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

F. De Turck, W. Kellerer, and G. Kormentzas (Eds.): DSOM 2008, LNCS 5273, pp. 69–82, 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).

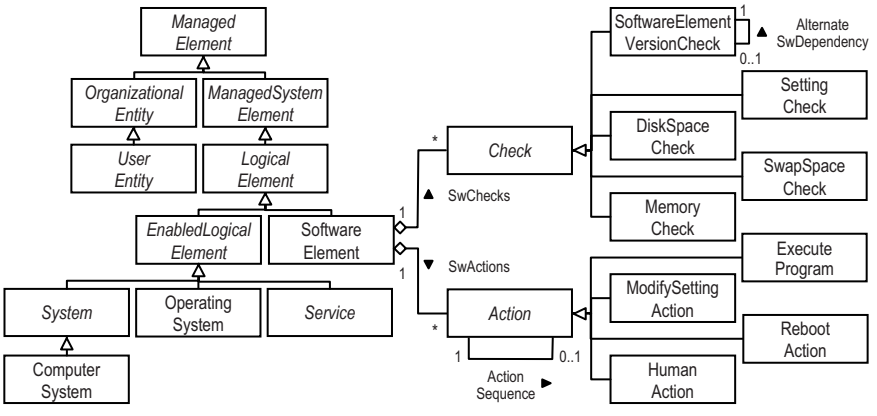

Fig. 1. Partial view of the IT Infrastructure model

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     s0 = initial snapshot of I, after the execution of 0 activities
5     f'(C', R, I, s0, A')
6     if (C' is refined)
7       return C'
8     else
9       return false
10  }
11
12  f'(C, R, I, si, A) {
13    if (A is empty)
14      return change plan C
15    else {
16      declare X: set of arrangements Y of activities
17      ai = i-est activity ∈ A
18      declare A' = A - {ai}
19      if (ai has no computable dependencies, given I, si, and R)
20        f'(C, R, I, si, A')
21      else {
22        X = set of arrangements Y of first level dependencies of ai, given I, si, and R
23        for each Yi ∈ X {
24          declare C' = C + Yi
25          declare A'' = A' ∪ Yi
26          si+1 = new snapshot of the IT infrastructure I, given C', I, and si
27          f'(C', R, I, si+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 $A' = \{Install\ WebApp\}$.

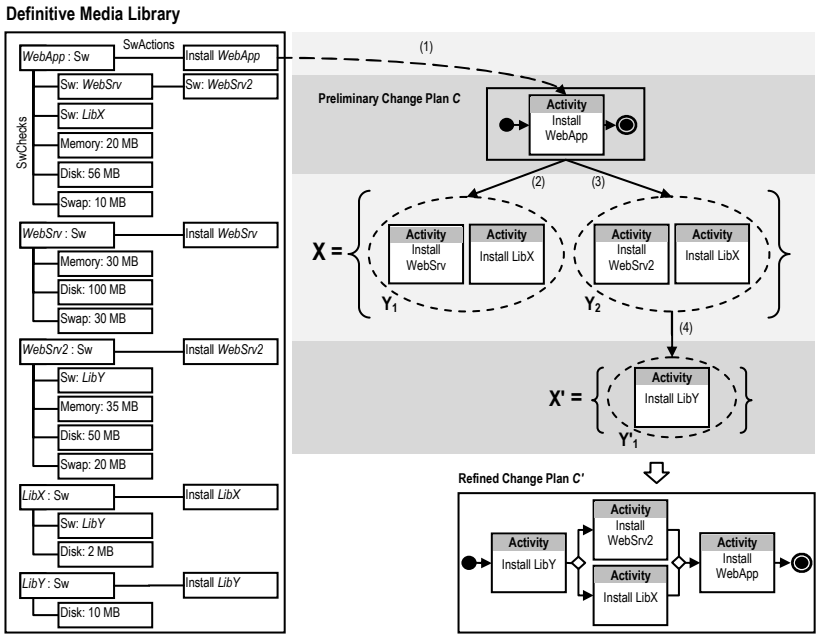

Fig. 3. Illustration of the functioning of f'

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.

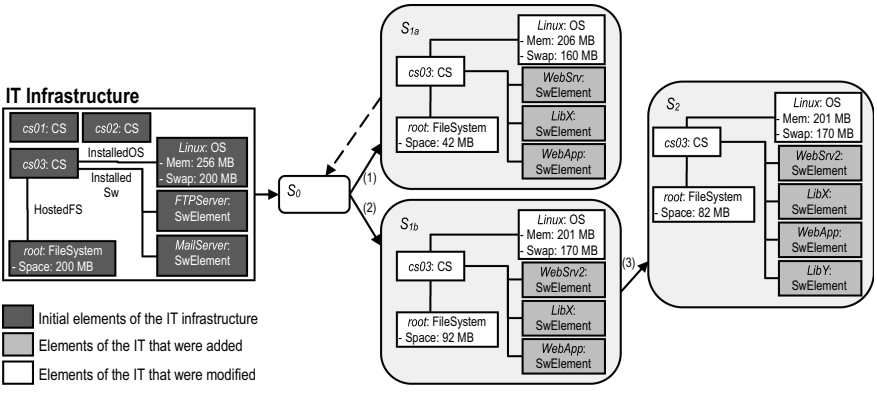

Fig. 4. Evolution of the snapshots as the change plan is refined

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.

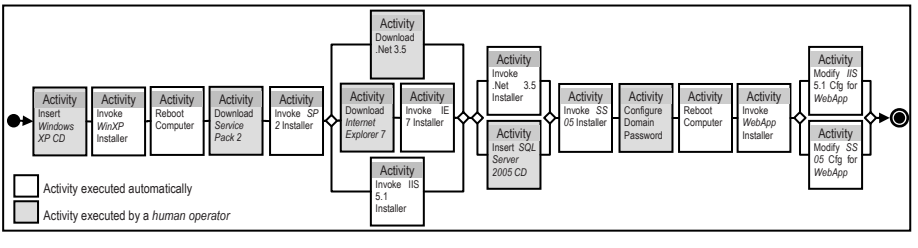

Fig. 5. Partial view of the actionable workflow for the installation of *WebApp*

[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)