice for BPEL is due to the standard's popularity and its adherence to coordinate distributed actions over IT infrastructures.

7.1.2 Change Designer and Planner Assistant

The Web interface is the front-end for the access to all features provided by the ChangeLedge system. Through a Graphical User Interface (GUI), the change initiator and operator may interact with the modules (*i*) *Template Management*, to specify new templates (either request or plan), (*ii*) *Change Specification*, to design new change documents (either from templates or from scratch), (*iii*) *Change Deployment*, to invoke both refinement of preliminary plans into actionable change plans and the execution of the resulting workflows, and (*iv*) *Template Mining*, to trigger the discovery of change templates from traces of previously executed change processes.

Figure 7.2 presents two snapshots of the ChangeLedge system. The interface illustrated in Figure 7.2 (a) is the starting point for the specification of a new change (either from template or from scratch). In this stage, the change initiator may inform the name of the change, reason, priority, and suggested deployment date and time, among other information described in Section 2.1. Having filled in this information, the change initiator will indicate the operations that make up the newly specified change, as well as the primary affected CIs.



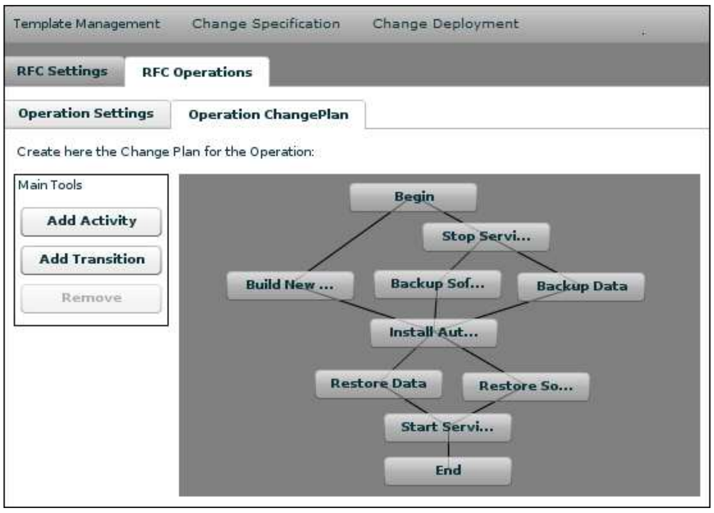
Template Management | Change Specification

RFC Settings | RFC Operations

RFC General Settings | RFC Authors

Name:	Migrate Web Server
Reason:	In response to customer request no. 200833451
Negatives Effects:	Violation to Service Level Agreement SA-1462/2008
Priority:	High
Suggested Deployment Date:	07/15/2008
Suggested Deployment Time:	10:00.00

(a) specification of new RFCs



(b) design of preliminary change plans

Figure 7.2: Graphical user interface of the ChangeLedge system

Likewise, Figure 7.2 (b) presents the main interface for the design of preliminary change plans. By interacting with this interface, the operator may detail the steps necessary to materialize the change objectives, either with the aid of templates (and the *composition*, *generalization* and *specialization* techniques, described in Chapter 4), or by designing the activities from scratch. After the design of the network of activities that will compose the preliminary plan, the RFC will be ready to be automatically refined into an actionable change plan (available in the *Change Deployment* menu option), and subsequently deployed over the managed IT infrastructure.

7.2 Experimental Evaluation

To prove the technical feasibility and the potentialities of using templates in the context of IT change management, we have conducted an experimental evaluation using the ChangeLedge system. All experiments have been conducted on a computer equipped with a Pentium^{TM} Centrino processor, 1.7 GHz of CPU clock, 2,048 KB of cache, and 512 MB of RAM memory.

We divide the presentation and discussion of the experiments conducted to evaluate our solution in two subsections. Subsection 7.2.1 describes the set of experiments performed to evaluate the use of change templates to instantiate RFC documents, and their refinement into actionable workflows. Subsection 7.2.2, in turn, emphasizes the evaluation of the template mining mechanism.

7.2.1 Change Design and Planning Evaluation

The IT infrastructure employed for the evaluation of the change design and planning functionalities of ChangeLedge is equivalent to the environment of a research

& development department of an organization. It is composed of 65 workstations, located in seven rooms, running either Windows XP SP2 or GNU/Linux. The environment is also composed of four servers, $Server_1$, $Server_2$, $Server_3$, and $Server_4$, whose relevant settings to the context of our evaluation are presented in Table 7.1. Similarly, the content of the Definitive Media Library (DML) is summarized in Table 7.2.

Table 7.1: General settings of the servers employed in the change design and planning evaluation

Server Name	Installed Operating System	Available Disk Space (MB)	Total Physical Memory (MB)
$Server_1$	None	20,480	2,048
$Server_2$	Windows 2003 Server	71,680	4,096
$Server_3$	Debian GNU/Linux	51,200	4,096
$Server_4$	Debian GNU/Linux	102,400	4,096

To conduct the evaluation of the proposed templates, we have instantiated, using templates available in the *Change Templates* repository, several change requests, of incremental complexity, considering the same target IT infrastructure. For the sake of brevity, we focus our attention on five of these changes. The first two have as objective the installation of an *e-Commerce Web application* (*WebApp*), one of them having $Server_1$ as target CI and the other, $Server_3$. The third RFC comprises two operations: one to install and configure an authentication server on $Server_3$, and the other to install and configure a network monitoring platform on $Server_4$. The fourth RFC comprises the migration of the entire system installed on $Server_3$ to $Server_4$. Finally, the fifth RFC consists in updating software packages installed in 47 out of the 65 stations that compose the IT infrastructure (typical procedure in several organizational contexts).

Having presented a general view of the experimental setup, Subsection 7.2.1.1 describes both (*i*) the qualitative analysis of the use of templates within the context of change management and (*ii*) an evaluation of the generation of actionable change plans aware of the runtime constraints imposed by the target IT infrastructure. Subsequently, Subsection 7.2.1.2 presents results on the performance of the ChangeLedge system in the computation of actionable workflows.

7.2.1.1 Qualitative Analysis

The first two previously described change requests were instantiated from the same template illustrated in Figure 7.3, *Install Web Application* (which, in turn, was specialized from the template presented in Figure 4.1). During the instantiation of the RFC documents, the CIs that were supposed to be primarily affected by the changes have been identified: the *e-Commerce Web application* (from the DML, and common to both RFCs), $Server_1$ (to the first RFC), and $Server_3$ (to the second one).

After the design of the RFC documents, we have instantiated preliminary plans for the installation of the *e-Commerce Web application.* The activities present in the preliminary plans, instantiated from the plan template also illustrated in Figure 7.3, are: *Install Web Application* (the core activity of the plan), *Load Database Schema* (creation of the database schema to be used by the application and population of the

Table 7.2: System requirements for the software present in the DML

Software Name	Disk Space (MB)	Memory (MB)	Software Dependencies
e-Commerce Web App for Windows	512	128	SQL Server Internet Information Server
e-Commerce Web App for Linux	512	128	*mysql-server-5.0* *php5-mysql* *apache*
Internet Information Server 5.1	15	16	Windows XP Service Pack 2
Internet Information Server 7.0	15	16	Windows Vista Service Pack 1
.Net Framework 4.8	280	256	Internet Explorer Internet Information Server Windows XP Service Pack 2
SQL Server 2005	425	512	Internet Explorer Windows XP Service Pack 2 .Net Framework
SQL Server 2008	1,460	1,024	Internet Explorer Windows Vista Service Pack 1
Internet Explorer 7	64	128	Windows XP Service Pack 2
Windows XP SP 2	1,800	-	Windows XP
Windows Vista SP 1	5,445	-	Windows Vista
Windows XP	1,500	128	-
Windows Vista	15,000	1,024	-
mysql-server-5.0	70	128	*mysql-common-5.0* *mysql-client-5.0* GNU/Linux
mysql-client-5.0	16	8	*mysql-common-5.0* GNU/Linux
mysql-common-5.0	1	-	GNU/Linux
apache	1	32	*apache-common* *apache2-utils* GNU/Linux
apache-common	3	-	*apache2-utils* GNU/Linux
apache2-utils	1	-	GNU/Linux
php5-mysql	1	-	*libapache-mod-php5* *mysql-client-5.0* *php5*
libapache-mod-php5	5	-	*apache-common* *php5-common*
php5	1	-	*php5-common*
php5-common	1	-	GNU/Linux
Debian GNU/Linux 4.0	1,024	256	-

database with preliminary data), *Configure the Network's DNS* (publication of the address in which the Web application may be reached), and *Start Web Application* (start of the main service provided by the Web application). This instantiation has consisted basically in assigning the CIs affected by each activity that were not yet

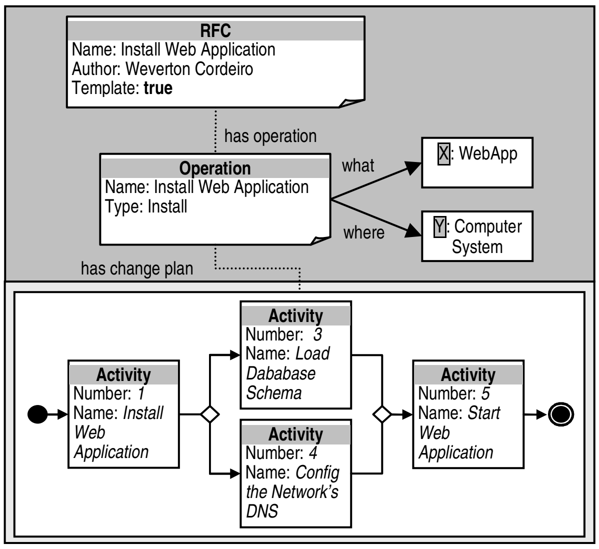

RFC
Name: Install Web Application
Author: Weverton Cordeiro
Template: true
has operation
Operation
Name: Install Web Application
Type: Install
what
where
X: WebApp
Y: Computer System
has change plan
Activity
Number: 1
Name: Install Web Application
Activity
Number: 3
Name: Load Dababase Schema
Activity
Number: 4
Name: Config the Network's DNS
Activity
Number: 5
Name: Start Web Application

Figure 7.3: Template for the installation of a generic Web application (*WebApp*)

assigned by the binding mechanism (explained in Subsection 4.1).

Focusing on activity *Install Web Application*, the specification of this activity, according to the Activity Modeling Notation, was *install SoftwareElement what at ComputerSystem where*. The two affected CIs *what* and *where*, automatically assigned with the CIs specified during the instantiation of the RFCs, were *e-Commerce Web application* and $Server_1$, respectively, for the first RFC. For the second, *what* was also assigned to *e-Commerce Web application*. The CI labeled *where*, however, was assigned to $Server_3$.

The actionable workflows generated to accomplish the objectives of the first two RFCs are illustrated, respectively, in Figures 7.4 and 7.5. For the sake of clarity, only the refinement of the *Install Web Application* is presented in both figures. The linkage between the activities that compose the workflows reflects the dependencies between installed packages, configuration procedures, among others. For example, in Figure 7.5, installing *apache* on top of Debian GNU/Linux requires the prior installation of the basic libraries provided by *apache-common*. In addition, having *e-Commerce Web Application* functional requires configuring the Web Server (*IIS 5.1*, for Windows, and *apache*, in the case of GNU/Linux) and granting permissions for the use of the Database Management System (*mysql-server*, in the change plan for the GNU/Linux environment, and *SQL Server 2005*, in the case of Windows XP).

The reader may note that there are significant differences between the workflows present in the figures. This is explained by the diverse group of affected CIs and *checks/actions* involved in their manipulation. To illustrate, *Internet Information Server* only requires a few steps to have it installed and functional (extracted from the chain of actions represented using the *IT infrastructure* model, described in Subsection 3.3). Furthermore, it depends on the prior installation of the operating system (dependency depicted from the *OperatingSystemVersionCheck* information, also represented using the *IT infrastructure* model). For Debian GNU/Linux, on the other hand, the *apache* installer requires several dependencies to be satisfied (in the target environment, only *apache2-utils* and *apache-common* dependencies were not yet fulfilled), plus the prior installation of the operating system. One may also note that the activities present in the workflow may be either automatically executed

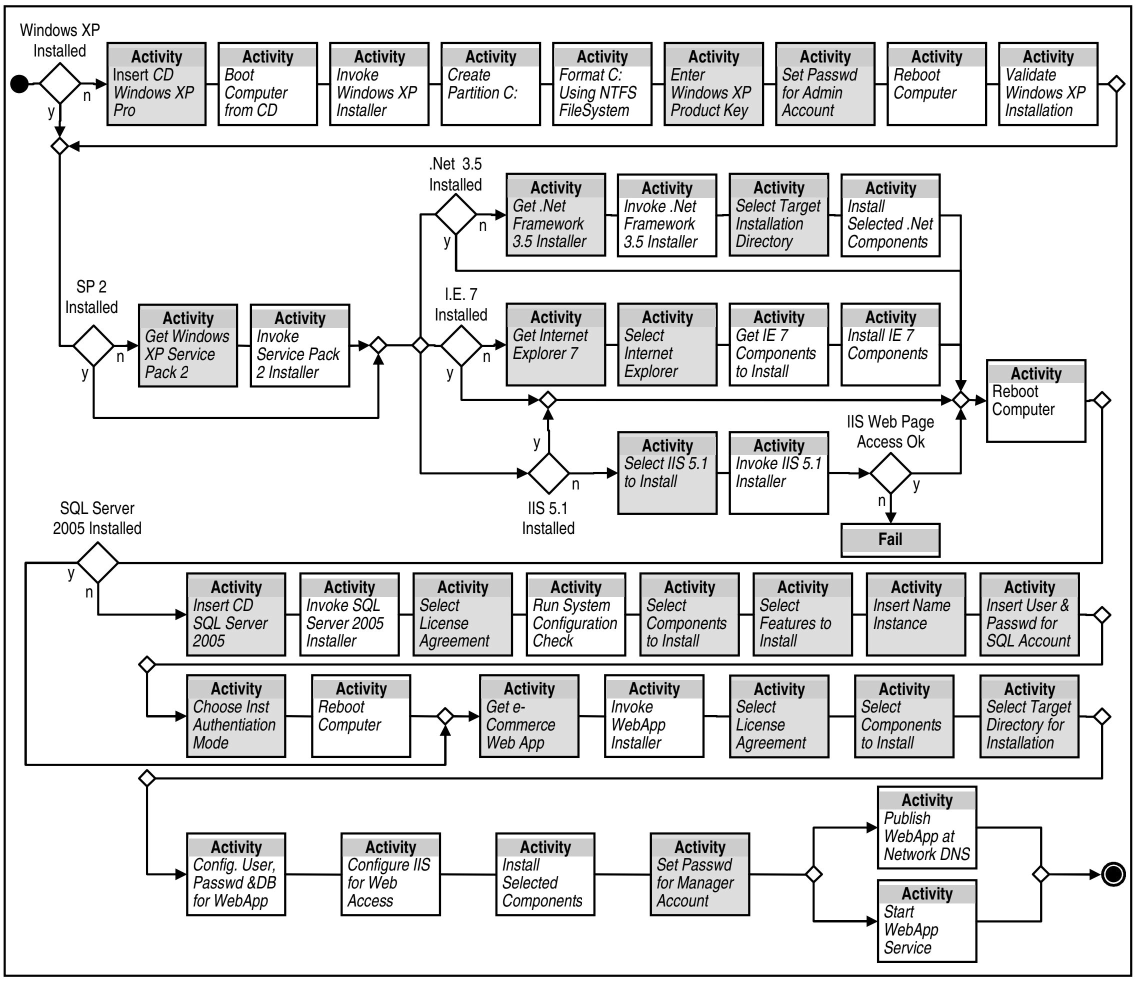

Windows XP Installed
Activity Insert CD Windows XP Pro
Activity Boot Computer from CD
Activity Invoke Windows XP Installer
Activity Create Partition C:
Activity Format C: Using NTFS FileSystem
Activity Enter Windows XP Product Key
Activity Set Passwd for Admin Account
Activity Reboot Computer
Activity Validate Windows XP Installation
.Net 3.5 Installed
Activity Get .Net Framework 3.5 Installer
Activity Invoke .Net Framework 3.5 Installer
Activity Select Target Installation Directory
Activity Install Selected .Net Components
SP 2 Installed
Activity Get Windows XP Service Pack 2
Activity Invoke Service Pack 2 Installer
I.E. 7 Installed
Activity Get Internet Explorer 7
Activity Select Internet Explorer
Activity Get IE 7 Components to Install
Activity Install IE 7 Components
Activity Reboot Computer
IIS Web Page Access Ok
Activity Select IIS 5.1 to Install
Activity Invoke IIS 5.1 Installer
Fail
IIS 5.1 Installed
SQL Server 2005 Installed
Activity Insert CD SQL Server 2005
Activity Invoke SQL Server 2005 Installer
Activity Select License Agreement
Activity Run System Configuration Check
Activity Select Components to Install
Activity Select Features to Install
Activity Insert Name Instance
Activity Insert User & Passwd for SQL Account
Activity Choose Inst Authentiation Mode
Activity Reboot Computer
Activity Get e-Commerce Web App
Activity Invoke WebApp Installer
Activity Select License Agreement
Activity Select Components to Install
Activity Select Target Directory for Installation
Activity Config. User, Passwd &DB for WebApp
Activity Configure IIS for Web Access
Activity Install Selected Components
Activity Set Passwd for Manager Account
Activity Publish WebApp at Network DNS
Activity Start WebApp Service
y
n

Figure 7.4: Partial workflow for installing *WebApp* on top of *Windows XP Professional*

(background in white) or require the intervention of a human operator (background in gray).

The implementation of the actionable workflow presented in Figure 7.4 requires, considering the information in Table 7.1, about 4,596 MB of disk space, and a minimum of 1,168 MB of available physical memory from $Server_1$. As for the workflow presented in Figure 7.5, the demands are 1,124 MB of disk space and about 424 MB of physical memory. Since both $Server_1$ and $Server_3$ have sufficient disk space for the installation procedures present in the workflows, the implementation of both RFCs is likely to succeed. Moreover, all the installed software should execute normally, given that the target servers have sufficient physical memory.

An alternative plan for the Windows environment (Figure 7.4) is the one in which *SQL Server 2008* is installed instead of *SQL Server 2005*, and *Internet Information Server 7.0*, instead of *IIS 5.1*. As a consequence, *Windows Vista* and *Windows Vista Service Pack 1* would be installed as well, instead of *Windows XP Service Pack 2* and *Windows XP*, due to the pre-requisite information. For the same reason, the installation of *.Net Framework 3.5* would not be present in this alternative plan. This plan would require 22,496 MB of available disk space from $Server_1$ to be executable, amount beyond the 20,480 MB currently available. Therefore, it would not be generated by our refinement algorithm, since it is impractical considering the

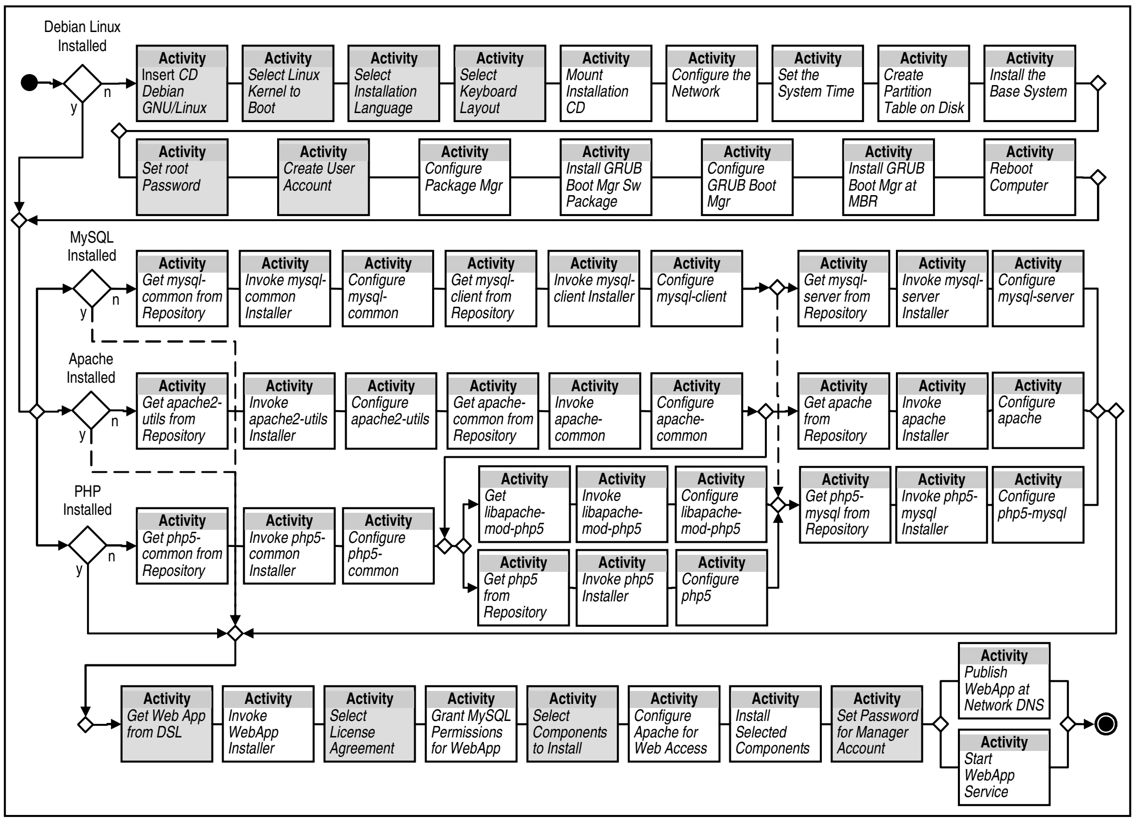

Debian Linux Installed
y
n
Activity Insert CD Debian GNU/Linux
Activity Select Linux Kernel to Boot
Activity Select Installation Language
Activity Select Keyboard Layout
Activity Mount Installation CD
Activity Configure the Network
Activity Set the System Time
Activity Create Partition Table on Disk
Activity Install the Base System
Activity Set root Password
Activity Create User Account
Activity Configure Package Mgr
Activity Install GRUB Boot Mgr Sw Package
Activity Configure GRUB Boot Mgr
Activity Install GRUB Boot Mgr at MBR
Activity Reboot Computer
MySQL Installed
Activity Get mysql-common from Repository
Activity Invoke mysql-common Installer
Activity Configure mysql-common
Activity Get mysql-client from Repository
Activity Invoke mysql-client Installer
Activity Configure mysql-client
Activity Get mysql-server from Repository
Activity Invoke mysql-server Installer
Activity Configure mysql-server
Apache Installed
Activity Get apache2-utils from Repository
Activity Invoke apache2-utils Installer
Activity Configure apache2-utils
Activity Get apache-common from Repository
Activity Invoke apache-common
Activity Configure apache-common
Activity Get apache from Repository
Activity Invoke apache Installer
Activity Configure apache
PHP Installed
Activity Get php5-common from Repository
Activity Invoke php5-common Installer
Activity Configure php5-common
Activity Get libapache-mod-php5
Activity Invoke libapache-mod-php5
Activity Configure libapache-mod-php5
Activity Get php5 from Repository
Activity Invoke php5 Installer
Activity Configure php5
Activity Get php5-mysql from Repository
Activity Invoke php5-mysql Installer
Activity Configure php5-mysql
Activity Get Web App from DSL
Activity Invoke WebApp Installer
Activity Select License Agreement
Activity Grant MySQL Permissions for WebApp
Activity Select Components to Install
Activity Configure Apache for Web Access
Activity Install Selected Components
Activity Set Password for Manager Account
Activity Publish WebApp at Network DNS
Activity Start WebApp Service

Figure 7.5: Partial workflow for installing *WebApp* on top of *Debian GNU/Linux*

imposed resource constraints.

