CHANGEREFINERY: Assisted Refinement of High-Level IT Change Requests

David Trastour
HP Laboratories, UK
david.trastour@hp.com

Robert Fink, Feng Liu
Ludwig-Maximilians-Universität München
robert.fink@campus.lmu.de, liufeng@nm.ifi.lmu.de

***Abstract*—The IT Infrastructure Library (ITIL) is a set of best practices that are widely accepted for IT service management. Change management is a core ITIL process that oversees the handling of IT changes and ensures that all change requests are carefully prioritised and authorised, that business and technical impacts are understood, and that required resources are available. During this process, IT operations teams first need to understand the change requests that are generated by business and IT personnel. They must then develop and execute concrete IT change plans for each request. The increasingly large and complex IT environment (people, technology and processes) presents a number of challenges to the efficient and effective design of the ever higher volume of IT changes: Change requests can be ill-defined, company policies and best practices are not systematically captured and enforced, manually designing changes is time consuming and error-prone. To overcome these issues we propose in this paper an automated planning based approach to change design. We illustrate how change knowledge can be represented to encode best practices and how to refine high-level change requests into concrete plans. A prototypical implementation shows the feasibility of the approach and demonstrates the concept of a change catalogue that can be presented to business users.**

***Index Terms*—IT Change Management, Assisted Design, Automated Planning**

I. INTRODUCTION

In order to adequately support businesses that evolve at an increasing pace, IT organisations, systems and processes must constantly be adjusted: IT must introduce new business services, and enhance or modify existing ones. Technology considerations are also a source of IT changes, for instance by adopting a new technology that is more cost efficient or by performing required maintenance on a hardware or software component. For these reasons, IT organisations must cope with an increasing large number of changes. The IT Infrastructure Library (ITIL) [1] is a set of widely adopted [2] best-practices for IT service management that defines common vocabularies and processes for service strategy, service design, service transition, service operation and continual service improvement. To ensure an efficient and prompt handling of all IT changes, ITIL recommends implementing a Change Management process to funnel all Requests for Change (RFCs) to a Change Advisory Board (CAB). The Change Management process ensures that RFCs are carefully evaluated, prioritised and authorised, that their business and technical impacts are understood, and that they are scheduled in order to meet human and technical resource requirements.

While implementing ITIL change management goes some way towards a more effective handling of IT change, a high volume of changes can still put a strain on IT operations team. We have observed in some HP customer sites volumes of more than one thousand change requests per day. Without automation, decision-support and knowledge management tools, handling such volumes of changes can become problematic, as bottlenecks and inefficiencies appear in the process.

In this paper, we concentrate on the change design step, in which one or several IT practitioners develop a detailed plan of action to deploy the change onto the IT infrastructure. To the best of our knowledge, designing IT changes is still a manual process in practice, and no tool support exists that can assist an IT practitioner in deriving a detailed plan of action from the textual description provided in the RFC.

We have identified four shortcomings in this current state of affairs. First, there can be an information impedance mismatch between change requesters and IT practitioners. In a survey conducted to identify the main issues faced by change managers [3], the problem of ill-defined RFC was ranked among the three most important challenges. Indeed, RFC can be raised by technicians, for instance for the remediation of a security risk, but also by business users, for instance to change the behaviour of a business service. While technical requesters may have enough knowledge and technical vocabulary to describe the RFC to the appropriate level of details, this is often not the case for a business requester. The information provided by the business user tends to be incomplete and/or unclear for the IT technician. The second problem we have recognised is that best-practices and IT policies are usually stored in unstructured formats (documents, web pages, presentations, or emails) and there is no systematic way of capturing this knowledge and making sure it is reused. The third problem we have identified is that manually designing changes is a time consuming activity that puts of lot of strain on the IT operations teams; this is exacerbated if the design process requires multiple iterations between the requester and the IT practitioners to clarify the meaning of the change request. Finally, manually designing change plans is error-prone since the validation needs to be done manually.

In the next section, we show that several aspects of change management have been addressed in the literature, but much still needs to be done to address these four problems in IT change design.

In this paper, we present a solution for assisting in the refinement of high-level change requests into concrete actionable change workflows. Our solution introduces the notion of a change catalogue which provides a clear interface between change requesters and IT practitioners and reduces the need to rely on textual descriptions. Best practices are captured and reused in the decomposition of change requests into change workflows. Finally, corporate IT policies are modelled and enforced in change designs. Our solution is evaluated through the use of CHANGEREFINERY, a prototypical implementation of our assisted change design system.

The remainder of this paper is organised as follows. The next section examines how our approach relates to existing work in the fields of IT change management and of applications of automated planning techniques. We then introduce the concept of a change catalogue to present to change requesters and describe the information models to encode change knowledge (Section III). Section IV presents the architecture of the solution and details the algorithms used in the refinement of high-level change tasks. Section V describes details and experimental results of our prototypical implementation, CHANGEREFINERY. Finally, we conclude by presenting our future work intentions.

II. RELATED WORK

Most IT management software vendors provide some support to implement the change management process, as laid out in the ITIL Service Transition book [4]. Process management tools such as HP Service Manager or BMC Remedy Change Management help to track the evolution of a change in all phases of its lifecycle, but they provide no decision-support functionality for change design. In another class of products, orchestration capabilities for automated provisioning have been developed in solutions such as HP Operations Orchestration or Tivoli Intelligent Orchestrator. While these solutions provide graphical workflow editors, all decisions made during the design process are left to the technician using the editor, and no assistance is provided.

With CHAMPS (Change Management with Planning and Scheduling), Keller et al. [5] produced the seminal work for automation and optimisation in change management. In this work, change design was inferred from software and hardware dependencies, and was primarily targeted at software installation and de-installation. Knowledge reuse and the concept of high-level change requests were not the topic of this work.

Cordeiro et al. [6] have proposed CHANGELEDGE, a conceptual solution for the automated refinement of preliminary change plans into actionable workflows. After defining a conceptual model, CHANGELEDGE uses the concept of change templates to capture and reuse change knowledge in the design of recurrent or similar IT changes. CHANGELEDGE and our solution, CHANGEREFINERY address different issues in the problem of IT change design. First, CHANGELEDGE requires a skilled technician to understand the textual description found in the RFC and to draft a preliminary plan. It does not address the information impedance mismatch problem between change requesters and IT practitioners mentioned in section I. Also, CHANGELEDGE does not provide mechanisms to encode best practices, our second challenge; it generates any sequence of steps in which dependencies and constraints between activities are satisfied. There are two issues with this approach: on one hand, it assumes complete and accurate knowledge of the constraints and dependencies on the IT model, and on the other hand, IT practitioners are usually only comfortable using tried and tested methods and would only trust a small subset of such automatically generated workflows. Finally, while CHANGEREFINERY and CHANGELEDGE solve different problems, they can be seen as complementary: the former *assists* an IT practitioner in refining high-level change tasks, the latter *automates* the generation of low-level workflows. We will explore how to combine the two approaches in a future work.

Other aspects of change management such as change scheduling have been proposed. Sauvé et al. [7] take the example of change scheduling to demonstrate linkage models between IT availability metrics and business objectives. Rebouças et al. [8] use the linkage model from [7] and formalise the problem of business-driven scheduling of changes, in which changes need to be assigned to maintenance windows. Trastour et al. [9] go one step further by breaking down changes into the elementary activities that compose them and by providing a scalable solution to the change scheduling the problem. Finally, Setzer et al. [10] studied the impact of IT changes onto business processes and considered stochastic durations. The investigation of these business impact analysis techniques for the design of changes as well as the integration of planning and scheduling of changes is left for future work.