In summary, the use of change templates was shown to be flexible, enabling (*i*) the reuse of knowledge, through the specification of similar, recurrent changes, (*ii*) a faster design of preliminary change plans, and (*iii*) the automated refinement of preliminary change documents into actionable workflows. Moreover, the generation of accurate, workable change plans, composed by activities that do not hinder the execution of subsequent ones, has potential to decrease the occurrence of change-related incidents and service-delivery disruption caused by failed changes.

7.2.1.2 Quantitative Analysis

Table 7.3 characterizes, synthetically, the actionable workflows generated for the five submitted RFCs. We highlight in the table the number of activities, as well as the number of computer systems (stations), operating systems, and software affected in both the preliminary (specified by a human operator with the aid of templates) and actionable plans (generated by the system). Taking the fourth RFC as example, one may note that the final change plan has 182 activities, automatically refined from a 40% smaller preliminary plan.

The time consumed by the ChangeLedge system to generate the aforementioned actionable plans is presented in Table 7.4. The system has performed satisfactorily, demanding from a few hundreds of milliseconds (1,209 for the first scenario) to a few dozens of seconds (62 for the last scenario) to generate the refined plans. We have also calculated a confidence interval of 95% for the measured times, considering 10 repetitions of the refinement process for each change document. The values

Table 7.3: Numeric complexity of the submitted changes (pre and post refinement)

Scenario	Preliminary Plan				Refined Plan			
	Activities	Affected Stations	Affected OSes	Affected Software	Activities	Affected Stations	Affected OSes	Affected Software
1	3	2	0	1	54	2	1	7
2	3	2	0	1	70	2	1	12
3	4	2	0	2	30	2	1	26
4	46	3	0	5	182	3	1	47
5	235	47	0	6	613	47	2	29

obtained show that the amount of time necessary for the automated refinement is expected to vary minimally.

Table 7.4: Time consumed by the *ChangeLedge* system to generate actionable workflows

Scenario	Refinement Time (ms)	Standard Deviation	Average Refinement Time per Activity (ms)	Confidence Interval	
				Lower Bound (ms)	Upper Bound (ms)
1	1,209	10.6	22	1,188	1,230
2	1,430	7	20	1,416	1,443
3	2,360	7.7	78	2,345	2,375
4	3,966	41.2	21	3,885	4,047
5	62,901	91	102	62,722	63,079

Note that the time to generate a refined change plan (and the average time to build a single activity that composes it), in each scenario, is negligible compared to the time that an operator would spend to generate the same workflow, using any off-the-shelf workflow designer. The reason is as follows. An operator, during the manual refinement of preliminary plans, needs to identify every dependency between software packages to be installed and CIs to be affected by the change. Subsequently, he/she must translate the dependencies that were not yet fulfilled in the target IT environment into actions to be executed. The complexity of these two tasks increases dramatically with the number of affected components and unmet dependencies. Furthermore, there is the additional effort to identify such information from distributed sources, causing the operator to miss important information, and therefore generate incomplete plans. In contrast, our refinement algorithm is able to capture the dependency information behind requested changes, and compute the require actions in a very short time. To conclude, our refinement algorithm is not only feasible to generate complete and correct plans, but has potential to reduce, in a significant way, time and efforts demanded to this end.

7.2.2 Evaluation of the Template Mining Mechanism

The conceptual and technical feasibility of change template mining mechanism proposed in this thesis has been evaluated considering the extraction of change templates from synthetically generated traces. This decision was made because it enables a more thorough analysis of the mechanism, by enabling the comparison of the extracted templates with the real processes that have implemented the changes. Again, for the sake of brevity, we focus our analysis on three of these templates. As a result of the template mining process, we have observed the correctness and completeness of the obtained templates, in addition to performance indicators.

The IT infrastructure employed in the evaluation of the template mining functionality of ChangeLedge was composed of four servers: *Server*$_1$, having no operating system installed on top of it; *Server*$_2$, having *Windows 2003* installed; *Server*$_3$, and *Server*$_4$, both having *Debian GNU/Linux* installed.

In regard to the change logs consumed during the mining process, they were generated by simulation of the execution of four different RFCs on the previously described IT environment. For each RFC, we have generated one change log with 500 change traces. Each simulation has considered uniform distributions for the time spent in the execution of each activity, as well as distinct probabilities, defined empirically, for choices present in the change plans.

The first two RFCs used in the simulation had as objective the installation of an *e-Commerce Web application* (*WebApp*), one of them having *Server*$_1$ as target CI (*Install e-Commerce Web application at Server*$_1$) and the other, *Server*$_3$ (*Install e-Commerce Web application at Server*$_3$). The third RFC comprised the migration of the entire system installed on *Server*$_3$ to *Server*$_4$ (*Migrate System Hosted on Server*$_3$ *to Server*$_4$). Information about the number of activities and types of transitions that composed the change plan associated to each of the three RFCs is presented in Table 7.5.

Table 7.5: Characteristics of the change plans employed in the template mining evaluation

RFC	Activities	Conditional Transitions	Unconditional Transitions
1	26	16	29
2	84	26	115
3	245	100	383

7.2.2.1 *Qualitative Analysis*

A partial view of the log generated from the execution of the first change is presented in Table 7.6, whereas the characteristics of the templates extracted from the logs of the three RFCs are summarized in Table 7.7. An interesting point to highlight is that templates 1 and 2, generated from RFCs 1 and 2, respectively, encode the prototypical steps (illustrated in Figure 7.6) to implement a same general change. This is explained by the same goal that both RFCs share, even though they are concerned with distinct environments, *i.e.*, having different sets of dependencies to be fulfilled.

Figures 7.6 and 7.7 show a partial view of the templates extracted from the execution traces of the first and third changes, respectively. The reader may note, in the change template from Figure 7.6, that *Activity 2*, *Install Security Updates for WebApp*, is conditionally executed (for example, whenever the system in which *WebApp* is installed requires these updates). An indication of existence of this decision may be observed, for example, in traces 3 and 4. In the first trace, *Activity*

Table 7.6: Partial view of the change log of RFC *Install e-Commerce Web application at* $Server_1$

Change Trace	Executed change actions
Trace 1	$a_1, a_4, a_5, a_3, a_7, a_6, a_8, a_9$
Trace 2	$a_1, a_2, a_5, a_7, a_4, a_3, a_6, a_8, a_9$
Trace 3	$a_1, a_2, a_4, a_3, a_5, a_6, a_7, a_8, a_9$
Trace 4	$a_1, a_5, a_4, a_3, a_6, a_7, a_8, a_9$
Trace 5	$a_1, a_4, a_3, a_6, a_5, a_7, a_8, a_9$

Table 7.7: Characteristics of the templates discovered using the template mining mechanism

Template	Activities	Conditional Transitions	Unconditional Transitions
1	9	2	15
2	9	2	15
3	57	0	65

2 is executed after *Activity 1*, *Install WebApp*. In the second, though, *Activity 2* is not executed. The reader may also note that the order of execution of *Activities 3*, *4*, *5*, *6*, and *7* varies in the change traces, indicating the existence of parallelism upon their execution. As shown in Figure 7.6, this behavior is also captured and materialized in the extracted change template.

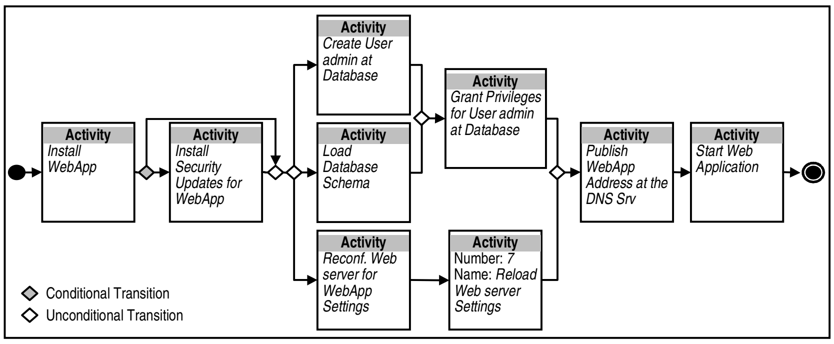

Activity
Install WebApp
Activity
Install Security Updates for WebApp
Activity
Create User admin at Database
Activity
Load Database Schema
Activity
Reconf. Web server for WebApp Settings
Activity
Grant Privileges for User admin at Database
Activity
Number: 7
Name: Reload Web server Settings
Activity
Publish WebApp Address at the DNS Srv
Activity
Start Web Application
Conditional Transition
Unconditional Transition

Figure 7.6: Partial view of the template extracted from the change log of RFC *Install e-Commerce Web application at* $Server_1$

As for the change template illustrated in Figure 7.7, we highlight the size and complexity of the depicted change template, extracted from a change log that comprised over 245 distinct, low level change actions. It is important to mention that the completeness and generality of these templates is highly influenced by the number of change traces available during the template mining process. For example, if a smaller set of change traces were used to compute the change template from Figure 7.7, the representativeness of the 245 activities would be affected, thus making the template mining mechanism to interpret some (or most of them) as noise. As a consequence, the resulting template would be less complete and general (*e.g.*, composed of fewer activities or transitions between them).

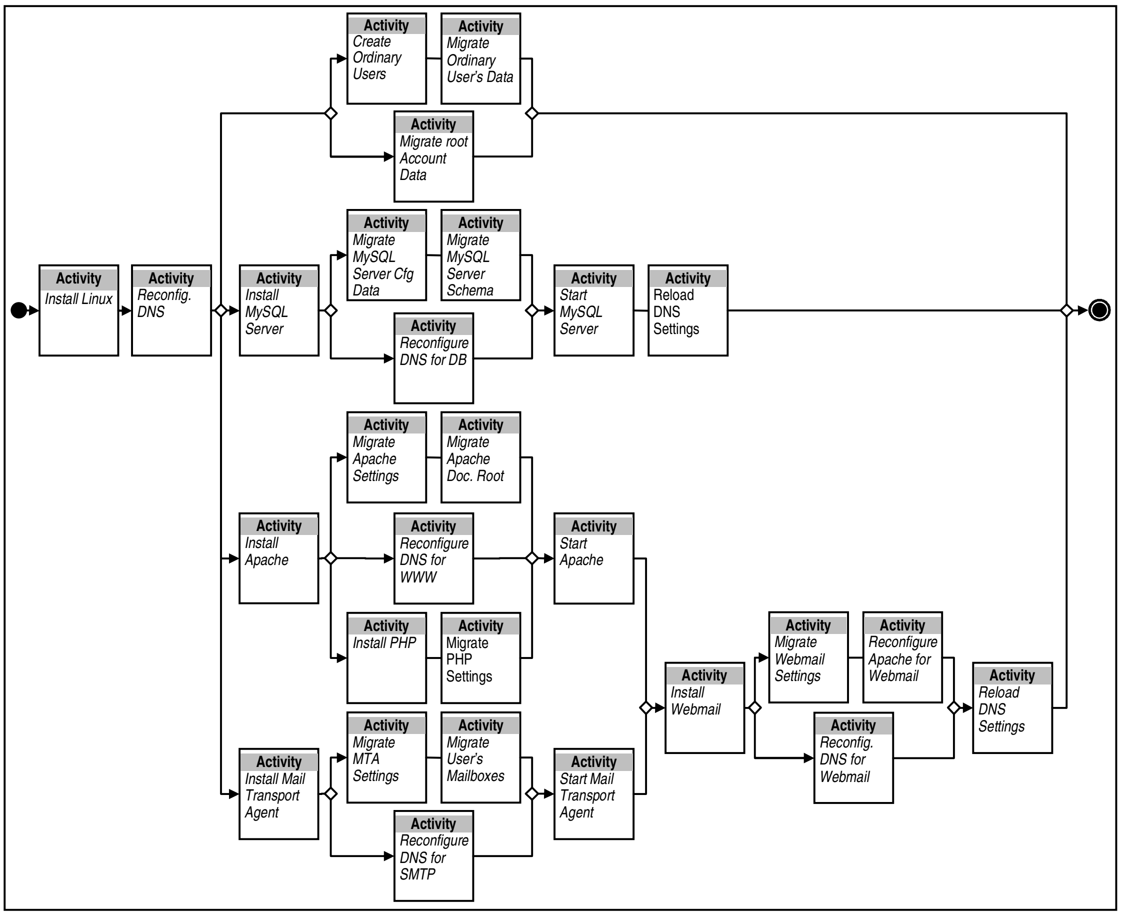


Figure 7.7: Partial view of the template generated for RFC *Migrate System Hosted on Server*$_3$ *to Server*$_4$

7.2.2.2 Quantitative Analysis

The performance of ChangeLegde to extract the previously mentioned templates is depicted in Table 7.8. Exhibiting a performance similar to the planning functionality (as shown in Subsection 7.2.1.2), the ChangeLegde system has demanded a few dozens of seconds (from 31 to 112) to generate the templates.

Table 7.8: Time consumed by the *ChangeLedge* system to extract templates

Scenario	Mining Time (s)	Confidence Interval	
		Lower Bound (ms)	Upper Bound (ms)
1	31	29	33
2	36	34	39
3	112	104	121

From the results presented in Table 7.8, we expect the template mining processing time to vary minimally, for each scenario. These results show that our template mining mechanism not only generates complete and correct change templates, but has potential to perform it in a time of lower magnitude than would be spent by a skilled human operator, using any off-the-shelf workflow editor.

8 CONCLUSION

We have discussed in this thesis some of the benefits of capturing and reusing IT change knowledge. The lack of a common standard to aid the design of changes, along with proper tool support to assist this process, makes such knowledge reuse difficult in practice. To address these issues, we have proposed (*i*) a conceptual solution to support the design and planning of IT changes, (*ii*) the use of change templates as a mechanism to ease the formalization and reuse of the experience accumulated within organizations in relation to IT changes, and (*iii*) a mechanism that enables the capture and reuse of knowledge – through the use of change templates – from historic data generated during the implementation of past changes in organizations. Our solution is supported by ChangeLedge, a prototypical implementation of a change management system.

8.1 Contributions of this Thesis

The main novel contributions of our work are fourfold. First, we have introduced request and plan templates as a mechanism to formalize, preserve, and reuse knowledge acquired by change managers and operators in the conduction of changes. Operations such as association, composition, generalization, and specialization expand the potentialities of using templates in the specification of changes. Second, the template-based mechanism is accompanied by an algorithm that enables the generation of detailed, actionable workflows aligned to IT resource constraints. Third, we have tailored the process mining techniques proposed in the literature (MARUSTER et al., 2002) to the context of IT change management, and proposed a mechanism to convert process models, discovered from historic data, into change management templates. The discovered templates, in turn, enable the reuse of past change processes in a broader range of scenarios, having settings diverse of the original process. Fourth, in order to support the reuse of knowledge through templates, the automated refinement of change plans, and the discovery of change templates from historic data, we have introduced an end-to-end solution, supported by a real system, to allow planning and implementation of IT changes to be designed and executed.

The results obtained are quite positive. The use of request and plan templates showed to be flexible to allow the design of RFCs and (different levels of) preliminary change plans, for several types of IT changes. The possibility of *associating*, *composing*, *specializing*, and *generalizing* templates provides the change operators with a powerful mechanism to structure knowledge that otherwise would remain with individuals. In addition, the automated refinement of preliminary change plans (designed by human operators) into actionable workflows ran on the order of hundreds of milliseconds to dozens of seconds and has resulted in highly detailed change plans. The generated plans have respected the restrictions imposed by the target environment (*e.g.*, memory and disk space constraints), and included aspects not explored in previous investigations (*e.g.*, activities that involve human operators). Furthermore, the discovery of change templates from execution traces has showed to be feasible, yielding the generation of templates – also on the order of hundreds of milliseconds to dozens of seconds – that capture the nuances of the changes implemented by the original plans.

8.2 Considerations on the Proposed Solution

In this section, we provide a critical analysis of the proposed solution, highlighting the relationship with other concepts, scope, and applicability.

ChangeLedge comprises a *workflow* of ordered steps (change design, planning, execution, *etc.*), with documents or data being coordinately passed from one participant (a person or a software component) to another. Therefore, from this point of view, we can state that our solution implements sort of a *business logic*. However, this "logic" is not exclusive to an organization, but general, since it is aligned to the best practices and processes defined by ITIL.

On the other hand, looking at our solution from a systems' perspective, it is important to mention that ChangeLedge process information received from change managers, IT operators, and IT related repositories. This is another facet of the basic business logic present in our solution, in order to ensure that the documents produced along the change design process are consistent. Apart from that, the automated refinement of RFCs into detailed change plans and the deployment of the changes are not subject to business rules and consistencies.

It is also important to highlight that ChangeLedge provides an end-to-end solution for the conduction of IT changes, from the early design to the deployment and evaluation of the achieved results. However, it does not support any means for understanding the several aspects involved in the changes to be performed (*e.g.*, why are they needed, which strategies to use, *etc.*). Therefore, ChangeLedge does not indicate suitable methodologies to conduct the requested changes, considering any identified aspects and trade-offs, following the same line of systems thinking (BOARDMAN; SAUSER, 2008) and systems-of-systems methodology (JACKSON; KEYS, 1984).

In regard to the scope of our solution, it is targeted at the *design*, *planning*, and *deployment* of changes. Therefore, other phases that comprise the traditional change management process, *e.g.*, *assessment*, *testing*, *authorization*, and *schedule*, are envisaged as directions and trends for long-term, future investigation in this area. As for the execution of the resulting actionable workflows and the interface necessary in the CIs for their remote manipulation, these issues have been subject of another work by our research group and, for this reason, have not been presented here. The interested reader may refer to Machado *et al.* (2008) for additional information.

Now focusing on the template mining functionality of our solution, it is important to mention that the change templates extracted with our mechanism are not influenced by traces that describe changes that have failed. Since these traces differ significantly from the traces that describe successful changes, the events registered by them (*e.g.*, direct and indirect succession between activities) are regarded as *noise*

(MEDEIROS et al., 2004). As a consequence, they become statistically irrelevant to the template mining process.

Another important aspect worth discussing is the applicability of our solution. Since our solution aims to be aligned to the set of best practices and processes recommended by ITIL, it may be applied for management of Information Technology assets in the several levels of an organization, *i.e.*, from switches, routers, and data links, to services and distributed applications. Furthermore, our solution has the potential to fit adequately in several contexts, from small to large organizations, observed the adherence of their respective management practices to ITIL's recommendations.

8.3 Research Avenues for Future Investigations

Despite the progresses achieved and ongoing investigations, and in addition to the long-term research topics enumerated in Section 8.2, there are several other research opportunities in the field of IT change management that merit attention. Considering a short-term research agenda, we intend to investigate decision support mechanisms to help operators understand the trade-offs between alternative change designs. Moreover, since our problem of IT change design concerns the realization of sequences of activities from a description of the goal and an initial state of the IT environment, we plan to explore how IT change design can take advantage of Artificial Intelligence (AI) planning techniques (NAU et al., 2003).

As a simplification assumption, we have considered *and/or branches/joins* as not being influenced by environmental data. We intend to address this issue by incorporating decision mining techniques (ROZINAT; AALST, 2006) to our template mining mechanism, in order to generate change templates composed of decision structures that take into account these data. Finally, we plan to (*i*) evaluate our solution taking into consideration real-life data (*e.g.*, mining change logs generated by Opsware (OPSWARE, 2008)), (*ii*) further investigate the sensitivity of the template mining process to the number and nature of the traces available, and the complexity of changes available in execution logs, and (iii) deal with the problem of conflicts (in terms of goals, resource access, etc.) between changes designed, planned, and deployed in parallel.

REFERENCES

ACTIVEENDPOINTS. **ActiveBPEL for SOA Orquestration**. Available at: <http://www.activebpel.org>. Visited on: May 2007.

BOARDMAN, J.; SAUSER, B. **Systems Thinking**: coping with 21st century problems. Boca Raton, USA: CRC Press, 2008.

BON, J. V.; JONG, A. de et al. **IT Service Management - An Introduction**. Zaltbommel, NL: Zaltbommel : Van Haran Publishing, 2007.

CODEHAUS. **The Streaming API for XML (StAX)**. Available at: <http://stax.codehaus.org>. Visited on: Jun. 2007.

CORDEIRO, W.; MACHADO, G.; ANDREIS, F.; SANTOS, A.; BOTH, C.; GASPARY, L.; GRANVILLE, L.; BARTOLINI, C.; TRASTOUR, D. A Runtime Constraint-Aware Solution for Automated Refinement of IT Change Plans. In: IFIP/IEEE INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEMS: OPERATIONS AND MANAGEMENT, DSOM, 19., 2008, Samos Island, Greece. **Managing Large Scale Service Deployment**: proceedings... Berlin: Springer, 2008. p.69–82.

CORDEIRO, W.; MACHADO, G.; ANDREIS, F.; SANTOS, A.; BOTH, C.; GASPARY, L.; GRANVILLE, L.; BARTOLINI, C.; TRASTOUR, D. ChangeMiner: a solution for discovering it change templates from past execution traces. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, IM, 11., 2009, New York, USA. **Proceedings...** Piscataway: IEEE Operations Center, 2009. p.1–8.

CORDEIRO, W.; MACHADO, G.; DAITX, F.; BOTH, C.; GASPARY, L.; GRANVILLE, L.; 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, 11., 2008, Salvador, Brazil. **Proceedings...** Piscataway: IEEE Operations Center, 2008. p.355–362.

CORMEN, T. H. et al. **Introduction to Algorithms**. Cambridge, USA: The Massachusetts Institute of Technology Press, 2001.

DMTF. **Common Information Model**. Available at: <http://www.dmtf.org/standards/cim>. Visited on: Apr. 2007.

DUMITRAS, T. et al. Ecotopia: an ecological framework for change management in distributed systems. In: WORKSHOP ON ARCHITECTING DEPENDABLE SYSTEMS, WADS, 6., 2007, Edinburgh, Scotland. **Architecting Dependable Systems IV**: proceedings... Berlin: Springer, 2007. p.262–286. (Lecture Notes in Computer Science, v.4615).

FINK, R. **Modelling and Assisting the Design of IT Changes**. 2009. Dissertação (Mestrado em Ciência da Computação) — Institut für Informatik, Der Ludwig-Maximilians-Universität, München, Germany.