In the area of IT infrastructure design, Ramshaw et al. [11] propose a constraint programming approach to design IT infrastructure from SLA requirements. The main difference with what we propose here is that [11] focuses on generating IT system configurations that satisfy capacity and performance requirements, while we are interested in the processes for transforming existing IT systems.

Finally, the general field of automated planning [12], an area of artificial intelligence, provides techniques to generate plans of actions meeting certain objectives. This field has evolved from classical planning to more specialised forms of planning. In particular, Hierarchical Task Network (HTN) planning [12], [13] is well suited for domains in which plans are known to exhibit a hierarchic structure; the planning algorithm is then guided by predefined hierarchical decomposition templates. Although automated planning research has had successes in many domains (Mars exploration [14], crisis intervention [15], workflow planning in Grid computing [16], analysis and life cycle management of plans [17]), to our knowledge, it has not been applied to change design.Our proposed assisted design solution adopts HTN planning principles on knowledge representation and plan generation.

III. Information Model for Change Plan Refinement

We present our solution in two steps; in this section we describe the information model used to represent change knowledge, and in the following section we will present the architecture of our solution and the reasoning techniques for assisted change design that make use of this knowledge.

To minimise IT service disruptions and the need for further corrective changes, IT practitioners favour to use procedures that are fully understood and have previously been tested. In our experience, even within large IT organisations, these procedures can be seen as *change recipes*, since they are rarely formalised; they may be documented in unstructured documents shared within workgroups, or implicitly memorised by managers or technicians. These recipes are typically scattered within the IT organisation, and a single change may involve recipes from different IT workgroups, such as a database workgroup or a Unix workgroup. Moreover, change recipes can be concrete (*e.g.* the steps to build a Windows 2003 web server to corporate standards), or abstract (*e.g.* the steps to consolidate a set of servers in a data centre) and would require further refinements to obtain an actionable workflow. We propose to formalise these change recipes in the form of *best practices* for change design to assist IT pratitioners in producing actionable change plans. In addition to these, IT practitioners must also consider corporate *IT policies*. Such policies can impose constraints on the possible configuration of the IT infrastructure (*e.g.* all external facing web servers must have a firewall) and on the processes used to perform changes (for instance, all changes affecting the payroll system must be authorised by the financial controller). Finally, IT practitioners must take into account the state of IT Operations, *i.e.* the state of the IT infrastructure and of the IT processes they need to interact with in order to carry on their work. A Configuration Management Database (CMDB) is typically used to model and store infrastructural information and interdependencies between all entities (*configuration items* or *CI*s) that potentially take part in the execution phase of a change. Other tools, such as a service management tool, track the state of IT processes.

Fig.1 depicts the main use case our solution is addressing. A business user, the *change requester*, has limited IT knowledge and needs to raise a request for change (RFC). Instead of relying solely on textual description and being confronted with the problem of ill-defined RFC mentioned in Section I, the change requester browses the *change catalogue*, selects one of the high-level change tasks presented to her and fills in the required parameters. The RFC is later handled by one or several IT practitioners to be designed, documented, tested and implemented. Using our solution, the RFC is continuously refined into smaller and smaller change tasks, until eventually an actionable workflow is generated. The output uses best practices from various IT workgroups and complies with all corporate IT policies. A *change knowledge manager* role is responsible for maintaining the change catalogue and for interacting with IT practitioners to ensure that best practices for change design are captured and maintained.

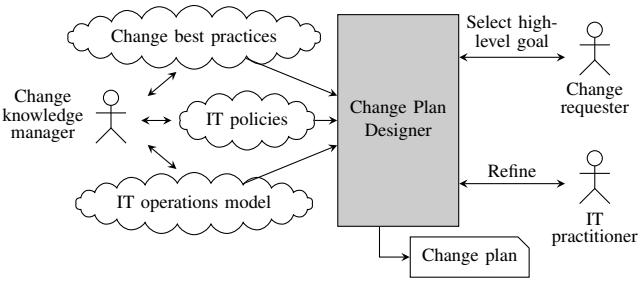

Fig. 1. Assisted design use case

Having presented the different sources of change knowledge and our assisted change refinement use case, we now describe our approach to provide an integrated view on change knowledge. We first present the IT operations model, the change catalogue and best practices model, the main contribution of this information model, and finally the IT corporate policies model.

A. IT Operations Model

Our solution requires an information model representing concepts from IT Operations that are used in the definition of change activities. We assume that an object-oriented formulation exists and that all classes have a root class, in order to have a generic way to refer to all such IT concepts. The model may represent hardware and software components, IT technicians and their skills, or steps in IT processes. For the purpose of this paper, we have chosen to instantiate the IT Operations model as a subset of the Common Information Model (CIM) [18]. Being an object-oriented framework, CIM defines hierarchies of configuration items through specialization and dependencies between items by associations, compositions and aggregations. The root class of the CIM model is *ManagedElement*.

The remainder of this paper will present a J2EE scenario (Fig.5 and Fig.6) featuring hierarchies of hardware components (web servers, load balancers, database servers), different kinds of databases, J2EE application containers, J2EE applications connected to databases, JDBC resources, and load balancer applications. Additionally, in compliance with the ITIL guidelines for configuration management systems, technicians and their respective skills in operating the above-mentioned entities are also included. This J2EE scenario can be extended to support other domains or more complicated infrastructures by adding new classes and dependencies from CIM. Alternatively, the CIM model could be substituted by models from commercial CMDB products.

B. Change Catalogue and Best Practices Model

We propose a model whose aims are on one hand to provide a clear interface between change requesters and IT practitioners, and, on the other hand, to formalise the sets of best practices for IT changes. Fig.2 depicts a UML representation of this information model.

We introduce the notion of hierarchically decomposable change *Tasks*. In our model, change Tasks convey the meaning of *what* will be done, not *how* it will be done. For instance, deploying a web server is a Task that could be implemented in many ways, depending on the expected traffic or on whether the server will be externally facing. A business user will typically not be concerned with the technical implementation details of a change, and should only have to deal with change Tasks. Change Tasks can represent activities at different levels of abstraction, from high-level Tasks that a business user may ask for (*e.g.* altering a business process in an SAP system) to low-level technical Tasks (*e.g.* installing an Oracle database).

To be able to represent workflows supporting parallel and sequential actions, our model includes *TemporalConstraint*s which provide qualitative ordering constraints between Tasks. A *TaskNetwork* is hence a partially ordered set of Tasks.

The *ChangeCatalogue* is the subset of predefined Task networks that an IT organisation chooses to expose to its users. We propose the change catalogue as analoguous to the ITIL service catalogue [4]: it contains a set of high-level IT changes that are available to change requesters. An RFC hence contains an instance of a Task network.

A change Task can have any number of *Variable*s. Variables have a name, a type, which is either a data type (*e.g.* an integer representing a number of users) or a class from the IT operations model (the *WebServer* class), and an instance value. Values for all variables present in the original Task network (from the RFC) must be specified by the change requester. As we will see in the following section, the change design process consists of recursively refining change Tasks into sub-tasks until a concrete workflow is obtained. Variables are then used to pass parameters between a Task and its sub-tasks.

Borrowing from the Hierarchical Task Network (HTN) planning paradigm [12], a change Task can be realised by one of two *Refinements*, an Operator or a Method. *Operators* are atomic activities that have known *Effects* on the IT Operations model. For instance, Operators accomplishing the *Install-Databse* Task in Fig.6 have the effect of adding a Database CI to the Knowledge Base. *Methods* encode the best practices or recipes mentioned earlier, and are the mechanism we use to refine a Task into further lower-level Tasks. In our model, several Operators or Methods could implement the same Task, leading to different effects and Task decompositions. For instance, the *InstallApplication* Task in Fig.5 could be decomposed by the shown simple sub-workflow (*DeployApplication*, *InstallDatabase*, *AddDbResourceToContainer*), but one could also define an alternative Method to refine the same Task, for instance with additional authorisation and backup steps.