GRANITEDS. **Granite Data Services**. Available at: <http://www.graniteds.org>. Visited on: Aug. 2007.

ISACA. **Control Objectives for Information and related Technologies (COBIT)**. Available at: <http://www.isaca.org/cobit>. Visited on: May 2008.

JACKSON, M.; KEYS, P. Towards a System of Systems Methodologies. **Journal of the Operational Research Society**, [S.l.], v.35, p.473–486, 1984.

KELLER, A. Automating the Change Management Process with Electronic Contracts. In: IEEE INTERNATIONAL CONFERENCE ON E-COMMERCE TECHNOLOGY WORKSHOPS, CECW, 7., 2005, München, Germany. **Proceedings...** Piscataway: IEEE Operations Center, 2005. p.99–107.

KELLER, A. et al. The CHAMPS System: change management with planning and scheduling. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, NOMS, 9., 2004, Seoul, Korea. **Proceedings...** Piscataway: IEEE Operations Center, 2004. p.395–408.

LACY, S.; MACFARLANE, I. **ITIL Service Transition Version 3.0**. London, UK: The Stationery Office, 2007.

MACHADO, G.; CORDEIRO, W.; DAITX, F.; BOTH, C.; GASPARY, L.; GRANVILLE, L.; SAIKOSKI, K.; SAHAI, A.; BARTOLINI, C.; TRASTOUR, D. Enabling Rollback Support in IT Change Management Systems. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, NOMS, 11., 2008, Salvador, Brazil. **Proceedings...** Piscataway: IEEE Operations Center, 2008. p.347–354.

MARIN, M.; NORIN, R.; SHAPIRO, R. **Workflow Process Definition Interface – XML Process Definition Language**. [S.l.: s.n.], 2008. Available at: <http://www.wfmc.org>. Visited on: Oct. 2008.

MARUSTER, L. et al. Process Mining: discovering direct successors in process logs. In: INTERNATIONAL CONFERENCE ON DISCOVERY SCIENCE, DS, 5., 2002, Lübeck, Germany. **Proceedings...** Berlin: Springer-Verlag, 2002. p.364–373.

MEDEIROS, A. K. A. de et al. **Process Mining**: extending the alpha-algorithm to mine short loops. Eindhoven, The Netherlands: [s.n.], 2004. Available at: <http://prom.win.tue.nl/research/wiki/publications/medeiros2004>. Visited on: May. 2008.

NAU, D. S. et al. SHOP2: an htn planning system. **Journal of Artificial Intelligence Research**, [S.l.], v.20, p.379–404, 2003.

OASIS. **Business Process Execution Language, Version 2.0**. Available at: <http://docs.oasis-open.org/wsbpel/2.0>. Visited on: May 2007.

OGC. **Information Technology Infrastructure Library (ITIL)**. Available at: <http://www.itil-officialsite.com>. Visited on: May 2008.

OPSWARE. **Opsware Data Center Automation Platform**. Available at: <http://www.opsware.com>. Visited on: Jun. 2008.

REBOUCAS, R. et al. A Decision Support Tool to Optimize Scheduling of IT Changes. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, IM, 10., 2007, München, Germany. **Proceedings...** Piscataway: IEEE Operations Center, 2007. p.343–352.

REDHAT. **Hibernate**: relational persistence for java and .net. Available at: <http://www.hibernate.org>. Visited on: Jun. 2007.

ROZINAT, A.; AALST, W. van der. Decision Mining in ProM. In: INTERNATIONAL CONFERENCE ON BUSINESS PROCESS MANAGEMENT, BPM, 4., 2006, Vienna, Austria. **Proceedings...** [S.l.]: Springer, 2006. p.420–425.

ROZINAT, A. et al. Discovering colored Petri nets from event logs. **International Journal on Software Tools for Technology Transfer**, Berlin, Heidelberg, v.10, n.1, p.57–74, 2007.

SAUVÉ, J. P. et al. On the Risk Exposure and Priority Determination of Changes in IT Service Management. In: IFIP/IEEE INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEMS: OPERATIONS AND MANAGEMENT, DSOM, 18., 2007, San Jose, CA. **Managing Virtualization of Networks and Services**: proceedings... Berlin: Springer, 2007. p.147–158. (Lecture Notes in Computer Science, v.4785).

SHANKAR, C. et al. Specification-Enhanced Policies for Automated Management of Changes in IT Systems. In: LARGE INSTALLATION SYSTEM ADMINISTRATION CONFERENCE, LISA, 20., 2006, Washington, USA. **Proceedings...** Berkeley: CA: USENIX Association, 2006. p.103–118.

TRASTOUR, D. et al. Activity-Based Scheduling of IT Changes. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS INFRASTRUCTURE, MANAGEMENT AND SECURITY, AIMS, 1., 2007, Oslo, Norway. **Inter-Domain Management**: proceedings... Berlin: Springer, 2007. p.73–84. (Lecture Notes in Computer Science, v.4543).

APPENDIX A RESUMO ESTENDIDO DA DISSERTAÇÃO

O uso e a importância de sistemas em rede e distribuídos para apoiar as atividades de negócio das organizações tem aumentado de forma significativa recentemente. Esse cenário tornou imperativo o uso de abordagens efetivas para a gerência desses sistemas (também chamados de sistemas de Tecnologia da Informação – TI nesta dissertação) de forma adequada e eficiente. A Biblioteca de Infraestrutura de Tecnologia da Informação (*Information Technology Infrastructure Library*, ITIL) (OGC, 2008) se tornou, nesse contexto, um dos conjuntos mais aceitos e adotados de boas práticas e processos para a implantação e gerência de serviços de TI (REBOUCAS et al., 2007), possuindo uma importância especial para as organizações que possuem serviços dinâmicos, de larga escala e em constante evolução.

Para lidar com mudanças em sistemas de TI, frequentemente introduzidas em resposta às necessidades de negócio, a ITIL define o processo de gerência de mudanças (*change management*) (LACY; MACFARLANE, 2007). Esse processo tem por objetivo maximizar o valor que mudanças adicionam aos negócios, através da minimização de ocorrência dos incidentes relacionados à mudanças e disrupção no provisionamento de serviços. Para este fim, a gerência de mudanças recomenda o uso de métodos e procedimentos padronizados para lidar com mudanças.

Conforme descrito no livro Suporte a Serviços da ITIL (*ITIL Service Transition book*) (LACY; MACFARLANE, 2007), a gerência de mudanças compreende desde a especificação de documentos chamados Requisições de Mudança (*Requests For Change*, RFC), por um *solicitante da mudança* (*change initiator*), até a geração, quer seja por um *operador de TI* (*IT operator*) ou automatizada, de *planos de mudança executáveis* (*actionable change plans*). As RFCs, tipicamente solicitadas por usuários de negócios, expressam as mudanças requisitadas em um alto nível de abstração (por exemplo, *Implantar Novo Serviço de Comércio Eletrônico* e *Aumentar a Capacidade do Enlace de Dados*). Planos de mudança executáveis, por sua vez, são *workflows* que compreendem ações de baixo nível (por exemplo, instalação de *software*, manipulação de tabelas de roteamento e modificações em configurações) a serem implantadas na infra-estrutura de TI gerenciada. Se executados, esses planos de mudança executáveis (também chamados de *workflows* executáveis nesta dissertação) devem implantar as mudanças requisitadas. As fases subsequentes de um processo de gerência de mudanças tradicional são a *avaliação*, o *teste*, a *autorização*, o *agendamento*, a *implantação*, e o *registro* das ações executadas na Base de Dados de Gerência de Configuração (*Configuration Management Database*, CMDB).

Para alcançar os objetivos da gerência de mudança previamente mencionados, é de fundamental importância reutilizar a experiência adquirida com mudanças passadas no projeto de requisições futuras. A ITIL sugere o uso de modelos de mudança (*change models*) como uma forma para permitir o reuso de tal experiência em mudanças recorrentes e similares. Idealmente, a criação desses modelos pode ser concretizada considerando duas abordagens distintas. Em uma abordagem *top-down*, operadores de TI podem projetar os modelos manualmente, com base no conhecimento adquirido no passado. Em uma perspectiva alternativa, *bottom-up*, esses modelos poderiam ser extraídos a partir de traços de mudanças passadas obtidos com orquestradores de mudanças (ou com qualquer orquestrador de *workflows* de propósito geral).

Na prática, no entanto, as RFCs e os planos de mudança têm sido geralmente descritos e documentados de forma *ad hoc* (REBOUCAS et al., 2007), na qual a linguagem natural é frequentemente empregada para expressar *o quê*, *por quê*, *quando*, *onde* e *como* as mudanças devem ser executadas. A falta de um padrão comum amplamente aceito e adotado por usuários de negócio e operadores de TI para apoiar o projeto de mudanças leva, por exemplo, a documentos especificados ou com excesso de detalhamento ou com informações insuficientes, e à interpretações errôneas dos documentos relacionados a mudanças produzidos. Mais importante – e o foco da investigação apresentada nesta dissertação – isso impede que o conhecimento possuído/adquirido pelo pessoal responsável pela especificação, planejamento e condução de mudanças seja reutilizado em requisições recorrentes e similares.

São destacadas duas motivações que justificam a importância de reutilizar a experiência dos operadores na implantação de mudanças em TI. Primeiro, à medida que planos de mudança são recorrentemente instanciados, eles se tornam mais estáveis e, portanto, o uso dos mesmos pode resultar em menos incidentes (na execução de mudanças). Segundo, o tempo consumido na especificação e planejamento de mudanças tende a ser reduzido, dado que as mesmas raramente serão geradas "do início".

Desde a concepção da ITIL, tem havido pesquisa substancial em questões relacionadas à gerência de mudanças em TI. Por exemplo, passos importantes têm sido dados em direção à exploração de oportunidade de paralelismo na execução de planos de mudança (KELLER et al., 2004), otimização de ambos o agendamento de mudanças em janelas de manutenção e a atribuição de técnicos para executá-las (TRASTOUR et al., 2007), e na obtenção de agendamentos de mudanças que satisfaçam a objetivos/restrições de negócios (REBOUCAS et al., 2007). No entanto, apesar dos potenciais benefícios de se reusar o conhecimento oriundo de mudanças em TI, este tópico tem sido pouco abordado em investigações passadas (KELLER et al., 2004; LACY; MACFARLANE, 2007; KELLER, 2005).

Para lidar com essa lacuna, nesta dissertação é proposta uma solução, baseada nas recomendações para a gerência de mudanças proposta pela ITIL, para apoiar o projeto e planejamento de mudanças em sistemas de TI. As contribuições desta dissertação se desdobram em quatro. Primeiro, é introduzido o conceito de *templates* de mudança como um mecanismo para formalizar, preservar e reusar a experiência acumulada nas organizações em relação a mudanças em TI. Esses *templates* podem existir em níveis de abstração diferentes de um sistema de gerenciamento de mudanças. São caracterizados os *templates de requisição* os utilizados pelo solicitante da mudança para especificar RFCs. De forma complementar, *templates de plano* compreendem atividades de alto nível requeridas para satisfazer os objetivos da

RFC. Segundo, é proposto um algoritmo para o refinamento automatizado de atividades em alto nível, presentes em *templates* de plano, em atividades de mais baixo nível, de modo a produzir planos de mudança detalhados e executáveis. O algoritmo proposto foca no impacto que as ações precedentes já computadas causarão na infraestrutura de TI, ao computar as ações subsequentes. Como consequência, os planos de mudança executáveis gerados utilizando o algoritmo proposto estarão "cientes" das limitações, em tempo de execução, impostas pelo ambiente de TI alvo da mudança(ex., limitações de espaço em disco e memória). Terceiro, é apresentado um mecanismo, inspirado em técnicas de mineração de processos (MARUSTER et al., 2002), para extrair *templates* de mudança a partir de traços de execuções gerados por orquestradores de mudanças. Em contraste com as técnicas de mineração de *workflows* já estabelecidas, o mecanismo proposto concentra-se na obtenção de *templates* que capturam a essência dos processos de mudança previamente executados. Os *templates* extraídos, por sua vez, permitem o reuso e aplicabilidade de tais processos em cenários com configurações diversas em relação ao cenário original (ex., relacionamentos de dependência diferentes entre os componentes de TI afetados, ou requisitos de sistema específicos). E quarto, para apoiar o reuso de conhecimento por intermédio de templates, o refinamento automatizado de planos de mudança, e a extração de *templates* de mudança a partir de dados históricos, é introduzida uma solução fim-a-fim, apoiada por um sistema real, para permitir o planejamento e implantação de mudanças em TI.

O uso de *templates* de mudança, o refinamento de planos preliminares em *workflows* executáveis e a extração de *templates* a partir de traços de execuções foram avaliados por meio de uma implementação prototípica de um sistema de gerência de mudanças chamado ChangeLedge. Além de permitir o projeto (manual), a extração (automatizada) e o (re)uso de *templates* na especificação de novos documentos de mudança, o sistema é capaz de computar planos de mudança executáveis a partir da correlação de RFCs instanciadas/planos de mudança elaborados com informações de dependências disponíveis em repositórios relacionados à TI. Os planos executáveis resultantes podem ser, por sua vez, trivialmente traduzidos em uma linguagem de *workflow* e executados por qualquer sistema de implantação de mudanças. Para provar conceito e viabilidade técnica da solução proposta, foram conduzidos vários experimentos para explorar as potencialidades do emprego de *templates* como um mecanismo para preservar e reusar o *expertise* adquirido com mudanças em TI.

A.1 Contribuições

As principais contribuições apresentadas nesta dissertação se desdobram em quatro. Primeiro, foram introduzidos os *templates* de requisição e de plano como um mecanismo para formalizar, preservar e reusar o conhecimento adquirido por gerentes e operadores de TI na condução de mudanças. As operações de associação, composição, generalização e especialização expandem as potencialidades do uso de *templates* na especificação de mudanças. Segundo, o mecanismo de *templates* é acompanhado por um algoritmo que permite a geração de *workflows* detalhados, executáveis e alinhados com as limitações de recursos de TI. Terceiro, as técnicas de mineração de processos propostas na literatura (MARUSTER et al., 2002) foram adaptadas ao contexto da gerência de mudanças em TI, e foi proposto um mecanismo para converter modelos de processos, extraídos a partir de dados históricos, em *templates* de mudança. Os *templates* extraídos, por sua vez, permitem o reuso de processos de mudança executados no passado em uma ampla variedade de cenários, com configurações diversas em relação às do processo original. Quarto, com o objetivo de apoiar o reuso de conhecimento por meio de *templates*, o refinamento automatizado de planos de mudança, e a extração de *templates* de mudança a partir de dados históricos, foi apresentada uma solução fim-a-fim, apoiada por sistema, para permitir o planejamento e implantação de mudanças em TI.

Os resultados obtidos são bastante positivos. O uso de *templates* de requisição e de plano mostrou-se flexível para permitir o projeto de RFCs e planos de mudança preliminares (em níveis diferentes de granularidade), para vários tipos de mudanças em TI. A possibilidade de *associar*, *compor*, *especializar* e *generalizar templates* proporciona aos operadores de TI um mecanismo poderoso para estruturar o conhecimento que, de outra forma, permaneceria com os indivíduos. Além disso, o refinamento automatizado de planos de mudança preliminares (projetados por operadores humanos) em *workflows* executáveis foi realizado na ordem de centenas de milissegundos a dezenas de segundos, e resultou em planos de mudança altamente detalhados. Os planos gerados respeitaram às restrições impostas pelo ambiente alvo (ex., limitações de memória e de espaço em disco), e incluíram aspectos não explorados em investigações anteriores (ex., atividades que envolviam operadores humanos). Por fim, a extração de *templates* de mudança a partir de traços de execuções se mostrou viável, levando à geração de *templates* – também na ordem de centenas de milissegundos a dezenas de segundos – que capturam as *nuances* das mudanças implantadas pelos planos originais.

A.2 Considerações sobre a Solução Proposta

O sistema ChangeLedge compreende um *workflow* de passos ordenados (projeto de mudança, planejamento, execução, *etc.*), com documentos ou dados sendo transmitidos de forma coordenada de um participante (uma pessoa ou um componente de *software*) para outro. Portanto, considerando esse ponto de vista, é possível afirmar que a solução proposta implementa uma forma de *lógica de negócio*. No entanto, essa "lógica" não é exclusiva de uma organização, porém geral, uma vez que está alinhada com as boas práticas e processos definidas pela ITIL.

Por outro lado, olhando para a solução proposta em uma perspectiva de sistema, é importante mencionar que o ChangeLedge processa informações recebidas de gerentes de mudança, operadores e de repositórios relacionados à TI. Essa é outra face da lógica de negócio básica presente na solução, com o objetivo de garantir que os documentos produzidos ao longo do processo de projeto de mudanças são consistentes. Apesar disso, o refinamento automatizado de RFCs em planos de mudança detalhados e a implantação das mudanças não estão sujeitas a regras de negócio e consistências.

É importante destacar também que o ChangeLedge proporciona uma solução fim-a-fim para a condução de mudanças em TI, desde o projeto até a implantação e avaliação dos resultados alcançados. No entanto, o mesmo não oferece apoio de qualquer natureza para o entendimento dos vários aspectos relacionados com as mudanças a serem conduzidas (ex., por quê são necessárias, quais estratégias utilizar, *etc.*). Portanto, o ChangeLedge não indica metodologias adequadas para conduzir as mudanças requisitadas, considerando e identificando aspectos e *trade-offs*, seguindo a mesma linha de *dinâmica de sistemas* (*systems thinking*) (BOARDMAN; SAUSER, 2008) e *systems-of-systems methodology* (JACKSON; KEYS, 1984).

Em relação ao escopo da solução proposta, ele compreende o *projeto*, *planejamento*, e *implantação* de mudanças. Logo, outras fases que compõem um processo tradicional de gerência de mudanças, ex., *avaliação*, *teste*, *autorização* e *agendamento*, são tratadas como direções e tendências para investigação de longo prazo nessa área. Sobre a implantação das mudanças descritas nos *workflows* executáveis gerados pelo sistema e a interface necessária nos Itens de Configuração (*Configuration Items* – CIs) para sua manipulação remota, esses têm sido abordados em outros trabalhos conduzidos por este grupo de pesquisa e, por esta razão, não são apresentados nesta dissertação. O leitor interessado pode referir-se a Machado *et al.* (2008) para informações adicionais.

Agora olhando para a funcionalidade de extração de *templates* da solução proposta, é importante mencionar que os *templates* gerados pelo mecanismo apresentado nesta dissertação não são influenciados por traços que descrevem mudanças que falharam. Uma vez que tais traços diferem significativamente dos traços que descrevem mudanças executadas com sucesso, os eventos registrados pelos mesmos (ex., sucessão direta e indireta entre as atividades) são considerados como *ruídos* (MEDEIROS et al., 2004). Consequentemente, esses tornam-se estatisticamente irrelevantes para o processo de extração de *templates* de mudança.

Outro aspecto importante que merece discussão é a aplicabilidade da solução proposta. Dado que a solução visa estar alinhada com o conjunto de boas práticas e procesos recomendados pela ITIL, ela pode ser aplicada para a gerência de recursos de tecnologia da informação nos vários níveis de uma organização , *i.e.*, de *switches*, roteadores e enlaces de dados, até serviços e aplicações distribuídas. Mais ainda, a solução proposta tem potencial para encaixar-se adequadamente em diversos contextos, de pequenas a grandes organizações, desde que as mesmas observem a aderência de suas práticas de gerência com as recomendações da ITIL.

A.3 Trabalhos Futuros

Apesar dos progressos alcançados e das investigações em andamento, e somando-se aos tópicos de pesquisa de longo prazo enumerados na Seção A.2, há várias outras oportunidades de pesquisa no campo de gerência de mudanças em TI que merecem atenção. Considerando uma agenda de pesquisa de curto prazo, tem-se como objetivo investigar mecanismos de suporte à decisão que ajudem operadores a entender os *trade-offs* entre projetos de mudança alternativos. Além disso, uma vez que o problema do projeto de mudanças em TI está relacionado com a materialização de uma sequência de atividades a partir de uma descrição dos estados inicial e final do ambiente de TI, pretende-se explorar como o projeto de mudanças em TI pode tirar proveito de técnicas de Inteligência Artificial (IA) (NAU et al., 2003).

Como premissa simplificatória, foi considerado que os *and*/*or branches*/*joins* presentes nos *workflows* não são influenciados por dados do ambiente no qual o mesmo é executado. Essa questão deverá ser abordada por meio da incorporação de técnicas de mineração de decisões (ROZINAT; AALST, 2006) ao mecanismo de mineração de *templates*, com o objetivo de gerar *templates* de mudança compostos de estruturas de decisão que levem em consideração tais dados. Por fim, pretende-se (*i*) avaliar a solução proposta nesta dissertação levando em consideração traços de execuções reais (ex., a mineração de traços históricos gerados pelo Opsware (OP-SWARE, 2008)), (*ii*) investigar a sensitividade do processo de mineração de *templates* em relação ao número de traços disponíveis e à complexidade das mudanças disponíveis em tais traços, e (*iii*) abordar o problema de conflitos (em termos de objetivos, acesso a recursos, etc.) entre mudanças projetadas, planejadas, e executadas em paralelo.

APPENDIX B PUBLISHED PAPER – NOMS 2008

In this attachment we present the paper entitled “A Template-based Solution to Support Knowledge Reuse in IT Change Design”, one of the deliverables of the work described in this thesis. The paper introduces the concept of templates in the context of IT change management. Furthermore, the paper depicts the conceptual solution to allow planning and implementation of IT changes to be, respectively, designed and executed. Finally, the paper presents the results obtained with the experimental evaluation using the ChangeLedge system, showing the technical feasibility of the use of change templates to promote the reuse of knowledge acquired from previous changes.

- Title: A Template-based Solution to Support Knowledge Reuse in IT Change Design
- Conference: IEEE/IFIP Network Operations and Management Symposium (NOMS 08)
- URL: http://www2.dcc.ufmg.br/eventos/noms2008/
- Date: 7-11 April 2008
- Venue: Pestana Bahia Hotel, Salvador, Brazil

A Template-based Solution to Support Knowledge Reuse in IT Change Design

Weverton Luis da Costa Cordeiro,
Guilherme Sperb Machado, Fábio Fabian Daitx,
Cristiano Bonato Both, Luciano Paschoal Gaspary,
Lisandro Zambenedetti Granville
Institute of Informatics
Federal University of Rio Grande do Sul, Brazil
{weverton.cordeiro, gsmachado, ffdaitx,
cbboth, paschoal, granville}@inf.ufrgs.br

Akhil Sahai[1], Claudio Bartolini[1],
David Trastour[2], Katia Saikoski[3]
[1]HP Laboratories Palo Alto, USA
[2]HP Laboratories Bristol, UK
[3]HP Brazil R&D, Brazil

{akhil.sahai, claudio.bartolini,
david.trastour, katia.saikoski}@hp.com

Abstract– **Capturing and reusing the experience of operators in implementing IT changes is an important aspect of IT service management, as it may result in fewer incidents (upon change execution) and faster specification of change plans, to mention just a few potential advantages. Nevertheless, in practice, changes are usually described and documented in an *ad hoc* fashion, due to the lack of proper support to assist the design process. This hampers knowledge acquired when specifying, planning, and carrying out previous changes to be reused in subsequent requests. In order to address this issue, we propose the use of *change templates* as a mechanism to formalize, preserve, and reuse the experience accumulated within organizations in relation to IT changes. Our solution is analyzed through a prototypical implementation of a change management system and a case study based on a real-life scenario.**

I. Introduction

The Information Technology Infrastructure Library (ITIL) has become, in the recent past, one of the most important references for Information Technology (IT) infrastructure management [1]. ITIL is composed of a set of best practices and processes that spread from service strategy to service operation. Among these processes, *change management* plays an important role in the efficient and prompt handling of IT changes [2].

Change management, as described in the ITIL Service Support book [3], defines that changes required to be executed over the managed IT infrastructure should be specified in documents called *Requests for Change* (RFC). RFCs are then supposed to be processed, either manually or automatically, for the generation of *change plans*. A change plan consists of a workflow of actions that, when executed, will move the managed system from the current workable state into another workable state.

In practice, RFCs and change plans are traditionally modeled in an *ad hoc* fashion [4], where natural language is often employed to express what, why, when, where, and how changes should be executed. The lack of a common standard widely accepted and adopted by IT practitioners to assist this process often leads, for example, to documents with either too much or not enough information than actually required and to erroneous interpretation of the produced change-related documents. More important than that − and the focus of our investigation − it hampers knowledge owned/acquired by the personnel responsible for specifying, planning, and carrying out changes to be reused in recurrent or similar requests.

We highlight two reasons that justify the importance of reusing the experience of operators in the implementation of IT changes. First, as change plans are recurrently instantiated, they become more stable and, therefore, their use may result in fewer incidents (upon change execution). Second, the time consumed by the IT personnel in the specification and planning of changes tends to be reduced, given that they will seldom be generated "from scratch".

Since the inception of ITIL, there has been substantial research on IT change management issues. For example, important steps have been taken towards automated planning and scheduling of change plans [2], policy definition for event reaction in IT systems [5], and business-driven change schedule optimization [4]. However, despite the potential benefits of reusing knowledge behind IT changes, this topic has been barely addressed in previous investigations [2, 3, 6] (this will be further discussed in Section II).

To bridge this gap, in this paper we propose the use of *change templates* as a mechanism to formalize, preserve, and reuse the experience accumulated within organizations in relation to IT changes. These templates may exist at different abstraction levels of a change management system. We characterize *request templates* as those used by the change requester to specify RFC documents. In contrast, *plan templates* comprise activities, which represent large-grained steps required to accomplish the RFC objectives. These activities may need to be iteratively refined into finer-grained ones in order to produce detailed, actionable change plans (also called actionable workflows throughout this paper).

Our solution has been evaluated through a prototypical implementation of a change management system called ChangeLedge. In addition to allowing templates to be designed and (re)used in the specification of new change documents, the system is able to compute actionable change plans by correlating instantiated RFCs/sketched plans with dependency information available in IT-related repositories. The resulting actionable workflows can, in turn, be straightforwardly translated into a workflow language and executed by any off-the-shelf deployment system. To prove concept and technical feasibility of our proposed solution, we have also conducted a case study to explore the potentialities of employing templates as a resource for preserving and reusing the expertise acquired with IT changes.

The remainder of this paper is organized as follows. Section II discusses related work on IT change management. Section III introduces our conceptual solution. Section IV details how the concept of templates is tailored to knowledge reuse in the context of IT change management. Section V emphasizes implementation aspects of ChangeLedge and presents the results achieved with the case study. Section VI concludes the paper with remarks and perspectives for future work.

II. Related Work

In recent years, several research efforts on IT change management have been carried out within the operations and management community. In this section, we cover some of the most prominent investigations.

The ITIL Service Support book [3] recommends the use of *change models* to both define a library of recurrent changes and foresee the impact associated to them, once performed. However, since ITIL concentrates on documenting generally applicable industry best practices, it is out of its scope to propose *how* to materialize such models. In spite of this, ITIL represents a common ground upon which our solution is built.

Keller *et al.* [2] have proposed CHAMPS, a system for automating the generation of change plans that explore a high degree of parallelism. The topic of knowledge reuse was not the focus of this work. Even though the system enables some degree of *reuse* through *abstract workflows*, the scope of the reuse is limited. That is so because these workflows cannot be generalized, once they are tied to a given combination of software packages. Refinements, on the other hand, are restricted to "*editing*/*saving as*" the already existing workflows. The lack of specialization/generalization capabilities hampers knowledge to be better structured and systematically reused in different future changes.

Aware of the importance of formalizing IT change documents, Keller [6] has introduced, in a subsequent work, the concept of *electronic contracts*. Four types of contracts are proposed: Requests for Change, Deployment Descriptors, Policies and Best Practices, and Service Level Agreements (SLAs). In regards to the specification of RFCs − central to our investigation − the author only enumerates the parameters to be supplied in an RFC, but does not propose a more robust model to express them.

Other relevant research efforts in the field of IT change management have been recently published. Shankar *et al.* [5] have proposed an enhanced policy-based solution to execute changes on the IT infrastructure, triggered by the managed system and not by a human operator, in response to the occurrence of events. Looking at IT changes from a business perspective, Rebouças *et al.* [4] have investigated approaches for the planning and scheduling of changes aiming at minimizing costs (e.g., labor costs and financial loss due to SLA disruption).

There are additional relevant research efforts published in other fields, e.g. Software Engineering and Knowledge Management, which explore similar aspects of our work. Nevertheless, due to space constraints and the fact that they are not related to IT change management itself, they are not approached in this paper.

Although change management is a relatively new discipline in the IT context, the area has been quickly progressing, as evidenced by the previously mentioned related work. Nevertheless, little has been done to facilitate change design and implementation by capturing, organizing, and reusing the knowledge acquired within IT organizations. In the following sections we envisage a solution to address these issues.

III. Conceptual Solution

A conceptual solution is proposed to allow planning and implementation of IT changes to be designed and executed. In contrast to previous investigations, our solution focuses on providing a systematic way to design changes − from RFCs to resulting change plan documents − by reusing the knowledge accumulated in the past. Figure 1 depicts the basis of our solution, highlighting its main conceptual components, personnel involved, and their interactions.

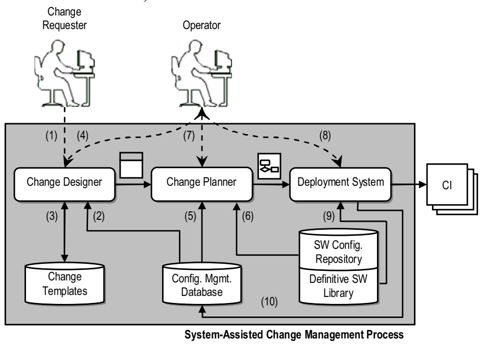


Fig. 1. Elements of the proposed solution and interactions.

The *Change Requester* starts a change process by interacting with the component *Change Designer* in order to specify an RFC (flow 1 in the figure). The *Configuration Management Database* (CMDB) provides the change requester with an updated view of the IT infrastructure (flow 2) so that he/she is able to precisely identify the hardware and software elements (Configuration Items – CIs) involved in the desired change. The creation of a new RFC document can be done "from scratch" or instantiated from a *request template* stored in a *Change Template* repository (3). The intent of a request template is to specify the set of high level objectives that must be met by a change. RFC templates will be created for routine and recurrent changes and will be persisted in the repository (3).

After an RFC is instantiated, an *Operator*, who is responsible for modeling the change procedure, interacts with the component *Change Designer* (4), now to sketch a preliminary change plan. In this stage, the operator specifies large-grained steps required to fulfill the RFC objectives, possibly taking advantage of *plan templates* available in the *Change Templates* repository (3). Plan templates encode composable workflows of change procedures and represent the knowledge gained from past experience in an IT department. Such templates are also stored in the Change Template repository (3). Request templates and plan templates will be described in more detail in Section IV.

The final refinement of the preliminary change plan into an actionable workflow is then performed, without human intervention, by the component *Change Planner.* This refinement is computed using a *task refiner algorithm*, based on both factual information about the IT infrastructure (5) and information about acceptable configurations and dependencies among software packages, available in the *Software Configuration Repository* (6). Thereafter, the resulting change plan can be modified by the operator as to precisely reflect his/her needs (7) and finally translated to a workflow language.

In the last step of the change process, the operator may invoke the actual execution of the workflow by interacting with an off-the-shelf *Deployment System* (8). To carry out some of the actions contained in the workflow, such a system may consume software packages available in the *Definitive Software Library* (9). After executing the change plan, the deployment system is responsible for updating the CMDB with the changes performed on every target CI (10).

Having presented a general view of our solution, in the following subsections we describe in more detail three important "building blocks" of the proposal: the model for the management and persistence of IT information, the model to design change-related documents, and the algorithm to generate actionable workflows.

A. IT Infrastructure Model

We chose to base our *IT Infrastructure Model* on a subset of the Common Information Model (CIM), defined by the Distributed Management Task Force (DMTF) [7]. It allows the representation of computing and business entities comprising an organization, as well as the relationships among them. Figure 2 shows a partial view of the model. The root class is *ManagedElement*. Through specialization, it is possible to represent any CI (e.g., physical devices, computer and application systems, and services), as well as IT personnel. Relationships such as associations, compositions, and aggregations, most of them omitted in the figure for the sake of legibility, map the dependencies among the elements comprising the infrastructure.

The model extends the scope of a traditional CMDB since it also incorporates classes such as *Check* and *Action*, which represent relevant information required for managing the lifecycle of CIs (e.g., software upgrade and application system installation/uninstallation). In Figure 2, we illustrate some specializations of classes *Check* and *Action* for the manipulation of a *Software Element.*

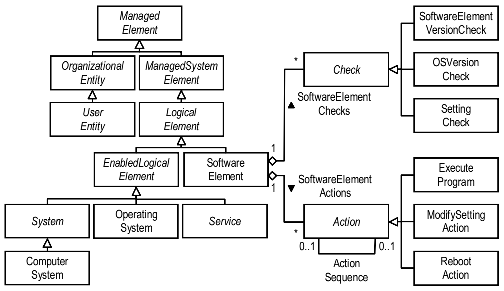


Fig. 2. Partial view of the IT infrastructure model.

An instance of class *Check* defines a condition to be met or characteristic required by the associated *Software Element* for it to evolve to a new state (e.g., deployable, installable, executable, or running). Possible checks include verification of software dependencies, available disk space and memory, and required environment settings. Each instance of class *Action,* in its turn, represents an operation of a process to change the state of the associated *Software Element* (e.g., from installable to executable). Examples of actions are invocation of a software installer/uninstaller, manipulation of files and directories, and modification of configuration files.

In addition to being used to represent the current IT infrastructure, the same model is also employed to specify *new* hardware and software packages (along with their dependencies) that may be required throughout a change process. Our conceptual solution proposes these two uses of the same information model to be materialized, respectively, in the CMDB and in the Software Configuration Repository, as previously presented in Figure 1.

B. Requests for Change & Change Plan Model

Our conceptual solution for template-based change management includes the proposal of a *Requests for Change & Change Plan Model*, to be employed in the design of change-related documents. The model relies on both (a) guidelines presented in the ITIL Service Support book [3] in regards to the change management process and (b) the workflow process definition, proposed by the Workflow Management Coalition (WfMC) [8].

Figure 3 presents a partial view of the model. It is structured in two connected parts. The former (whose background is highlighted in gray) permits the modeling of an RFC, while the latter (with white background) provides classes for expressing the corresponding change plan. Each one is detailed below.

An RFC (represented by class *RFC* in the model) is composed of operations (class *Operation*) that indicate, in a high level of abstraction, what changes the requester (class *UserEntity* of the IT Infrastructure model) would like to perform. It is also important to notice that each operation has an explicit reference to the CIs over which the change operation is supposed to be executed (class *ManagedElement*). Among other classes related to an RFC are: *ProblemReport* (problem that motivated the change request), *CABRecommendation* (RFC impact and resource requirement assessments), *RFCReview* (modifications to the RFC after its creation), and *RFCAuthorization* (approval of the RFC by a Change Authority).

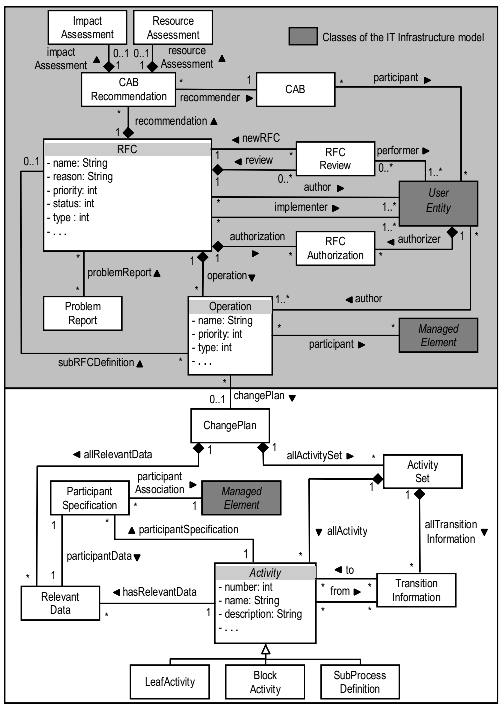


Fig. 3. Partial view of the change document model.

Every operation of an RFC has an associated change plan (class *ChangePlan*) that consists of a network of activities (class *Activity*) and their relationships. These activities may represent either low-level, actionable tasks (*LeafActivity*) or large-grained ones (*BlockActivity/SubProcessDefinition*), being the latter subject to further levels of refinement. In general, an activity manipulates one or more CIs (class *ManagedElement*). The class *ParticipantSpecification* describes the resources which perform a given activity, for example a human, a software package, or an automated machine. Transitions between activities are represented by instances of the class *TransitionInformation*, which may be of type *branch* (with conditions or not) or *join*. Finally, the class *RelevantData* allows the specification of data consumed and produced by each activity. We would like to point out that change plans modeled using the aforementioned classes are vendor neutral and, therefore, can be easily mapped to any workflow language, such as the Business Process Execution Language (BPEL) [9].

C. Generation of Change Plans

As previously mentioned, the refinement of preliminary change plans into actionable workflows is computed by means of a *task refiner and scheduler*. In our first iteration at this problem, we propose an approach that concentrates on *task refinement*. The algorithm is presented in Figure 4 below.

```
Refine-Operation (var rfc: RFC)
For each rfc.operation do
    for each operation.activity do
        Adt ← activity dependency tree
        current_set ← leaf nodes from adt
        while current_set is not empty do
            for each node in current_set do
                add node to activity subprocess
                delete node from adt
                delete node from current_set
        current_set ← leaf nodes from adt
```

Fig. 4. Pseudocode for the task refiner algorithm.

The algorithm processes the preliminary change plan associated to each RFC operation separately, originating independent actionable workflows. For each large-grained activity of a change plan, it executes the following steps: (1) checks dependencies among CIs affected by the activity; (2) generates new fine-grained activities based on these dependencies, building an activity dependency tree; and (3) orders the activities executing a reversal traversal on the tree, generating an actionable subworkflow.

For the refinement process to be automatically computable, the large-grained activities must be expressed unambiguously and understood by the algorithm. To address this issue, the class *Activity* introduced in the previous subsection presents an attribute called *description*, which is filled in with a sentence written in accordance to the *Activity Modeling Notation* (AMN), proposed in the scope of this research. Table I enumerates some example activities that can be specified using AMN.

TABLE I
ACTIVITY MODELING USING AMN

Abstract Activity	Abstract Specification
Start a Service	start Service <Service> with <Parameters>
Stop a Service	stop Service <Service> with <Parameters>
Install a Software	install SoftwareElement <SoftwareElement> at ComputerSystem <ComputerSystem> with <Parameters>
Uninstall a Software	uninstall SoftwareElement <SoftwareElement> from <ComputerSystem> with <Parameters>
Configure a Generic Element	configure ManagedSystemElement <ManagedSystemElement> at ComputerSystem <ComputerSystem> using Setting <Setting> with <Parameters>

For illustration, consider an activity that consists of installing a Web application X on a computer Y. Its specification using AMN, according to the previous table, is *install SoftwareElement* X *at ComputerSystem* Y *with* Z. In this example, X and Y are references to objects from the Software Configuration Repository and the CMDB, respectively. Z is a pointer to an artifact that contains a series of parameters taken as input by the Web application installer.

The automated refinement of the activity *install* mentioned above is started by looking up pre-requisites (instances of class *Check*) to be met and actions (instances of *Action*) to be executed to move application X from state *installable* to state *executable* (i.e., an installation procedure). This process is recursively executed (e.g., for new instances of *SoftwareElement* identified as dependencies) until all pre-requisites are solved. The result of this refinement, as mentioned earlier, is a subworkflow of actions necessary to materialize the activity goal.

IV. Change Management Templates

The conceptual solution presented in the previous section is the core basis upon which the notion of IT change templates is built. In this section, we explore it in more detail. Subsection A characterizes request and plan templates. The association between such templates, during the design of changes, is explained in Subsection B. Finally, Subsection C discusses composition, specialization, and generalization of templates.

A. Request and plan templates

In the context of this work, we define templates as parameterized building blocks that can be reused in the design of changes similar to previously executed ones. Explored in different levels of the change process, they are categorized as request or plan templates. Each one is detailed below.

A **request template**, defined by means of the upper part of the model illustrated in Figure 3, defines a set of prototypical operations that need to be performed on the IT infrastructure. The person producing such a template (e.g., change requester or operator) is free to specify which attributes/references will have fixed values and which will be left to be provided during the instantiation of a new RFC. As an example, Figure 5 (background highlighted in gray) presents a request template to deploy a hosting service in a dedicated server. This template is composed of two operations: *Build Dedicated Web Hosting Service* and *Lease Dedicated Data Circuit*. In the case of the first operation, the references *what* to be installed and *where* are defined to be of type *Web Application* and *Computer System*, respectively, thus specifying the type of CIs that may be affected.

The request template just described could be used, for example, to instantiate two distinct RFCs: one to install a J2EE Web application on a RISC computer and one to install an ASP.NET Web application on a 64-bit computer. For that to be possible, J2EE and ASP.NET apps should be present in the Software Configuration Repository (as *Web Application* objects), while RISC and 64-bit computers should appear (as *ComputerSystem* objects) in the CMDB.

Going down to the level of change plans, **plan templates** have a central role in the process of promoting the reuse of the experience accumulated by operators. Specified using the classes illustrated in the lower part of Figure 3, these templates comprise steps necessary to materialize recurrent changes. The large-grained activities of a plan template are typically loosely coupled, learned and tuned after a multitude of attempts to implement the corresponding changes successfully. For this reason, they are not automatically computable based on dependency information.

Figure 5 (background in white) depicts an example of a plan template for the installation of a new server, from scratch, and a Web application on top of it. *Activity 1* consists of building the new server into *Computer System* Y. *Activity 2* installs the *Web Application* X into the newly built server. Next, two parallel tasks take place: *Activity 3*, which loads the default database for use by the *Web Application* X, and *Activity 4*, which configures the users that will have remote access to the application. Finally, *Activity 5* comprises the instructions for publishing the URL of the Web application in the network's DNS server. This activity is defined to be executed in a final step because clients must not have access to the Web application until it is fully installed and operational.

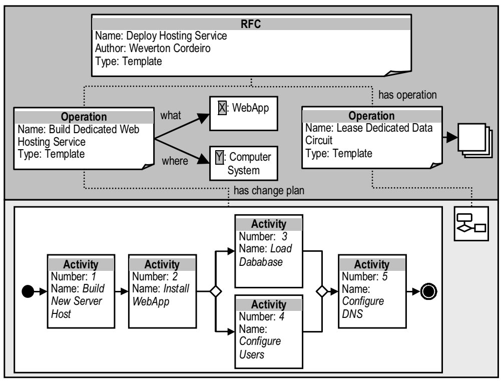


Fig. 5. Examples of request and plan templates.

B. Association between request and plan templates

Request and plan templates can co-exist independently from each other. This decoupling is necessary, since, on one hand, different changes may share similar sets of implementation procedures, hence similar plan templates, and on the other hand, there may be more than one valid design for a given operation.

The association of templates can be done in one of two ways: (1) gradually, after the instantiation of an RFC (taking advantage of a request template or not); or (2) at any point in time before an RFC instantiation. Regardless of the moment of such associations, they are achieved through the establishment of a relationship between the classes *Operation* (at the RFC level) and *ChangePlan* (at the change plan level).

When associating templates, the operator must specify the mapping between attributes/references from the RFC/request template and variables of the binded plan templates. It is expected that all attributes and/or references of the former are linked to one (or more) variable(s) of the latter. Variables of the plan template not set by this process must be assigned "values" by the operator in order to produce a consistent and complete preliminary change plan.

In the example illustrated in Figure 5, one can observe the association between a request and a plan template. One of the operations, *Build Dedicated Web Hosting Service*, has been associated to the plan template depicted in the lower part of the figure (explained in the previous subsection).

C. Composition, specialization, and generalization of templates

Reuse of knowledge in the context of IT changes is facilitated through the use of template *composition*, *specialization*, and *generalization*. These techniques are introduced next, considering both request and plan templates.

Template **composition** allows complex changes to be designed based on the experience formalized and preserved in previous processes. The basic principle is to use existing templates as "building blocks" in the design of new change documents. From the perspective of request templates, composition is possible, for example, by grouping operations that have been specified elsewhere in different request templates. This type of composition is supported by the association between the classes *RFC* and *Operation* of the model presented in Subsection III.B.

It is in the context of plan templates, though, that composition can be better taken advantage of. In this case, large-grained activities found in a plan template can be materialized by other plan templates, possibly forming a hierarchy of *nested templates*. Figure 6 illustrates a template for installing an operating system that is nested to *Activity 1*, *Build New Server Host*, of the plan template presented in Figure 5. Such nesting is made possible by using the class *SubProcessDefinition*.

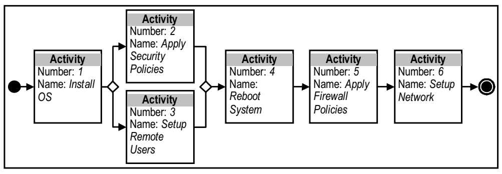


Fig. 6. A template for installing an operating system.

Specialization enables the creation of change templates to address non-ordinary changes, whose activities usually deviates slightly from those found in typical procedures. For request templates, specialization may be useful, for instance, to both extend an existing one with new operations and/or restrict the scope of the CIs that may be affected by them. For a concrete example, consider the request template shown in Figure 5. It could be specialized so that the affected CI, labeled as *what* in operation *Build Dedicated Web Host Service,* was restricted to J2EE Web applications, instead of any Web application, as originally defined.

One can also use specialization as a surrogate to model customized plan templates. For example, a template that dictates how an operating system should be installed within an organization may have different specializations to address the specific needs of each department. In this example, class *BlockActivity* could be employed to refine a standard procedure to accommodate those specific configuration requirements.

The specialization of plan templates may alternatively serve the purpose of allowing the operator to guide/influence the refinement of change plans into actionable workflows. For example, consider the plan template depicted in Figure 5. Being aware that the Web application requires a Data Base Management System (DBMS), the operator may indicate, in the design of the template, which DBMS should be installed, instead of leaving the decision to the *Change Planner*.

Finally, **generalization** let the operator to start thinking of a specific template (either request or plan) and, after that, generalize it in order to make it suitable for more general contexts. For example, an operator may design a template to perform some change considering the IT scenario of a particular department. The e

.ssence of the change may be latter captured and applied to other departments as well, by means of generalization.

For request templates, the generalization could be performed, for example, by changing the affected CI of an *install* operation, from *Web Application* to simply *SoftwareElement*. In the case of plan templates, finer-grained activities, for example defined within a *BlockActivity*, could be replaced with a more general, larger-grained activity.

V. Implementation and Case Study

Our template-based solution for change management is supported by the ChangeLedge system, a prototypical implementation developed as part of this work. Next, we present an overview of the system and a case study, which has been conducted based on a real-life scenario.

Starting with ChangeLedge, its components *Change Designer* and *Change Planner*, described previously in Section III, have been implemented using the Java programming language. The repositories have been created through the use of the object-relational mapper Hibernate and the MySQL database. In relation to the documents exchanged among the components, XML is used to code those produced by the *Change Designer* and consumed by the *Change Planner.* Their processing is done by means of the Streaming API for XML (StAX). Similarly, the actionable workflows produced by the *Change Planner* are also XML documents, in this case compliant with XML Process Definition Language (XPDL) [8]. After translated into a workflow language, they may be finally executed by a *Deployment System.*

To prove concept and technical feasibility of our proposal, we have conducted an experimental evaluation using the ChangeLedge system. It consisted of the instantiation of a same request template to specify two different RFCs. The RFCs were then associated to a same plan template, which was latter instantiated to meet each RFC objectives. Finally, the preliminary change documents were consumed by the *Change Planner* in order to generate actionable workflows.

The templates used in our case study are shown in Figure 7. The request template expresses the upgrade of services belonging to the portfolio of an Application Service Provider (ASP) customer, whereas the associated plan template represents one among several approaches that could be employed by the ASP to perform this upgrade. The plan entails the migration of the whole system installed at computer X to another computer Y. Note that *Activity 2* reuses, through composition, the same nested template that realizes *Activity 1* of the plan template shown in Figure 5.

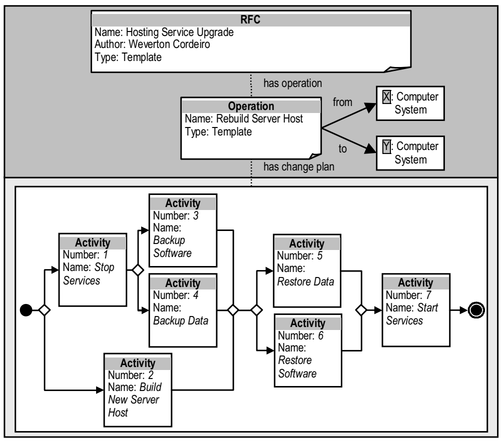

RFC
Name: Hosting Service Upgrade
Author: Weverton Cordeiro
Type: Template
has operation
Operation
Name: Rebuild Server Host
Type: Template
from
to
X: Computer System
Y: Computer System
has change plan
Activity Number: 1 Name: Stop Services
Activity Number: 2 Name: Build New Server Host
Activity Number: 3 Name: Backup Software
Activity Number: 4 Name: Backup Data
Activity Number: 5 Name: Restore Data
Activity Number: 6 Name: Restore Software
Activity Number: 7 Name: Start Services

Fig. 7. Template for the migration of a computing system.

To carry out the experiment, we have modeled an IT infrastructure composed of four computers: A, B, C, and D. Computers A and B were specified as having Windows 2003 Server and GNU/Linux OSs installed, respectively. On top of the former were installed Internet Information Server (IIS), SQL Server 2000, and an ordinary ASP.NET Web app. The latter was setup with Tomcat Web server, MySQL, and a servlet. Computers C and D, however, had nothing installed.

In addition, we have instantiated, through the component *Change Designer*, two change documents using the templates previously described: one to migrate the entire system hosted in computer A (label *from* of the template) to computer C (label *to*) and one to perform the same migration, but from computer B (*from*) to D (*to*). Thereafter, the resulting change documents were submitted to the *Change Planner* in order to compute the corresponding actionable workflows.

A partial view of the actionable workflows generated for each RFC is depicted in Figures 8 and 9. Due to space constraints, only the refinement of *Activity 6* is shown in both figures. These subworkflows orchestrate the installation of all the software packages present in the old computer. The linkage of activities reflects the

dependency among the components to be installed. For example, in Figure 8 Web app depends on IIS and SQL Server 2000; in Figure 9, the servlet depends on MySQL and Tomcat, and Tomcat requires the previous installation of the Java SE Development Kit (JDK).

Our experiments conducted on a computer with AMD Athlon^{tm} XP processor, 2 GHz of CPU clock and 1Gb of RAM memory have showed that our algorithm took 101 and 149 millisseconds to refine the preliminary change plans, respectively. An operator spent 25 and 86 minutes to generate the same refined workflows, using the ActiveBPEL Designer.

Notice that the generated refinements are significantly different. This is explained by the diverse group of affected CIs and *checks*/*actions* involved in their manipulation. For example, the workflow produced for the first change document contains a *reboot* activity, which is required by SQL Server 2000 to complete its installation. The reader may also note that the activities may be either automatically executable (background in white) or require the intervention of a human operator (background in gray).

VI. Conclusions and Future Work

We have discussed in this paper some of the benefits of capturing and reusing IT change knowledge. The lack of a common standard to design changes along with proper tool support to assist this process make such knowledge reuse difficult in practice. To address these issues, we have proposed an original solution based on the concept of *change templates* to ease the formalization and reuse of the experience accumulated within organizations in relation to IT changes. Our solution is supported by ChangeLedge, a prototypical implementation of a change management system.

The results obtained, although not exhaustive, are quite positive. The use of request and plan templates showed to be flexible to allow the design of RFCs and (different levels of) preliminary change plans, for several types of IT changes. The possibility of associating, composing, specializing, and generalizing templates provides the change operators with a powerful mechanism to structure knowledge that otherwise would remain with individuals. In addition, the refinement of change documents has resulted in highly detailed change plans, including aspects not explored in previous investigations (e.g., activities that involve a human operator).

Since our main objective in this paper was to explore formalization and reuse of knowledge in the context of change management, considering a *broad* range of possible IT changes, we have not focused on generating *optimized* change plans. In a future investigation we intend to address this issue, possibly taking advantage of linear optimization, AI planning [10], or workflow/web service composition techniques [11]. Other perspectives for future work include: (i) investigating decision support mechanisms to help operators understand the trade-offs between alternative change designs, and (ii) exploring data mining techniques to create request and plan templates based on change history.

References

[1] ITIL. *Information Technology Infrastructure Library (ITIL)*. Office of Government Commerce (OGC), 2006. http://www.itil.co.uk.

[2] Keller, A.; Hellerstein, J. L.; Wolf, J. L.; Wu, K. L.; Krishnan, V. The CHAMPS System: Change Management with Planning and Scheduling. In *9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004)*, v. 1, pages 395-408, 2004.

[3] IT infrastructure Library. *ITIL Service Support*, version 2.3, Office of Government Commerce, 2000.

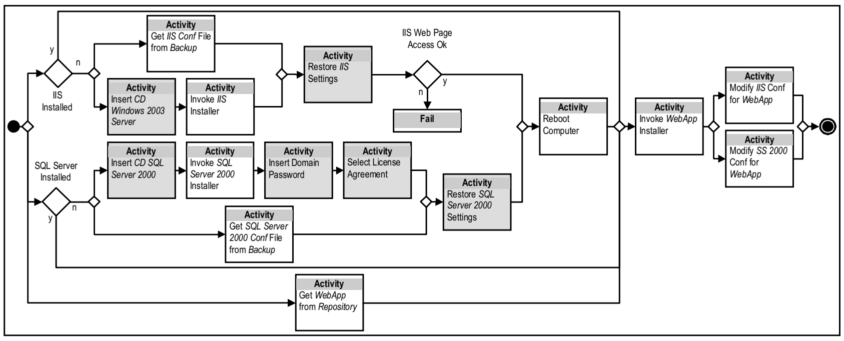


Fig. 8. Partial workflow for the migration of a Windows 2003 Server.

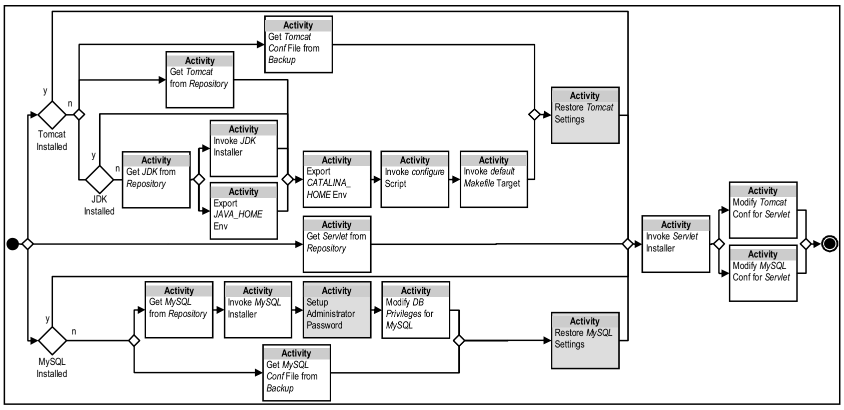


Fig. 9. Partial workflow for the migration of a GNU/Linux system.

[4] Rebouças, R.; Sauvé, J.; Moura, A.; Bartolini, C.; Trastour, D. A Decision Support Tool to Optimize Scheduling of IT Changes. In *10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)*, pages 343-352, 2007.

[5] Shankar, C.; Talwar, V.; Iyer, S.; Chen, Y.; Milojicic, D.; Campbell, R. Specification-enhanced Policies for Automated Management of Changes in IT Systems. In *20th USENIX Large Installation System Administration Conference (LISA 2006)*, 2006.

[6] Keller, A. Automating the Change Management Process with Electronic Contracts. In *7th IEEE International Conference on E-Commerce Technology Workshops*, pages 99 – 107, 2005.

[7] Distributed Management Task Force. Common Information Model. http://www.dmtf.org/standards/cim.

[8] The Workflow Management Coalition Specification. *Workflow Process Definition Interface - XML Process Definition Language*. http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf.

[9] Organization for the Advancement of Structured Information Standards. *Business Process Execution Language, version 2.0.* http://docs.oasis-open.org/wsbpel/2.0.

[10] Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; Yaman, F. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research*, v. 20, pp. 379-404, 2003.

[11] Sirin, E.; Parsia, B.; Hendlera, J. Template-based Composition of Semantic Web Services. In *AAAI Fall Symposium on Agents and the Semantic Web*, 2005.

APPENDIX C PUBLISHED PAPER – DSOM 2008

In this attachment we present the paper entitled “A Runtime Constraint-aware Solution for Automated Refinement of IT Change Plans”, another deliverable of the work described in this thesis. The paper presents an algorithm for the automated refinement of preliminary change plans into actionable workflows. The proposed algorithm takes into account the runtime constraints that emerge during the change plan execution (*e.g.*, lack of disk space and memory exhaustion). As a consequence, the generated change plans will be less prone to be prematurely aborted due to resource constraints. The experimental evaluation conducted shows the feasibility of the proposed algorithm, which is able to generate accurate, complete change plans in a time of lower magniture than would be spent by an experient operator to design the same plans.

- Title: A Runtime Constraint-aware Solution for Automated Refinement of IT Change Plans
- Conference: International Workshop on Distributed Systems: Operations and Management (DSOM 2008)
- URL: http://www.manweek.org/2008/dsom/
- Date: 22-26 September 2008
- Venue: Doryssa Bay Resort, Samos Island, Greece

A Runtime Constraint-Aware Solution for Automated Refinement of IT Change Plans

Weverton Luis da Costa Cordeiro[1], Guilherme Sperb Machado[1], Fabrício Girardi Andreis[1], Alan Diego Santos[1], Cristiano Bonato Both[1], Luciano Paschoal Gaspary[1], Lisandro Zambenedetti Granville[1], Claudio Bartolini[2], and David Trastour[3]

[1]Institute of Informatics, Federal University of Rio Grande do Sul, Brazil
[2]HP Laboratories Palo Alto, USA
[3]HP Laboratories Bristol, UK
{weverton.cordeiro, gsmachado, fgandreis, adsantos, cbboth, paschoal, granville}@inf.ufrgs.br,
{claudio.bartolini, david.trastour}@hp.com

Abstract. Change design is one of the key steps within the IT change management process and involves defining the set of activities required for the implementation of a change. Despite its importance, existing approaches for automating this step disregard the impact that actions will cause on the affected elements of the IT infrastructure. As a consequence, activities that compose the change plan may not be executable, for example, due to runtime constraints that emerge during the change plan execution (*e.g.*, lack of disk space and memory exhaustion). In order to address this issue, we propose a solution for the automated refinement of runtime constraint-aware change plans, built upon the concept of incremental change snapshots of the target IT environment. The potential benefits of our approach are (*i*) the generation of accurate, workable change plans, composed of activities that do not hinder the execution of subsequent ones, and (*ii*) a decrease in the occurrence of service-delivery disruptions caused by failed changes. The experimental evaluation carried out in our investigation shows the feasibility of the proposed solution, being able to generate plans less prone to be prematurely aborted due to resource constraints.

1 Introduction

The increasing importance and complexity of IT infrastructures to the final business of modern companies and organizations has made the Information Technology Infrastructure Library (ITIL) [1] the most important reference for IT service deployment and management. In this context, ITIL's best practices and processes help organizations to properly maintain their IT services, being of special importance to those characterized by their large scale and rapidly changing, dynamic services.

Among the several processes that compose ITIL, *change management* [2] plays an important role in the efficient and prompt handling of IT changes [3]. According to this process, changes must be firstly expressed by the *change initiator* using *Requests for Change* (RFC) documents. RFCs are declarative in their nature, specifying what

© IFIP International Federation for Information Processing 2008

should be done, but not expressing how it should be performed. In a subsequent step, an *operator* must sketch a preliminary *change plan*, which encodes high level actions that materialize the objectives of the RFC. Latter steps in this process include *planning*, *assessing and evaluating*, *authorizing and scheduling*, *plan updating*, *implementing*, and *reviewing and closing* the submitted change.

Change *planning*, one of the key steps in this process, consists in refining, either manually or automatically, the preliminary plan into a detailed, actionable workflow (also called actionable change plan in this paper). Despite the possibility of manually refining change plans, automated refinement has the potential to provide better results for the planning phase, since it (*i*) decreases the time consumed to produce such actionable workflows, (*ii*) captures the intrinsic dependencies among the elements affected by changes, and (*iii*) diminishes the occurrence of service disruptions due to errors and inconsistencies in the generated plans [4].

Since the inception of ITIL, there has been some preliminary research concerning the automated refinement of change plans. For example, important steps have been taken towards formalizing change-related documents [5], exploring parallelism in the execution of tasks [3], and scheduling of change operations considering the long-term impact on *Service Oriented Architecture* environments [6]. However, despite the progresses achieved in the field, proposed solutions for change planning only consider simple actions (installation, upgrade) and do not model the pre-conditions and effects of more complex actions. The pre-conditions could be of a technical nature, such as a memory requirement, or could impose constraints on the change process, for instance requiring authorization before executing a given task. Effects model how actions modify each element of the IT infrastructure (*e.g.*, adding memory into a server or modifying configuration parameters of a J2EE server). Without taking into account such considerations, the actionable workflow, when executed, may be prematurely aborted (*e.g.*, due to lack of resources), leading to service-delivery disruption and leaving the IT infrastructure in an inconsistent state.

To fill in this gap, we propose a solution for the automated refinement of change plans that takes into consideration the runtime constraints imposed by the target IT environment. In contrast to previous investigations, our solution focuses on the impact that already computed actions will cause on the IT infrastructure, in order to compute the subsequent ones. To this effect, we introduce in this paper the notion of *snapshots* of the IT infrastructure, as representations of the intermediate states that the IT infrastructure would reach throughout the execution of the change plan. As a result, the refined change plans generated by our solution will be less prone to prematurely termination, therefore reducing the occurrence of change-related incidents.

The solution proposed in this paper is evaluated through the use of CHANGELEDGE, a prototypical implementation of a change management system that enables the design, planning and implementation of IT changes. We have qualitatively and quantitatively analyzed the actionable workflows generated from several different preliminary plans, considering a typical IT scenario.

The remainder of this paper is organized as follows. Section 2 discusses some of the most prominent research in the field of IT change management. Section 3 briefly reviews the models employed to represent IT related information. Section 4 details our runtime constraint-aware solution for the automated refinement of IT change plans. Section 5 presents the results achieved using the CHANGELEDGE system. Finally, Section 6 concludes the paper with remarks and perspectives for future work.

2 Related Work

In the recent years, several research efforts have been carried out in the area of IT change design. In this section, we cover some of the most prominent investigations.

Keller *et al.* [3] have proposed CHAMPS, a system for automating the generation of change plans that explore a high degree of parallelism in the execution of tasks. Change planning and scheduling are approached as an optimization problem. Although the system is able to evaluate technical constraints in the planning and scheduling of changes, the scope is limited to Service Level Agreements and policies. Since fine-grained control of resource constraints was not the focus of the work, modifications on the infrastructure produced by the already processed tasks of the plan under refinement are not taken into account when computing the subsequent ones. As a consequence, the resulting change plans may not be executable in practice.

In a previous work [5], we have proposed a solution to support knowledge reuse in IT change design. Although the solution comprises an algorithm to generate actionable change plans, this algorithm also performs all the computations considering a static view of the IT infrastructure. Actually, it was out of the scope of that work, as a simplification assumption, to deal with runtime constraints in the refinement of change plans.

Despite not directly related with the problem addressed in this paper, some additional research efforts on change management published in the recent years merit attention. Dumitraş *et al.* [6] have proposed *Ecotopia*, a framework for change management that schedules change operations with the goal of minimizing service-delivery disruptions. In contrast to CHAMPS, Ecotopia optimizes scheduling by assessing the long-term impact of changes considering the expected values for *Key Performance Indicators*. Trastour *et al.* [7] have formulated the problem of assigning changes to maintenance windows and of assigning change activities to technicians as a mixed-integer program. The main difference between this work and Ecotopia is the fact that human resources are also taken into account. Sauvé *et al.* [8] have proposed a method to automatically assign priorities to changes, considering the individual exposure of each requested change to risks as its execution is postponed. Finally, in another previous work [9], we have introduced the concept of *atomic groups* in the design of change plans with the purpose of providing our end-to-end solution to IT change management with rollback support.

Although change management is a relatively new discipline, the area has been quickly progressing, as evidenced by the previously mentioned related work. Nevertheless, in the particular case of change planning, the existing solutions are severely lacking with respect to deployment feasibility and IT infrastructure predictability. In the following sections we envisage a solution to address these issues.

3 Building Blocks of the Proposed Solution

In order to support the automated refinement of change plans, it is of paramount importance to formalize the change-related documents. Actually, this was a major concern in our previous work [5], in which we proposed models to (*i*) characterize dependencies between the elements that compose the IT infrastructure, (*ii*) express information about software packages available for consumption by a change process, and (*iii*) express unambiguously the changes that must be executed on the managed infrastructure. In this section, we briefly review the models that materialize this formalization: *IT infrastructure* and *Requests for Change & Change Plan*.

The *IT Infrastructure* model is a subset of the Common Information Model (CIM) [10], proposed by the Distributed Management Task Force (DMTF). It allows the representation of computing and business entities comprising an organization, as well as the relationship among them. For the sake of legibility and space constraints, we present in Fig. 1 a partial view of the model.

The root class *ManagedElement* permits to represent any *Configuration Item* (CI) present in the IT infrastructure (*e.g.*, physical devices, computer and application systems, personnel, and services). Relationships such as associations, compositions, and aggregations, map the dependencies among the elements comprising the infrastructure. In addition, *Check* and *Action* classes in this model represent relevant information for managing the lifecycle of *software elements* (*e.g.*, software upgrade and application system installation/uninstallation).

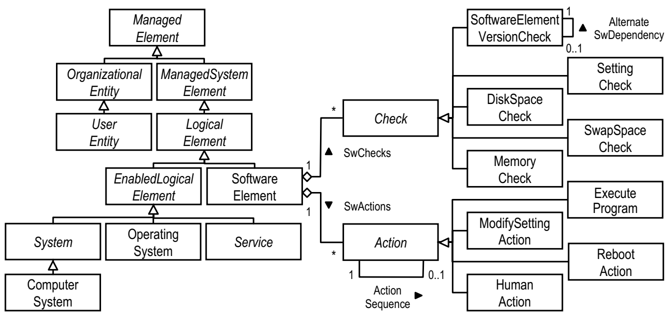


Fig. 1. Partial view of the IT Infrastructure model

Instances of class *Check* define conditions to be met or characteristics required by the associated software element for it to evolve to a new state (*e.g.*, *deployable*, *installable*, *executable*, or *running*). Possible checks include verification of software dependencies, available disk space and memory, and required environment settings. Each instance of class *Action*, in its turn, represents an operation of a process to change the state of the associated *SoftwareElement* (*e.g.*, from *installable* to *executable*). Examples of actions are invocation of a software installer/uninstaller, manipulation of files and directories, and modification of configuration files.

In addition to being used to represent the current IT infrastructure, the same model is also employed to define the *Definitive Media Library* (DML). The DML is a repository that specifies the set of software packages (along with their dependencies) that have been approved for use within the enterprise and that may be required throughout the change process.

In regard to the *Requests for Change & Change Plan* model, it enables the design of change-related documents and relies on both (*i*) guidelines presented in the ITIL Service Transition book [2], and (*ii*) the workflow process definition, proposed by the Workflow Management Coalition (WfMC) [11]. Classes such as *RFC* and *Operation* allow expressing the changes designed by the *change initiator*, while *ChangePlan*, *LeafActivity*, *BlockActivity*, *SubProcessDefinition*, and *TransitionInformation* enable the *operator* to model the preliminary plan that materializes the change. Please refer to our previous work [5] for additional information about this model.

4 Runtime Constraint-Aware Refinement of Change Plans

The models presented in the previous section represent the common ground for our runtime constraint-aware solution for automated refinement of IT change plans. In this section, we describe our solution by means of a conceptual algorithm, illustrated in Fig. 2.

In order to support our solution, we formalize a change plan C, in the context of this work, as a 4-tuple $\langle A, T, a_1, F\rangle$, where A represents the set of activities (or actions) $A = \{a_1, a_2, \ldots, a_n \mid n \in N \text{ and } n \geq 1\}$; T represents a set of ordered pairs of activities, called transitions, $T = \{l_1, l_2, \ldots, l_m \mid m \in N \text{ and } m \geq 1\}$; a_1 is the begin activity of the change plan ($a_1 \in A$); and F represents the set of end activities of the change plan ($F \subseteq A$). A transition $l = (a_i, a_j) \in T$ is directed from a_i to a_j, $\forall a_i, a_j \in A$, and may represent a conditional flow.

We denote our solution as a function $f(C, I, R) = C'$ (line 1), where C is the preliminary change plan; I represents the state of the IT infrastructure as in the instant in which the preliminary plan C is submitted for refinement; R represents the *Definitive Media Library* (DML); and C' represents the actionable workflow generated as a result of the refinement process.

As a first step towards the refinement, the submitted plan C is copied to C' (line 2), and the subset of unrefined activities contained in C is copied to A' (line 3). In a subsequent step (line 4), f creates an initial *snapshot* of the IT infrastructure, s_0. In the context of this work, we define snapshot as a representation of the differences between the current state of the IT infrastructure and the state it would reach after the execution of i activities contained in the change plan C ($0 \leq i \leq |A|$). These differences include, for example, newly installed (or removed) software, disk space and memory consumed (or freed), modified settings, and created (or deleted) files and directories (the dynamics of snapshots is further explained in Subsection 4.2). Considering that no new activities were added to the change plan C at the point s_0 is created, this step will yield a snapshot that describes no differences in comparison to the current state of the IT infrastructure.

As a last step, f invokes the execution of $f'(C', R, I, s_0, A')$ (line 5), which will actually perform the refinement process. We assume that C' is passed to f' by reference. Therefore, modifications performed to C' will be visible outside f'. After the execution of f', C' will be returned back to the *operator* (line 7), if refined (line 6). We consider a change plan C as refined if and only if, $\forall a \in A$, dependencies of a are already satisfied either by any $a_i \in A$ or by the current state of the IT infrastructure.

```
1   f(C, R, I) = C' {
2      declare C' = copy of the preliminary change plan C
3      declare A' = set of unrefined activities from the preliminary change plan C
4      s_0 = initial snapshot of I, after the execution of 0 activities
5      f'(C', R, I, s_0, A')
6      if (C' is refined)
7         return C'
8      else
9         return false
10  }
11
12  f'(C, R, I, s_i, A) {
13     if (A is empty)
14        return change plan C
15     else {
16        declare X: set of arrangements Y of activities
17        a_i = i-est activity ∈ A
18        declare A' = A - {a_i}
19        if (a_i has no computable dependencies, given I, s_i, and R)
20           f'(C, R, I, s_i, A')
21        else {
22           X = set of arrangements Y of first level dependencies of a_i, given I, s_i, and R
23           for each Y_i ∈ X {
24              declare C' = C + Y_i
25              declare A'' = A' ∪ Y_i
26              s_{i+1} = new snapshot of the IT infrastructure I, given C', I, and s_i
27              f'(C', R, I, s_{i+1}, A'')
28           }
29        }
30     }
31  }
```

Fig. 2. Conceptual algorithm for runtime constraint-aware refinement of change plan

In case the plan returned by f' is not refined, the operator will receive a negative feedback (line 9). This feedback will mean that an actionable and executable workflow (for the preliminary plan C submitted) could not be achieved. Having this feedback, the operator could reformulate and resubmit the preliminary plan, therefore starting the refinement process over again.

Having presented a general view of our solution, in the following subsections we describe in more detail the recursive search for a refined change plan, and the concept of snapshots of the IT infrastructure.

4.1 Refinement of the Preliminary Change Plan

Function f' solves the problem of modifying the received preliminary plan C into an actionable workflow by using the *backtracking* technique [12]. This technique permits exploring the space of possible refinements for C, in order to build a refined plan that meets IT resource constraints. Fig. 3 illustrates the execution of f' using a simplified example. For the sake of clarity, only two levels of recursion are presented.

The preliminary plan C in Fig. 3 materializes an RFC to install an e-Commerce Web application, and is composed of the task *Install WebApp*. This task represents a *BlockActivity* derived from the set of actions necessary to install *WebApp* (arrow 1 in Fig. 3). The first verification performed by f' (line 13 in Fig. 2) is whether A, the set of activities that remain unrefined in the received plan C, is empty or not. If A is empty, C is returned back to f. Considering the example in Fig. 3, f' will receive in its first invocation (line 5) the set A' = {*Install WebApp*}.

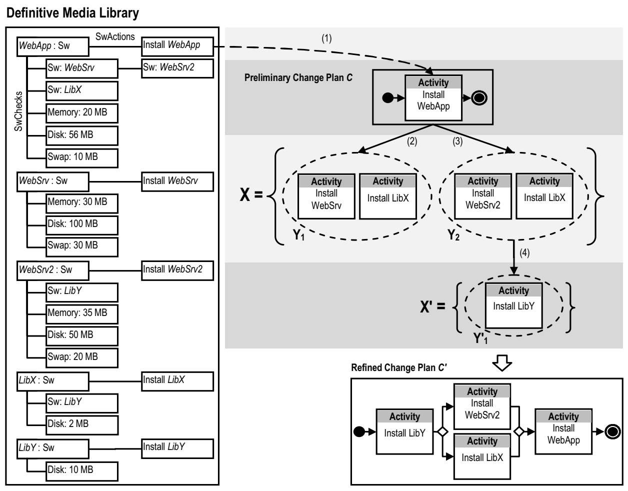


Fig. 3. Illustration of the functioning of f'

The algorithm f' starts by extracting an activity a_i from A (line 17), generating a new set A' (which contains all activities in A except a_i) (line 18). In our example, a_i is the activity *Install WebApp*, and the resulting A', an empty set. Subsequently, f' tests whether a_i has computable dependencies (line 19). An activity is said to have computable dependencies if: (*i*) the *Configuration Item* (CI) modified by a_i has checks (*SwChecks*) mapped in the DML and/or relationships in the IT repository (*e.g.*, shutting down service *Service*$_1$ requires shutting down *Service*$_2$ and bringing up *Service*$_3$), and (*ii*) the aforementioned dependencies (or checks) are not yet fulfilled in neither the current state of the IT infrastructure nor the current snapshot.

If a_i has no computable dependencies (*i.e.*, if all pre-conditions for the execution of a_i are already satisfied in either the IT or the current snapshot), f' invokes itself recursively (line 20), in order to refine another activity of the resulting A'. Otherwise, f' computes the set of arrangements of *immediate dependencies* (or *first level dependencies*) that (*i*) fulfill the pre-conditions for the execution of a_i, and (*ii*) would be executable in the current snapshot (considering the requirements of these arrangements). The arrangements returned from this step will be stored in X (line 22). In this set, Y_i represents each of the arrangements.

In our example, *Install WebApp* has two computable dependencies described in the DML: a web server (either *WebSvr* or *WebSrv2*) and a generic library (*LibX*). Therefore, the computation of X (line 22) yields a set containing two arrangements of possible immediate dependencies for a_i. The first is Y_1 = {*Install WebSrv*, *Install LibX*}, and the second is Y_2 = {*Install WebSrv2*, *Install LibX*}.

After that, f' searches for an arrangement Y_i in X that leads to a refined change plan (line 23). Although more than one Y_i may lead to a solution, the first Y_i to be tested will compose the refined plan. Considering the example, the first set tested was Y_1 (arrow 2 in Fig. 3), while the second was Y_2 (arrow 3).

The aforementioned test performed to an arrangement Y_i comprises four steps. First, a new change plan C' is created, by adding the activities in Y_i to C (line 24). Second, a new set of unrefined activities A'' is built, as a result of the union of the sets A' and Y_i (line 25). This is necessary because activities in Y_i may not be refined yet, therefore requiring a future processing. Third, the impact of running activities in Y_i is computed (line 26), considering both the current view of the IT infrastructure (from I) and the changes performed so far (materialized in the snapshot s_i). The result will be stored in the snapshot s_{i+1} (in our example, s_1 represents an incremental view of the snapshot s_0, after the execution of *Install WebSrv*, *Install LibX*, and *Install WebApp*). Finally, f' is invoked recursively to refine C'', given the newly computed A'' and s_{i+1} (line 27).

Observe that the addition of the activities in Y_i to the change plan C' (line 24) takes into account dependency (pre-requisite) information. In our example, since Y_1 = {*Install WebSrv*, *Install LibX*} is a set of dependencies of *Install WebApp* (*i.e.*, *Install WebSrv* and *Install LibX* must be executed prior to *Install WebApp*), adding these activities to C'' implies in the creation of the transitions l_i = (*Install WebSrv*, *Install WebApp*) and l_{i+1} = (*Install LibX*, *Install WebApp*), and subsequent addition of l_i and l_{i+1} to the set of transitions T of the change plan C''.

Putting all the pieces together, recursive invocations of f' is the mechanism employed to navigate through all paths in the *activity dependency tree* (which represents the dependencies between software packages captured from the DML). From the example illustrated in Fig. 3, in the first invocation to f' (line 5) the activity *Install WebApp* is processed. In the first-level recursion (arrow 2 in Fig. 3) of f' (line 27), the set of immediate dependencies Y_1 is tested. Once the test fails, the recursion returns, and then the set Y_2 is tested (arrow 3). This yields a new first-level recursion (line 27). Once the test to Y_2 is successful, a second-level recursion is performed, now to process the set Y = {*Install LibY*} (arrow 4). Since *Install LibY* has no computable dependencies, a third-level recursion of f' is performed (line 20). Finally, given that there are no dependencies left to refine, the recursive refinement is finished, and the resulting refined plan C' (Fig. 3) is returned back to f' (line 14).

4.2 Snapshots of the IT Infrastructure

The concept of *snapshot* is the notion upon which the recursive search for a refined change plan is built. Having the current snapshot s_i, the refinement algorithm may foresee the new state of the IT infrastructure after the execution of the actions already computed and present in the change plan C. Consequently, it will be able to identify dependencies that are executable, and then continue the refinement process.

Fig. 4 illustrates the snapshots that are created during the refinement process of our example. In this figure, CS stands for *computer system*, OS for *operating system*, and SwElement for *software element*. The initial snapshot in our example is s_0. The two arrows from s_0 represent two possible state transitions of the IT infrastructure after the execution of each of the arrangements returned for activity *Install WebApp*. The first transition (arrow 1 in Fig. 4) leads to snapshot s_{1a}, which represents the state after the execution of (the activities in) Y_1 plus *Install WebApp*. The second transition (arrow 2), on the other hand, leads to s_{1b}, which represents the state after the execution of Y_2 (plus *Install WebApp*). The dashed arrow from s_{1a} to s_0 represents the failed test made with Y_1 (in this case, f' goes back to the previous snapshot and attempts another arrangement of immediate dependencies contained in X, Y_2). Finally, the transition from snapshot s_{1b} to s_2 (arrow 3) represents the second-level recursion to f', when the activity *Install LibY* is added to the partially refined plan C.

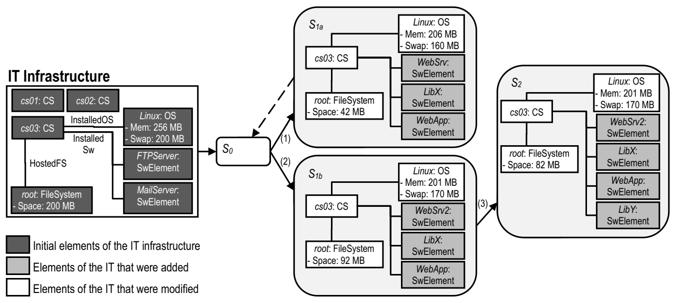

IT Infrastructure
cs01: CS
cs02: CS
cs03: CS
InstalledOS
Installed Sw
HostedFS
Linux: OS - Mem: 256 MB - Swap: 200 MB
FTPServer: SwElement
MailServer: SwElement
root: FileSystem - Space: 200 MB
S0
(1)
(2)
(3)
S1a
Linux: OS - Mem: 206 MB - Swap: 160 MB
cs03: CS
WebSrv: SwElement
LibX: SwElement
WebApp: SwElement
root: FileSystem - Space: 42 MB
S1b
Linux: OS - Mem: 201 MB - Swap: 170 MB
cs03: CS
WebSrv2: SwElement
LibX: SwElement
WebApp: SwElement
root: FileSystem - Space: 92 MB
S2
Linux: OS - Mem: 201 MB - Swap: 170 MB
cs03: CS
root: FileSystem - Space: 82 MB
WebSrv2: SwElement
LibX: SwElement
WebApp: SwElement
LibY: SwElement
Initial elements of the IT infrastructure
Elements of the IT that were added
Elements of the IT that were modified

Fig. 4. Evolution of the snapshots as the change plan is refined

Considering the representation of differences, the snapshots in Fig. 4 hold information about consumed resources and new settings present in the environment. For example, the reader may note that after the execution of activities in Y_2 and *Install WebApp*, the IT infrastructure would evolve to a new state, represented by s_{1b}. In this new state, the computer system *cs03* (*i*) has 108 MB less disk space available, and (*ii*) has the newly installed *SoftwareElements WebSrv2*, *LibX*, and *WebApp*.

Also observe that installing new software in a computer potentially increases the demand for more available physical memory (in the case of *cs03*, 55 MB more physical memory and 30 MB more swap space). Although the use of memory and swap space is flexible, the amount of such resource available for use imposes a limit, in terms of performance, in the software that may be running concurrently.

It is important to mention that the scope of the proposed snapshots is restricted to the change planning step. In addition, the information they hold is useful for the proposed refinement solution only. As a consequence, they do not take place in other phases of the change management process (*e.g.*, change testing or implementation).

4.3 Considerations on the Proposed Solution

According to the change management process, there are intermediate steps between the *design* and the actual *implementation* of a change. These steps are *assessment and evaluation*, *authorization and schedule*, and *plan updates*. The time scale to go through them may range from hours to days (or even weeks). During this period, the

IT infrastructure may evolve to a new, significantly different state (for example, due to other implemented changes). In this context, the runtime constraint-aware plan generated by our solution may not be executable upon implementation. This issue (that has been long associated with the change management process) may be tackled during the *plan updates* phase. The operator may either manually adjust the plan for the new IT scenario or re-invoke the proposed algorithm, and document the revised plan afterwards. From this point on, the time gap to implement the change should be kept to a minimum.

Another important aspect worth discussing is the *refinement flexibility* provided to the algorithm. This is regulated by the degree of detail of the preliminary plan submitted. A loosely defined preliminary plan tends to allow the algorithm to perform a broader search within the activity dependency tree. Consider, for example, an RFC to install a certain web-based application. Assuming this application depends on a Database Management System (DBMS), the operator may explicitly specify in the preliminary plan the DBMS to be installed or leave it up to the algorithm. In the latter case, the choice will be based on the alternative database packages available in the Definitive Media Library and on the runtime constraints.

To deal with the aforementioned flexibility, one could think of the existence of an *automated decision threshold*. This threshold could be specified in terms of number of software dependency levels. During the refinement process, dependencies belonging to a level above the configured threshold would be decided by the operator in an interactive fashion. Otherwise, the algorithm would do this on his/her behalf. Evaluating the pros and cons of setting a more conservative or liberal strategy is left for future work.

5 Experimental Evaluation

To prove the conceptual and technical feasibility of our proposal, we have (*i*) implemented our solution on top of the CHANGELEDGE system [5], and (*ii*) conducted an experimental evaluation considering the design and refinement of changes typically executed in IT infrastructures. Due to space constraints, we focus our analysis on five of these changes. As a result of the refinement of preliminary plans into actionable workflows, we have observed the correctness and completeness of the produced workflows (characterizing a more *qualitative* analysis of the proposed solution), in addition to performance indicators (*quantitative* analysis).

The IT infrastructure employed is equivalent to the environment of a research & development department of an organization. It is composed of 65 workstations, located in seven rooms, running either *Windows XP SP2* or *GNU/Linux*. The environment is also composed of four servers, *Server*$_1$, *Server*$_2$, *Server*$_3$, and *Server*$_4$, whose relevant settings to the context of our evaluation are presented in Table 1. Finally, the content of the Definitive Media Library is summarized in Table 2.

Table 1. Server settings

Server Name	Installed Operating System	Available Disk Space	Total Physical Memory
Server$_1$	None	20,480 MB	2,048 MB
Server$_2$	Windows 2003 Server	71,680 MB	4,096 MB
Server$_3$	Debian GNU/Linux	51,200 MB	4,096 MB
Server$_4$	Debian GNU/Linux	102,400 MB	4,096 MB

Table 2. System requirements for the software present in the DML [1]

Software Name	Disk Space	Memory	Software Dependencies
e-Commerce Web App[2]	512 MB	128 MB	SQL Server and Internet Information Server (IIS)
IIS 5.1	15 MB	16 MB	Windows XP Service Pack 2 (Win XP SP2)
IIS 7.0	15 MB	16 MB	Windows Vista Service Pack 1 (Win Vista SP1)
.Net Framework 3.5	280 MB	256 MB	Internet Explorer (IE), IIS, and Win XP SP2
SQL Server 2005	425 MB	512 MB	IE, Win XP SP2, and .Net Framework
SQL Server 2008	1,460 MB	1,024 MB	IE and Win Vista SP1
IE 7	64 MB	128 MB	Win XP SP2
Windows XP SP 2	1,800 MB	-	Windows XP
Windows Vista SP 1	5,445 MB	-	Windows Vista
Windows XP	1,500 MB	128 MB	-
Windows Vista	15,000 MB	1,024 MB	-

In regard to the submitted RFCs, the first two have as objective the installation of an e-Commerce web application (*WebApp*), one of them having *Server*$_1$ as target CI and the other, *Server*$_3$. The third RFC comprises two operations: one to install and configure a network monitoring platform on *Server*$_4$, and the other to install and configure an authentication server on *Server*$_3$. The fourth RFC comprises the migration of the entire system installed on *Server*$_3$ to *Server*$_4$. Finally, the fifth RFC consists in updating software packages installed in 47 out of the 65 stations that compose the IT infrastructure (typical procedure in several organizational contexts).

A partial view of the actionable workflow generated from the first RFC is presented in Fig. 5. Decision structures within the workflow were omitted for the sake of legibility. Observe that the linkage between the activities present in the workflow reflect the dependencies between the installed packages. For example, the e-Commerce Web application depends on services provided by the *SQL Server 2005* Database Management System and *Internet Information Server 5.1*. *SQL Server 2005*, in its turn, depends on the previous installation of the *.Net Framework 3.5*.

The reader may also note that implementing this actionable workflow requires, considering the information in Table 2, about 4,596 MB of disk space, and a minimum of 1,168 MB of available physical memory, from *Server*$_1$. Since this server has sufficient disk space for the installation procedures present in the workflow, the implementation of this RFC is likely to succeed. Moreover, all the installed software should execute normally, given that the target server has sufficient physical memory.

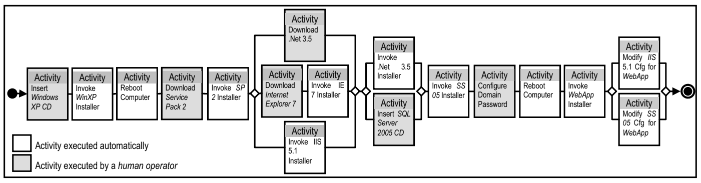

Activity Insert Windows XP CD
Activity Invoke WinXP Installer
Activity Reboot Computer
Activity Download Service Pack 2
Activity Invoke SP 2 Installer
Activity Download .Net 3.5
Activity Download Internet Explorer 7
Activity Invoke IE 7 Installer
Activity Invoke IIS 5.1 Installer
Activity Invoke .Net 3.5 Installer
Activity Insert SQL Server 2005 CD
Activity Invoke SS 05 Installer
Activity Configure Domain Password
Activity Reboot Computer
Activity Invoke WebApp Installer
Activity Modify IIS 5.1 Cfg for WebApp
Activity Modify SS 05 Cfg for WebApp
Activity executed automatically
Activity executed by a human operator

Fig. 5. Partial view of the actionable workflow for the installation of *WebApp*

[1] Source: http://www.microsoft.com

[2] The e-Commerce Web Application system requirements were estimated.

An alternative plan to the one present in Fig. 5 is the one in which *SQL Server 2008* is installed instead of *SQL Server 2005*, and *Internet Information Server 7.0*, instead of *IIS 5.1*. As a consequence, *Windows Vista* and *Windows Vista Service Pack 1* would be installed as well, instead of *Windows XP Service Pack 2* and *Windows XP*, due to the pre-requisite information. For the same reason, the installation of *.Net Framework 3.5* would not be present in this alternative plan. This plan would require 22,496 MB of available disk space from $Server_1$ to be executable, amount beyond the 20,480 MB currently available. Therefore, it would not be generated by our solution, since it is impractical considering the imposed resource constraints.

Table 3. Complexity of the change scenarios considering the number of activities and affected configuration items (pre and post refinement)

Scenario	Preliminary plan				Refined plan			
	Activities	Affected Stations	Affected OSes	Affected Software	Activities	Affected Stations	Affected OSes	Affected Software
1	1	1	0	1	19	1	1	6
2	1	1	0	1	23	1	1	22
3	4	2	0	2	30	2	1	26
4	46	3	0	5	182	3	1	47
5	235	47	0	6	613	47	2	29

Table 3 presents, synthetically, the computational processing spent by the ChangeLedge system to refine and generate actionable workflows for the five RFCs. We highlight Table 3 the number of activities, as well as the number of computer systems (stations), operating systems, and software affected in both the preliminary (specified by a human operator) and refined plans (generated by the system). Taking scenario 4 as example, one may note that the final change plan has 182 activities, automatically refined from a 40% smaller plan.

The performance of the ChangeLedge system to generate the actionable workflows characterized above is presented in Table 4. Our experiments were conducted on a computer equipped with a Pentium^{tm} Centrino processor, 1.7 GHz of CPU clock, 2,048 KB of cache, and 512 MB of RAM memory. The system has performed satisfactorily, demanding from a few hundreds of milliseconds (544) to a few dozens of seconds (57) to generate the aforementioned plans. We have also calculated a confidence interval of 95% for the measured times, considering 10 repetitions of the refinement process for each change document. As shown in Table 4, we expect the refinement time to vary minimally, for each scenario. The results show that our solution not only generates complete and correct plans, but has potential to reduce, in a significant way, time and efforts demanded to this end.

Table 4. Refinement processing time

Scenario	Refinement Time (ms)	Confidence Interval of the Refinement Time	
		Lower Bound (ms)	Upper Bound (ms)
1	544	535	552
2	942	937	947
3	1,754	1,736	1,771
4	3,879	3,811	3,947
5	57,674	57,482	57,866

6 Conclusions and Future Work

Change design is an undoubtedly fundamental building block of the IT change management process. However, existing computational solutions to help the generation of consistent, actionable change plans are still maturing and need more work so as to eliminate some usual simplification assumptions. In this paper, we have proposed a solution to automate the generation of change plans that take into account runtime resource constraints. This is a very important aspect to be considered in order to compute feasible plans, *i.e.*, plans in which no technical or human resource constraint is going to be violated during the execution of the plan.

The obtained results, although not exhaustive, were quite positive. The actionable workflows generated automatically from preliminary plans (designed by human operators) have respected the restrictions imposed by the target environment (*e.g.*, memory and disk space constraints). Furthermore, the refinement of change plans ran on the order of hundreds of milliseconds to dozens of seconds. This time is certainly of lower magnitude than the time that would be required by an experienced operator to accomplish the same task.

As future work we intend to investigate decision support mechanisms to help operators understand the trade-offs between alternative change designs. In addition, since our problem of IT change design concerns the realization of action sequences from a description of the goal and an initial state of the IT environment, we plan to explore how IT change design can take advantage of AI planning techniques [13]. There may be techniques from this field that our approach could benefit from, whether they are on the topic of knowledge representation, planning algorithms, or the integration of planning and scheduling.

References

1. Information Technology Infrastructure Library. Office of Government Commerce (OGC) (2008), `http://www.itil-officialsite.com`
2. IT Infrastructure Library: ITIL Service Transition, version 3. London: The Stantionery Office, p. 270 (2007)
3. 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, vol. 1, pp. 395–408, 19–23 (2004)
4. Oppenheimer, D., Ganapathi, A., Patterson, D.A.: Why do internet services fail, and what can be done about it? In: 4th Usenix Symposium on Internet Technologies and Systems, Seattle, USA (2003)
5. Cordeiro, W., Machado, G., Daitx, F., et al.: A Template-based Solution to Support Knowledge Reuse in IT Change Design. In: IFIP/IEEE Network Operations and Management Symposium, Salvador, Brazil, pp. 355–362 (2008)
6. Dumitraş, T., Roşu, D., Dan, A., Narasimhan, P.: Ecotopia: An Ecological Framework for Change Management in Distributed Systems. In: de Lemos, R., Gacek, C., Romanovsky, A. (eds.) Architecting Dependable Systems IV. LNCS, vol. 4615, pp. 262–286. Springer, Heidelberg (2007)

7. Trastour, D., Rahmouni, M., Bartolini, C.: Activity-based scheduling of IT changes. In: First ACM International Conference on Adaptive Infrastructure, Network and Security, Oslo, Norway
8. Sauvé, J., Santos, R., Almeida, R., Moura, A.: On the Risk Exposure and Priority Determination of Changes in IT Service Management. In: Distributed Systems: Operations and Management, San José, CA, pp. 147–158 (2007)
9. Machado, G., Cordeiro, W., Daitx, F., et al.: Enabling Rollback Support in IT Change Management Systems. In: IFIP/IEEE Network Operations and Management Symposium, Salvador, Brazil, pp. 347–354 (2008)
10. Distributed Management Task Force: Common Information Model, `http://www.dmtf.org/standards/cim`
11. The Workflow Management Coalition Specification: Workflow Process Definition Interface - XML Process Definition Language, `http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf`
12. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, McGraw-Hill (2001) ISBN 978-0-262-53196-2
13. Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J., Wu, D., Yaman, F.: SHOP2: An HTN Planning System. Journal of Artificial Intelligence Research 20, 379–404 (2003)

APPENDIX D PUBLISHED PAPER – IM 2009

In this attachment we present the paper entitled “ChangeMiner: A Solution for Discovering IT Change Templates from Past Execution Traces”, a deliverable recently accepted for publication in conference proceedings. The paper proposes the mechanism, inspired on process mining techniques, to discover change templates from past execution traces. Built upon the notion of change templates, the mechanism focuses on generating templates that capture the essence of previously executed change processes. The generated templates, in turn, enable the reuse and applicability of such processes in scenarios having settings diverse from the original ones (*e.g.*, different dependency relationships among affected IT components, or specific system requirements). The experimental evaluation, conducted using synthetically generated change traces, shows the potentialities of using change traces in the design and planning of IT changes.

- Title: ChangeMiner: A Solution for Discovering IT Change Templates from Past Execution Traces
- Conference: IFIP/IEEE International Symposium on Integrated Network Management (IM 2009)
- URL: http://www.ieee-im.org/2009/
- Date: 1-5 June 2009
- Venue: Hofstra University, Long Island, NY, USA

ChangeMiner: A Solution for Discovering IT Change Templates from Past Execution Traces

Weverton Luis da Costa Cordeiro,
Guilherme Sperb Machado, Fabrício Girardi Andreis,
Juliano Araújo Wickboldt, Roben Castagna Lunardi,
Alan Diego dos Santos, Cristiano Bonato Both,
Luciano Paschoal Gaspary,
Lisandro Zambenedetti Granville
Institute of Informatics
Federal University of Rio Grande do Sul, Brazil

{weverton.cordeiro, gsmachado, fgandreis,
jwickboldt, rclunardi, adsantos,
cbboth, paschoal, granville}@inf.ufrgs.br

David Trastour[1],
Claudio Bartolini[2],
[1]HP Laboratories Bristol, UK
[2]HP Laboratories Palo Alto, USA

{david.trastour,
claudio.bartolini}@hp.com

Abstract– **The main goal of change management is to ensure that standardized methods and procedures are used for the efficient and prompt handling of changes in IT systems, in order to minimize change-related incidents and service-delivery disruption. To meet this goal, it is of paramount importance reusing the experience acquired from previous changes in the design of subsequent ones. Two distinct approaches may be usefully combined to this end. In a top-down approach, IT operators may manually design change templates based on the knowledge owned/acquired in the past. Considering a reverse, bottom-up perspective, these templates could be discovered from past execution traces gathered from IT provisioning tools. While the former has been satisfactorily explored in previous investigations, the latter – despite its undeniable potential to result in accurate templates in a reduced time scale – has not been subject of research, as far as the authors are aware of, by the service operations and management community. To fill in this gap, this paper proposes a solution, inspired on process mining techniques, to discover change templates from past changes. The solution is analyzed through a prototypical implementation of a change template miner subsystem called ChangeMiner, and a set of experiments based on a real-life scenario.**

I. Introduction

The importance of Information Technology Service Management (ITSM) has increased significantly in the recent years, mainly due to the ever-growing importance of IT to support the business of modern companies and organizations [3]. In this context, the IT Infrastructure Library (ITIL) [1] has become an important and widely accepted best practice collection for ITSM, helping organizations to properly maintain their IT resources and services.

In order to deal with changes in IT systems, ITIL defines the *change management* process [2]. This process has the goal of maximizing the value of changes to business, by means of minimizing change-related incidents and service-delivery disruption. To this end, change management recommends the use of standardized methods and procedures for efficiently and promptly handling all change requests.

Following ITIL's definition of change management, the *change initiator* should express required changes in documents called *Requests for Change* (RFC). RFCs must be accompanied by a *change plan*, designed by an IT *Operator*, which specifies the steps necessary to materialize the RFC. The subsequent phases of a traditional change process are *planning*, *evaluation*, *test*, *authorization*, *scheduling*, *implementation*, and *record* of the executed actions in the *Configuration Management Database* (CMDB).

To meet the previously mentioned change management goals, it is of paramount importance reusing the experience acquired from the execution of previous changes in the design of subsequent ones. Ideally, this reuse can be done considering two distinct approaches. In a top-down approach, IT operators may design change models based on the knowledge owned/acquired in past changes. Considering a reverse, bottom-up perspective, change models could be discovered from logs gathered from IT provisioning tools.

There have been several investigations approaching the top-down perspective of change design [6][7]. Proposed solutions, however, often require recurrent changes to be designed "from scratch", hence hampering the reuse of knowledge acquired from previous ones. We started to address this issue in a previous work [4], by introducing the concept of *change templates* as a mechanism to promote the reuse of knowledge accumulated with past IT changes. Nonetheless, the use of change traces to discover change models has not been addressed in the field of change management.

We highlight three reasons for extracting change templates from traces. First, templates discovered from these traces are less prone to be incomplete, and more appropriate for a broader range of change scenarios. Conversely, manually designed templates may contain errors and/or be incomplete, considering that knowledge about changes may be scattered among IT operators and paper procedures. Furthermore, the lack of visibility of the entire IT infrastructure makes difficult to design templates generic enough to enable their applicability in several similar contexts. Second, several changes may have been executed in an *ad hoc* fashion [16] (*i.e.*, without any documented change plan). This is often the case, for example, during emergency changes, or in organizations that do not adopt a proper methodology to design changes. In these cases, template mining provides an opportunity to reuse the knowledge behind such *ad hoc* processes, demanding fewer efforts than would be required to manually design the change template. Third, discovering change templates from the automated analysis of execution traces has potential to reduce significantly the time consumed in the specification of new templates. In contrast, the manual design of change templates, often conducted "from scratch", tends to be a time-consuming (and also error prone) task.

The use of historic data to extract process models has been largely investigated by several communities (please refer to Rozinat *et al.* [5] for a research overview). Nevertheless, despite the potential benefits to the IT change management process, this topic has not been addressed in previous investigations [4][7][16].

To bridge this gap, in this paper we propose a mechanism to extract change templates from historic traces recorded in provisioning systems. In contrast to the already established workflow mining techniques, our solution focuses on generating templates that capture the essence of previously executed change processes. The discovered templates, in turn, enable the reuse and applicability of such processes in scenarios having settings diverse from the original process (*e.g.*, different dependency relationships among affected IT components, or specific system requirements).

The mechanism proposed in this paper is evaluated through the use of CHANGEMINER, a prototypical implementation of a change template miner subsystem, and a rich set of experiments. During the evaluation, we have observed the completeness and correctness of the change templates extracted from several change execution data, in addition to performance indicators.

The reminder of this paper is organized as follows. Section II discusses related work. Section III briefly reviews the building blocks that support our solution. Section IV details the process to discover change templates from execution traces. Section V emphasizes implementation aspects and presents the results achieved with an experimental evaluation. Finally, Section VI concludes the paper with remarks and outlines directions for future work.

II. RELATED WORK

In this section we review some investigations in the field of IT management and briefly describe some process mining techniques that support our work.

A. Management of IT Systems

Several aspects of IT management have been addressed in recently published work. Important steps have been taken, for example, towards formalization of change-related documents [6], automated planning of IT changes [4][8], and business-driven scheduling of IT changes [16].

There have also been investigations on automated planning and scheduling of change operations (*e.g.*, CHAMPS [7] and CHANGELEDGE [4]), and on deployment of change operations over managed IT infrastructures (*e.g.*, Opsware [10]). These solutions, however, require IT operators to explicit design change plans, and do not offer support to discover change processes from the analysis of execution traces.

In a previous work [4], we have proposed the concept of change templates as a mechanism to *formalize*, *preserve*, and *reuse* knowledge acquired from previous changes. Template *association*, *composition*, *generalization*, and *specialization* are explored as techniques that facilitate the reuse of knowledge during the design of changes. In a subsequent work [8], we have proposed a conceptual solution for the automated refinement of preliminary change plans into actionable workflows. Our solution accounts for the impact that computed actions will cause over the managed IT infrastructure, when computing the subsequent ones. Finally, in another previous work [11], we have introduced the concept of atomic groups in the design of change plans with the purpose of providing an end-to-end solution to IT change management with rollback support.

Despite the progresses achieved in the field, as previously discussed, the use of a mechanism to extract templates from previously executed changes has not been addressed in previous investigations. The need for such a mechanism is underscored by the fact that operator errors account to the largest fraction of service-delivery disruption caused by failed changes [12]. In the following subsection, we highlight some of the most prominent techniques that enable the discovery of process models from execution traces.

B. Process Mining

The idea of extracting workflow processes from execution logs has been explored in the field of software engineering and workflow management [5]. Seminal techniques proposed, however, were not able to mine workflows having *and*/*or branches*/*joins* [13]. As a consequence, the resulting workflows were limited to a purely sequential behavior.

To tackle this limitation, Maruster *et al.* [13] have proposed a mechanism to mine workflows from event logs, which explicitly distinguishes decision points that route the workflow enactment. In a subsequent work [14], the authors have extended the class of workflows their technique can mine, by enabling their algorithm to mine short loops from event logs. More recently [15], the authors have proposed a supplementary mechanism that enables understanding how environment data influences in the routing of a workflow execution. The proposed mechanism, called *Decision Miner*, was implemented and evaluated in the context of a process mining framework called ProM.

The solution for change template mining, proposed in this paper, adopts some principles and techniques introduced in the field of process mining. In the following sections, we describe the proposed solution in detail.

III. CONCEPTUAL SOLUTION FOR CHANGE TEMPLATE MINING

In order to support template mining, we have introduced a new component in the conceptual solution for planning and implementation of changes, proposed in a previous work [4]. This fundamental component – not approached in the original conceptual solution due to its complexity – materializes our mechanism for template mining and couples adequately, without any significant modification, in the already proposed conceptual solution.

In our solution, the discovery of change templates from execution traces represents the primary step for the conduction of a change management process. Fig. 1 presents an overview of the extended solution, highlighting the newly proposed component and interactions.

The *IT Operator* starts a change mining process by interacting with component *Change Miner* (flow 1 in Fig. 1). In this initial step, the operator indicates the traces that will be used for the analysis and subsequent template extraction. The execution traces consumed in this process are available from the *Configuration Management Database* (flow 2). These traces are typically recorded by the component *Deployment System* (a generic abstraction for IT provisioning tools) (3), during the execution of previous changes over the *Configuration Items* (CIs) present in the IT infrastructure. The *Change Miner* generates change templates that capture high-level steps involved in the changes associated to the analyzed traces. Thereafter, the resulting templates can be modified by the operator as to precisely reflect his/her needs (1), and finally stored in the *Change Templates* repository (4).

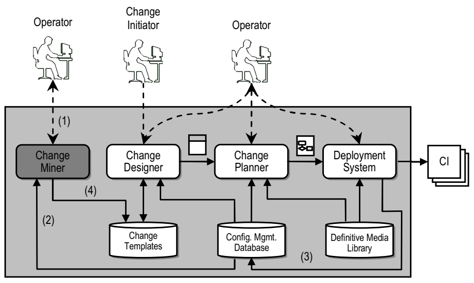


Fig. 1. Extended conceptual solution.

The extracted change templates may be latter (re)used, as envisaged in our original conceptual solution, in the design of future RFCs, solicited by change initiators, and preliminary change plans, by operators (interacting with the component *Change Designer*). Afterwards, the newly designed preliminary plans may be refined into *actionable workflows* (using the component *Change Planner*), and subsequently deployed over the managed CIs (with the aid of a *Deployment System*). Please refer to our previous work [4] for additional information about the use of change templates in the conduction of IT change management processes.

Having presented a general view of the solution, in the following subsections we describe, in more detail, the intermediate, grained-level phases of the change template discovery process, the models for the persistence of execution traces and specification of change documents, and the concept of change management templates.

A. Template Discovery Process Overview

The solution for template mining, encapsulated in the component *Change Miner*, comprises three distinct, well-defined steps. These steps are materialized by subcomponents *Workflow Miner*, *Decision Miner*, and *Template Converter*, illustrated in Fig. 2. The relationship between these subcomponents and with external entities is described next.

Once the change mining process is started by the operator, the component *Change Miner* triggers the execution of the subcomponent *Workflow Miner*. The purpose of this subcomponent is to extract workflows that reflect the behavior of similar changes executed in the past. To this end, it consumes, from IT-related repositories (flow 1 in Fig. 2), execution traces generated by IT provisioning tools. By analyzing these traces, the *Workflow Miner* is able to discover the low-level activities (and their relationship) that composed the past changes. This subcomponent produces typical workflows of activities having *and*/*or branches*/*joins*.

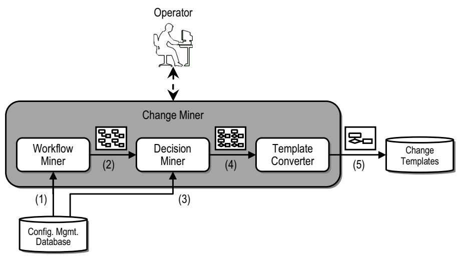


Fig. 2. Change miner subcomponents.

The workflows generated in the previous step are then processed by the subcomponent *Decision Miner* (flow 2). Also exploring historical information present in the *Configuration Management Database* (3), this subcomponent identifies how data has influenced the execution routing. Subsequently, it populates the *and*/*or branches*/*joins* present in the workflows with the conditions that needed to be satisfied for the transitions to take place. Finally, the resulting workflows are consumed by the subcomponent *Template Converter* (4). This subcomponent will abstract the change procedures described in the workflows as to enable their applicability in several, diverse change contexts. Thereafter, as previously mentioned, the resulting templates may be modified by the operator and stored in the *Change Templates* repository (5) for future (re)use.

B. Change Record and RFC & Change Plan models

The solution for extraction of change templates from change traces requires the use of models to (*i*) represent historic data gathered from different provisioning tools, and (*ii*) support the specification of change templates resulted from the mining process. In this subsection we cover, in more detail, the models that support our solution.

We have adopted a subset of the *Common Information Model* (CIM) [17] to represent traces generated by provisioning tools from different software vendors. This model allows representing log information generated by general-purpose systems and, consequently, enabling the applicability of our solution in several contexts. Fig. 3 presents a partial view of the model.

The root class of the model is *ManagedElement*. Through specialization, it allows the representation of the several Configuration Items (CIs), as well as IT personnel, which compose the IT infrastructure. One of the specialized classes, *LogicalElement*, enables the representation of IT provisioning systems that are employed for the implementation of changes over the managed CIs. Instances of class *MessageLog* represent, in the context of our work, the several changes executed over these CIs. Class *LogRecord*, in turn, holds information about tasks executed during the change process (in other words, it enables the representation of execution traces). These traces may be associated to one or more *ManagedElement*, in this case representing that the associated element was involved in the generation of that trace.

In regard to the *Requests for Change & Change Plan* model, it enables the design of change-related documents and relies on both (*i*) guidelines presented in the ITIL Service Transition book [2], and (*ii*) the workflow process definition, proposed by the *Workflow Management Coalition* (WfMC) [9]. A partial view of the model is presented in Fig. 4.

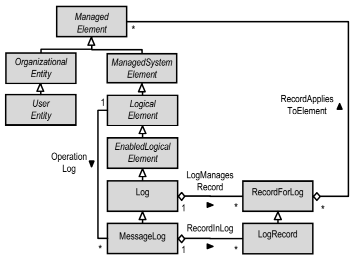


Fig. 3. Model for the representation of execution traces.

Classes such as *RFC* and *Operation* (highlighted in white) allow expressing changes, in a high level of abstraction, designed by the change initiator. Going down to the level of change plans, classes *ChangePlan*, *Activity*, and *Transition Information* (highlighted in grey) support the modeling of workflows that materialize changes specified in RFC documents.

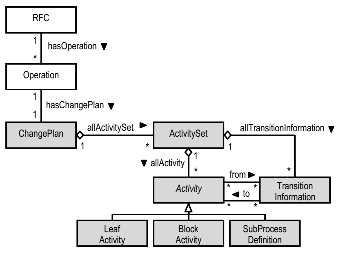


Fig. 4. RFC & Change Plan model.

C. Change Management Templates

In the scope of this research, we define change templates as parameterized building blocks that can be reused in similar, recurrent change requests. A **request template**, defined by means of the classes in white from the model of Fig. 4, defines a set of prototypical operations to be performed on the IT infrastructure. These operations have a type and a number of parameters (*Integer*, *String*, *etc.*, or a reference to a CI affected by the operation) to be consumed during the change.

Plan templates, defined by means of the classes in gray, comprise steps necessary to materialize recurrent changes described in RFCs/request templates. The large-grained activities that compose a plan template also have a number of parameters, which must be filled in during its instantiation into a change plan. Please refer to our previous work [4] for an in-depth explanation on the concept of change templates.

IV. Change Template Mining

The conceptual solution and building blocks presented in the previous section form the fundamental basis for our change template mining solution. Starting from past execution traces, our solution extracts the workflows behind the implemented changes and, afterwards, transforms them into change templates. In our first iteration at this problem, we focus on the extraction of templates that comprise simple execution routing structures, *i.e.*, composed of *and*/*or branches*/*joins*. In the following subsections, we depict our solution in more detail. Subsection A presents how actionable change plans are extracted from execution traces, and Subsection B details how the extracted plans are transformed into change templates.

A. Discovering Change Processes from Logs

As presented in Subsection III.B, subcomponent *Workflow Miner* is responsible for extracting the *actionable change plans* (or *actionable workflows*) executed for the implementation of past changes. To materialize the conceptual functionality of this subcomponent, we have tailored some concepts from already established process mining techniques [13][14] to the context of change management.

There are simplification assumptions adopted by process mining techniques which are also applicable in our work. First, they assume that each event present in an execution trace refers to an *activity* (*i.e.*, a well defined step in the workflow). Second, each event refers to a *case* (*i.e.*, a workflow instance). Finally, they assume that events present in the trace are totally ordered. Researchers in the field of process mining argue that, in fact, any information system using transactional systems will offer this information in some form [13]. In the context of our work, *activity* maps to *change action* (the smallest unit of a change present in an RFC document), *cases* represent instances of execution of RFCs onto the target IT environment, and IT provisioning tools are examples of *transactional systems*.

TABLE I
EXAMPLE OF CHANGE LOG

Change trace number	Executed change actions
Trace 1	a_1, a_3, a_5, a_6, a_4, a_9
Trace 2	a_1, a_3, a_4, a_5, a_6, a_9
Trace 3	a_2, a_5, a_6, a_3, a_4, a_9
Trace 4	a_1, a_7, a_3, a_8, a_4, a_9
Trace 5	a_2, a_3, a_7, a_4, a_8, a_9
Trace 6	a_2, a_3, a_5, a_4, a_6, a_9

In order to enable the applicability of process mining concepts in the field of change management, we denote a_i $\{0 \leq i \leq n\}$ as being a *change action* (smallest unit of a change) performed over the managed IT infrastructure. Sequences of change actions are represented by *change traces*. Finally, a *change log L* is a set of all traces related to a same process. We assume that *L* may be created by grouping together, from the log repository, traces that share a same attribute value (*e.g.*, the same *change ID*). To facilitate the understanding of the process, we illustrate each step considering the example of change log presented in Table I. In this example, there are six change traces, each composed by nine executed tasks.

The process for the extraction of change plans from execution logs is comprised of four high-level steps, which are described below.

1) *Creation of the Dependency/Frequency Table*: The Dependency/Frequency (D/F) table [13] contains information about the frequency of occurrence of events registered in the

change log (*e.g.*, number of executions of each *change action*, and number of succession relationships that occur between them). The information contained in the D/F table are (*i*) the identifier of task a_i and a_j, (*ii*) the overall frequency of task a_i, (*iii*) the overall frequency of task a_j, (*iv*) the frequency in which task a_i directly succeeds another task a_j ($a_j > a_i$), (*v*) the frequency in which task a_j directly succeeds another task a_i ($a_i > a_j$), (*vi*) the frequency in which a_i directly or indirectly succeeds another task a_j, but before the next occurrence of a_j ($a_j >>> a_i$), and (*vii*) the frequency in which a_j directly or indirectly succeeds another task a_i, but before the next occurrence of a_j ($a_i >>> a_j$). Table II shows an excerpt of D/F table for the change log illustrated in Table I. We highlight the number of times in which $a_3 > a_4$ holds, suggesting that a_3 and a_4 are likely to be connected in the workflow.

TABLE II
DEPENDENCY/FREQUENCY TABLE

$a_i a_j$	# a_i	# a_j	($a_j > a_i$)	($a_i > a_j$)	($a_j >>> a_i$)	($a_i >>> a_j$)
a_3a_1	500	404	236	0	404	0
a_3a_2	500	96	56	0	96	0
a_3a_3	500	500	0	0	0	0
a_3a_4	500	500	0	160	0	500
a_3a_5	500	339	114	147	0	339
a_3a_6	500	339	27	86	0	339

2) *Calculation of the Local Metric, Global Metric, and Causality Metric*: The Local Metric (LM) expresses the tendency of succession relation by comparing the magnitude of ($a_i > a_j$) versus ($a_j > a_i$). The Global Metric (GM), in contrast, determines the likelihood of succession between two change actions a_i and a_j considering the overall frequencies of these actions. Finally, the Causality Metric (CM) indicates the chance that action a_i causes the occurrence of action a_j. Table III presents the values for metrics LM, GM, and CM computed for the pairs of activities presented in the first field of Table II. For example, the high values of LM, GM, and CM for the pair a_3, a_4 reinforce the observation done for the data in Table II.

TABLE III
VALUES FOR METRICS LM, GM, AND CM

$a_i a_j$	LM	GM	CM
a_3a_1	0	0	0
a_3a_2	0	0	0
a_3a_3	0	0	0
a_3a_4	0.98	0.32	1.00
a_3a_5	0.50	0.09	0.32
a_3a_6	0.67	0.17	0.13

3) *Computation of the Existence of Direct Succession Between Tasks*: In this step, the three metrics LM, GM, and CM, computed in the previous step, are combined using a *Logistic Regression Model* [14]. The output of this combination is the probability π that two events a_i and a_j, present in the log, are in a direct succession relationship.

4) *Generation of the Actionable Change Plan*: This last step consists in identifying *and*/*or branches*/*joins* between tasks, considering the succession relationship information present in the D/F table. After identified, they are explicitly incorporated into the workflow. Finally, the activities in the discovered plan are associated to the *Configuration Items* they affect. Fig. 5 illustrates the change plan extracted from the change log partially illustrated in Table I.

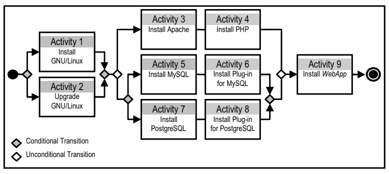


Fig. 5. Change plan for the change log from Table I.

The reader may note, in the workflow presented in Fig. 5, that activities a_3 and a_4 are in a direct succession relationship, confirming the evidences presented in Tables II and III.

Having presented an overview of the process to extract change plans from execution logs, the following subsection focuses on how these extracted plans are converted into change templates.

B. Extracting Templates from Process Models

Our solution for the conversion of an actionable change plan into a change template is materialized by a conceptual algorithm, illustrated in Fig. 6. The conceptual algorithm, denoted by *f*, takes as input the change plan *C* to be converted, the current state of the IT infrastructure *I*, and the *Definitive Media Library* (DML) *R*. The output is a change template *T* that comprises the high-level steps necessary to implement the same change described in *C* in a broader range of scenarios.

```
1   f(C, I, R) = T {
2      declare A = set of activities from the change plan C
3      declare S: empty set of activities
4      declare T: empty change template
5
6      for each activity a_i ∈ A do
7         S = S ∪ {set of dependencies of a_i, considering I and R}
8      T = A - S
9      declare B = set of activities a from the change template T
10     for each activity a_i ∈ B do {
11        for each activity b_i ∈ B do {
12           if (a_i ≠ b_i  and there is a path from a_i to b_i in change plan C) {
13              declare cond : null logical expression
14              P = {set of paths P_i that connect a_i to b_i in change plan C}
15              for each transition t_i arriving at b_i do {
16                 if (∃ P_i ∈ P | t_i ∈ P_i)
17                    cond = cond + (or (condition of transition t_i))
18              }
19              connect a_i to b_i in T using as condition the logical expression cond
20           }
21        }
22     }
23     declare D: empty set of variables v of managed elements
24     for each activity a_i ∈ B do {
25        transform all managed elements e affected by a_i into variables v
26        D = D ∪ {list of variables v present in activity a_i }
27     }
28     add D to the list of variables of the change template T
29     return T
30  }
```

Fig. 6. Conceptual algorithm for converting change plans into templates.

Fig. 7 illustrates the functioning of *f* using a simplified example. The change plan *C* presented in the figure – extracted using the change mining technique presented in the previous subsection – comprises the steps necessary to install a *Web Application* (*WebApp*).

As a first step towards the conversion of an actionable workflow into a change template, the set of activities that belong to the change plan *C* is copied to *A* (line 2 in Fig. 6). Next, an empty set of activities *S* (line 3), and an empty template *T* (line 4) are created. The newly created template *T*, in this case, will hold the result of the conversion process.

Subsequently, for each activity a_i that belongs to A (line 6), f computes the set of dependencies that must be satisfied prior to its execution, considering information about the target IT infrastructure I and dependencies between software packages available in R. The result of this computation is added to S (line 7). At this point, S is composed only of activities that satisfy any dependency of any other activities of the change plan C. In the example illustrated in Fig. 7, S will be composed of activities *Install WebSrv*, *Install DBServer*, and *LibX*. In this case, *WebSrv* and *DBServer* are software that must be installed prior to the installation of *WebApp* (according to *WebApp*'s *SwChecks*, present in the DML). *LibX*, in turn, is a software dependency for both *WebServer* and *DBServer* (as both require its prior installation, according to their *SwChecks*, also shown in the DML). Since activity *Configure DNS* is not an explicitly dependency found in the DML, it is not added to *S*.

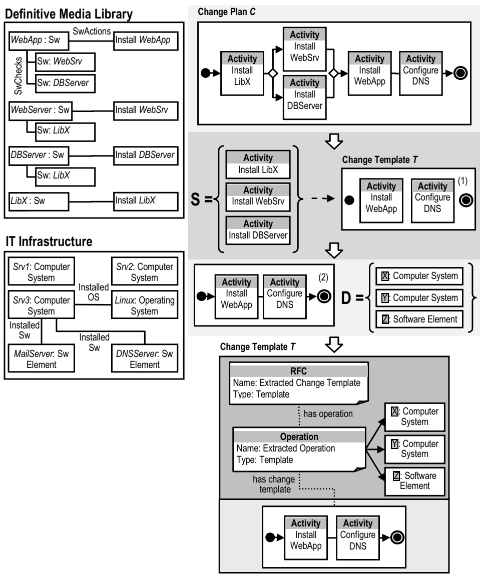


Fig. 7. Illustration of the template conversion process.

After processing the activities in A, the computed dependencies (in S) are excluded from A, and the remaining ones are added to T (line 8). At this point, T is composed only of activities that cannot be deduced through automated analysis of dependencies between elements from IT-related repositories. In the example of Fig. 7, the template T (box 1) is composed of activities *Install WebApp* and *Configure DNS*.

The activities that have been just added to T do not have any relationships to one another (*i.e.*, there are no conditional/unconditional transitions between them). This information must be obtained by observing how activities are linked to each other in the original change plan C. In order to compute this information, the set of activities contained in template T is copied to B (line 9). Subsequently, for each pair of ordered activities $(a_i, b_i) \in B$ (lines 10-11), if there is at least a path of activities starting in a_i and ending in b_i in the original change plan C (line 12), the following steps are performed. First, *cond* is initialized with a *null* logical expression (line 13). Subsequently, all different paths of activities P_{ii} from a_i to b_i are stored in P (line 14) (P_{ii}, in this case, is an ordered set of transitions that form a path of activities from a_i to b_i). Then, for each transition t_k that arrives at activity b_i (line 15), if this transition belongs to any of the paths in P (line 16), the condition of this transition is used to build a logical expression, which will be concatenated to *cond* (line 17). After all transitions that arrive to b_i are processed, a_i is connected to b_i in the change template T (line 19), using as transition condition the logical expression stored in *cond*. In Fig. 7, the template T shown in box 1 was modified by the addition of the transitions between the activities that compose it, yielding the one presented in box 2.

Once the transitions between activities that compose the change template T are created, f processes all managed elements affected by activities in T, converting references to them into *variables*. These variables are an important aspect of change templates, as they permit to abstract the managed elements that will be affected by the change (please refer to Cordeiro *et al.* [4] for further information on template variables). As a first step towards the processing of these variables, D is declared as a set of variables v of managed elements, *i.e.*, pointers to elements belonging to the IT infrastructure (line 23). Next, for each activity a_i of template T present in B (line 24), the managed elements affected by these activities are abstracted, *i.e.*, the activity will no longer be associated to a concrete managed element. Instead, it will reference variables v having the same class of the affected managed elements e (line 25). The variables resulted from this abstraction will be added to D (line 26).

In the example illustrated in Fig. 7, activity *Install WebApp* affects two managed elements: the software to be installed (*WebApp*), and the computer in which it will be installed (*Server*$_2$). After the abstraction step (line 25), the software installed by *Install WebApp* becomes Z (and has type *SoftwareElement*), and the computer in which the software is installed becomes X (of type *ComputerSystem*). *Configure DNS*, in turn, modifies software settings of the network's domain name server, located in computer *Server*$_3$. After the abstraction, the element affected by this activity becomes Y (*ComputerSystem*). All these three variables are stored in the set of variables D, as illustrated in Fig. 7.

As a last step, the variables present in D are added to the list of variables of the change template T (line 28) and returned to the change operator (line 29). In Fig. 7, the change template resulted from this process has three variables, X, Y, and Z, and is composed of two activities, ordered according to the relationship information acquired from the previous change plan C. At this point, the change template T is ready for use in the instantiation of similar future changes.

V. Experimental Evaluation

Our template mining solution is supported by ChangeMiner, a prototypical implementation of a subsystem within the ChangeLedge system [4]. To prove the conceptual and technical feasibility of our proposal, we have conducted an experimental evaluation considering the extraction of change templates from synthetically generated traces. This decision was made because it enables a more thorough analysis of the solution, by enabling the comparison of the extracted templates with the real processes that have implemented the changes. Due to space constraints, we focus our analysis on three of these templates. As a result of the template mining process, we have observed the correctness and completeness of the obtained templates, in addition to performance indicators.

The IT infrastructure employed in the case study was composed of four servers: *Server$_1$*, having no operating system installed on top of it; *Server$_2$*, having *Windows 2003* installed; *Server$_3$*, and *Server$_4$*, both having *Debian GNU/Linux* installed.

In regard to the change logs consumed during the mining process, they were generated by simulation of the execution of four different RFCs on the previously described IT environment. For each RFC, we have generated one change log with 500 change traces. Each simulation has considered uniform distributions for the time spent in the execution of each activity, as well as distinct probabilities, defined empirically, for choices present in the change plans.

The first two RFCs used in the simulation had as objective the installation of an *e-Commerce Web application* (*WebApp*), one of them having *Server$_1$* as target CI and the other, *Server$_3$*. The third RFC comprised the migration of the entire system installed on *Server$_3$* to *Server$_4$*. Information about the number of activities and types of transitions that composed the change plan associated to each of the three RFCs is presented in Table IV.

TABLE IV
CHARACTERISTICS OF THE CHANGE PLANS

RFC	Activities	Conditional Transitions	Unconditional Transitions
1	26	16	29
2	84	26	115
3	245	100	383

A partial view of the log generated from the execution of the first change is presented in Table V, whereas the characteristics of the templates extracted from the logs of the three RFCs are summarized in Table VI. An interesting point to highlight is that templates 1 and 2, generated from RFCs 1 and 2, respectively, encode the prototypical steps (illustrated in Fig. 8) to implement a same general change. This is explained by the same goal that both RFCs share, even though they are concerned with distinct environments, *i.e.*, having different sets of dependencies to be fulfilled.

TABLE V
PARTIAL VIEW OF THE CHANGE LOG FROM THE FIRST RFC

Change Trace	Executed change actions
Trace 1	$a_1, a_4, a_5, a_3, a_7, a_6, a_8, a_9$
Trace 2	$a_1, a_2, a_5, a_7, a_4, a_3, a_6, a_8, a_9$
Trace 3	$a_1, a_2, a_4, a_3, a_5, a_6, a_7, a_8, a_9$
Trace 4	$a_1, a_5, a_4, a_3, a_6, a_7, a_8, a_9$
Trace 5	$a_1, a_4, a_3, a_6, a_5, a_7, a_8, a_9$

TABLE VI
CHARACTERISTICS OF THE EXTRACTED TEMPLATES

Template	Activities	Conditional Transitions	Unconditional Transitions
1	9	2	15
2	9	2	15
3	57	0	65

Figs. 8 and 9 show a partial view of the templates extracted from the execution traces of the first and third changes, respectively. The reader may note, in the change template from Fig. 8, that *Activity 2*, *Install Security Updates for WebApp*, is conditionally executed (for example, whenever the system in which *WebApp* is installed requires these updates). An indication of existence of this decision may be observed, for example, in traces 3 and 4. In the first trace, *Activity 2* is executed after *Activity 1*, *Install WebApp*. In the second, though, *Activity 2* is not executed. The reader may also note that the order of execution of *Activities 3*, *4*, *5*, *6*, and *7* varies in the change traces, indicating the existence of parallelism upon their execution. As shown in Fig. 8, this behavior is also captured and materialized in the extracted change template.

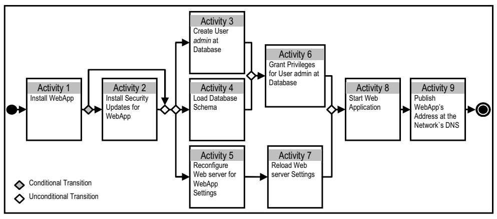

Activity 1
Install WebApp
Activity 2
Install Security Updates for WebApp
Activity 3
Create User admin at Database
Activity 4
Load Database Schema
Activity 5
Reconfigure Web server for WebApp Settings
Activity 6
Grant Privileges for User admin at Database
Activity 7
Reload Web server Settings
Activity 8
Start Web Application
Activity 9
Publish WebApp's Address at the Network`s DNS
Conditional Transition
Unconditional Transition

Fig. 8. Partial view of the change template extracted from the change log of the first RFC.

As for the change template illustrated in Fig. 9, we highlight the size and complexity of the depicted change template, extracted from a change log that comprised over 245 distinct, low level change actions. It is important to mention that the completeness and generality of these templates is highly influenced by the number of change traces available during the template mining process. For example, if a smaller set of change traces were used to compute the change template from Fig. 9, the representativeness of the 245 activities would be affected, thus making the template mining mechanism to interpret some (or most of them) as *noise*. As a consequence, the resulting template would be less complete and general (*e.g.*, composed of fewer activities or transitions between them).

The performance of CHANGEMINER to extract the previously mentioned templates is presented in Table VII. Our experiments were conducted on a computer equipped with a Pentium^{tm} Centrino processor, 1.7 GHz of CPU clock, 2,048 KB of cache, and 512 MB of RAM memory.

TABLE VII
TIME CONSUMED DURING THE TEMPLATE MINING PROCESS

Change Scenario	Mining Time (s)	Confidence Interval of the Mining Time	
		Lower Bound (s)	Upper Bound (s)
1	31	29	33
2	36	34	39
3	112	104	121

The CHANGEMINER subsystem has performed satisfactorily, demanding a few dozens of seconds (from 31 to 112) to generate the templates. We have also calculated a confidence interval of 95% for the measured times, considering 10 repetitions of the template mining process for each change. As shown in Table VII, we expect the template mining processing time to vary minimally, for each scenario. The results show that our solution not only generates complete and correct change templates, but has potential to reduce, in a significant way, time and efforts demanded to this end.

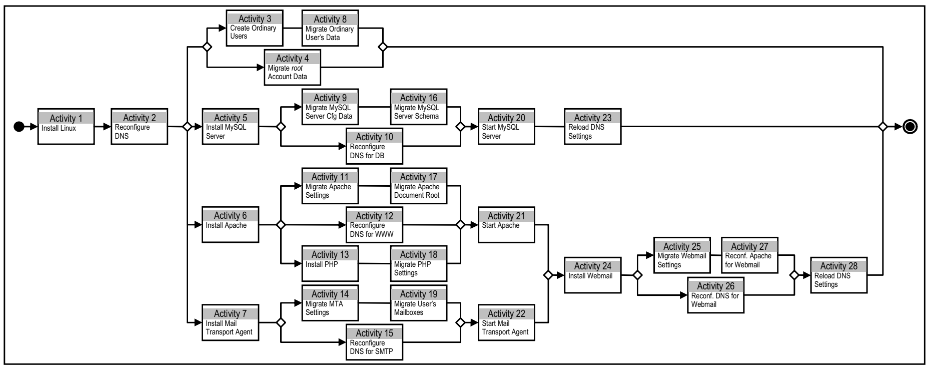


Fig. 9. Partial view of the change template generated for the third RFC.

VI. Conclusions and Future Work

Historic data generated by IT provisioning tools has a huge potential to reveal knowledge behind the conduction of past changes over managed IT infrastructures. However, the lack of a mechanism to extract IT change process from these logs, along with proper support to enable the applicability of these processes to a broader range of change scenarios, has hampered the reuse of such knowledge in practice. To address these issues, we have proposed ChangeMiner, a solution that enables the capture and reuse of knowledge – through the use of change management templates – from historic data generated during the implementation of past changes.

The experiments performed to evaluate our solution, although not exhaustive, have showed the feasibility of using execution logs to produce new change templates. The templates extracted have captured the nuances of the changes implemented by the original plans. Furthermore, the extraction of templates ran on the order of dozens of seconds. This time is certainly of lower magnitude than the time that a human operator would spend designing the same templates.

It is important to mention that the change templates extracted with our solution are not influenced by traces that describe changes that have failed. Since these traces differ significantly from the traces that describe successful changes, the events registered by them (*e.g.*, direct and indirect succession between activities) are regarded as noise. As a consequence, they become statistically irrelevant to the change plan mining process.

One simplification assumption that we have adopted in our solution was considering *and*/*or branches*/*joins* as not being influenced by environmental data. In a future work, we intend to incorporate decision mining techniques [15] to our solution, in order to generate change templates composed of decision structures that take into account these data. Moreover, we plan to (*i*) evaluate our solution taking into consideration real-life data (*e.g.*, mining change logs generated by Opsware [10]), and (*ii*) further investigate the sensitivity of the template mining process to the number and nature of the traces available, and the complexity of changes available in execution logs.

References

[1] Information Technology Infrastructure Library. Office of Government Commerce (OGC), 2008. http://www.itil-officialsite.com.

[2] IT Infrastructure Library: ITIL Service Transition, v. 3. London: The Stationery Office, 2007, 270 p.

[3] Sauvé, J. *et al.* On the Risk Exposure and Priority Determination of Changes in IT Service Management. In: Distributed Systems: Operations and Management, San José, USA, pp. 147-158, 2007.

[4] Cordeiro, W. *et al.* A Template-based Solution to Support Knowledge Reuse in IT Change Design. In: *IFIP/IEEE Network Operations and Management Symposium*, Salvador, Brazil, pp. 355-362, 2008.

[5] Rozinat, A. *et al*. Discovering Colored Petri Nets from Event Logs. In: *International Journal on Software Tools for Technology Transfer*, Vol. 10, No. 1, pp. 57-74, 2008.

[6] Keller, A. Automating the Change Management Process with Electronic Contracts. In: *7th IEEE International Conference on E-Commerce Technology Workshops*, pp. 99-107, 2005.

[7] Keller, A. *et al.* The CHAMPS system: change management with planning and scheduling, In: *IEEE/IFIP Network Operations and Management Symposium*. vol.1, pp. 395-408, 2004.

[8] Cordeiro, W. *et al.* A Runtime Constraint-Aware Solution for Automated Refinement of IT Change Plans. In: *Distributed Systems: Operations and Management*, Samos Island, Greece, pp. 69-82, 2008.

[9] The Workflow Management Coalition Specification. Workflow Process Definition Interface - XML Process Definition Language. http://www.wfmc.org/standards/ docs/TC-1025_10_xpdl_102502.pdf.

[10] Opsware Inc. Opsware Data Center Automation Platform. http://www.opsware.com/.

[11] Machado, G. *et al.* Enabling Rollback Support in IT Change Management Systems. In: *IFIP/IEEE Network Operations and Management Symposium*, Salvador, Brazil, pp. 347-354, 2008.

[12] Oppenheimer, D.; Ganapathi, A.; Patterson, D. A. Why do internet services fail, and what can be done about it? In: *4th Usenix Symposium on Internet Technologies and Systems*, Seattle, USA, 2003.

[13] Maruster, L. *et al.* Process Mining: Discovering Direct Successors in Process Logs. In: *5th International Conference on Discovery Science*, vol. 2534 of Lecture Notes in Artificial Intelligence, pp. 364-373. Springer-Verlag, Berlin, 2002.

[14] Medeiros, A. *et al.* Process Mining: Extending the alpha-algorithm to Mine Short Loops. In: *BETA Working Paper Series*, WP 113, Eindhoven University of Technology, Eindhoven, 2004.

[15] Rozinat, A.; van der Aalst., W. Decision Mining in ProM. In: *Business Process Management 2006*, vol. 4102 of Lecture Notes in Computer Science, pp. 420-425. Springer-Verlag, Berlin, 2006.

[16] Rebouças, R. *et al.* A Decision Support Tool to Optimize Scheduling of IT Changes. In: *IFIP/IEEE International Symposium on Integrated Network Management*, Munich, Germany, pp. 343-352, 2007.

[17] Distributed Management Task Force: Common Information Model. http://www.dmtf.org/standards/cim.

Informática UFRGS

PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

"ChangeLedge: Change Design and Planning in Networked Systems based on Reuse of Knowledge and Automation"

por

Weverton Luis da Costa Cordeiro

Dissertação apresentada aos Senhores:

[illegible]

Prof. Dr. Antônio Jorge Gomes Abelém
(UFPA)

[illegible]

Prof. Dr. Antônio Marinho Pilla Barcellos

[illegible]

Prof. Dr. Marcus Ritt

Vista e permitida a impressão.
Porto Alegre, ___/__/____

Luciano Gaspary

Prof. Dr. Luciano Paschoal Gaspary
Orientador

[illegible]

Prof. Álvaro Freitas Moreira
Coordenador do Programa de
Pós-Graduação em Computação - PPGC
Instituto de Informática - UFRGS