Because not all Refinements (Operators and Methods) may be applicable for a given Task at all times, Refinements are guarded by a *Condition* on the IT Operations model. The Refinement can be used only when the Condition is satisfied. For example, every Refinement of the Task *AddContainerToLb* in Fig.5 requires the existence of a *J2eeContainer* and a *LoadBalancer* CI in the Knowledge Base.

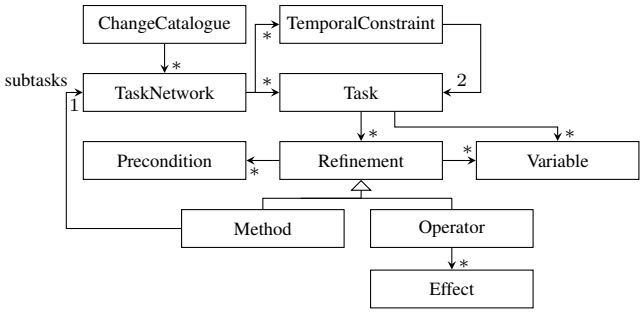

Fig. 2. Change Catalogue and Best Practices Model

C. IT Policies

Change plans typically have to comply with corporate policies resulting either from business requirements, service level agreements (SLA) or technical considerations. We identify two different classes of policies which constrain the change plan design process. First, policies may qualify the permitted states of the IT infrastructure, for example by proscribing the use of certain software versions and by only allowing certain hardware and sofware combinations. Following the ITIL recommendation, this set of policies should be made available in the Definitive Media Library [1]. The second type of policies addresses the logical and temporal structure of change plans. Examples of such policy could be the obligation to perform systematic backups before certain classes of change Tasks or the need to perform changes during maintenance windows specified in an SLA.

Our solution is agnostic to the choice of the policy language (*e.g.* RuleML [19]). The only requirement is the ability to express rules on the IT operations model and on the change catalogue and best practices model.

IV. Assisted Refinement of Change Tasks

Having described the information view of our approach, we now present the conceptual architecture, give the requirements for the main components of the solution, and explain the algorithm used to refine high-level change Tasks into actionable workflows.

Fig.3 depicts the architecture of our solution. The *Change Planner* is the core component that compiles change plans from the various information repositories. The *IT Knowledge Base* holds the IT Operations model and acts as a simulation environment for the Planner. The *Change Catalogue repository* holds the change catalogue and best practices model and includes Tasks, Methods and Operators. The *Temporal Reasoner* is used by the Planner to enforce temporal consistency while building the plans. Finally, the *Policy Repository* and *Engine* store and enforce the IT corporate policies.

A. IT Knowledge Base

The IT Knowledge Base component stores the IT Operations model and acts as an abstract interface between the Planner and different types of IT infrastructure and operational databases. We now describe the functionalities of the IT

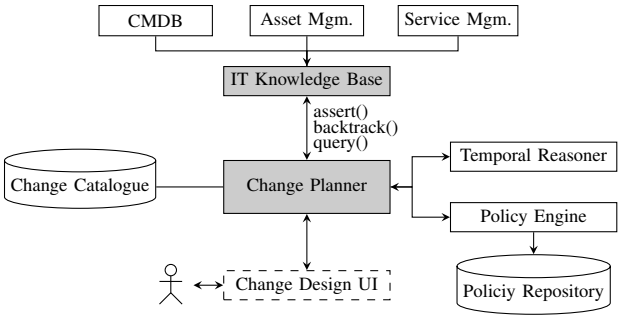

Fig. 3. Architecture of the change plan designer

Knowledge Base. In order to simulate the effects of Operators, the Change Planner must be able to *assert* facts. In our object-oriented model, this means creating new objects, creating or changing dependencies between objects, and changing the values of attributes of objects. Because the Change Planner is implemented as a backtracking search (see Section IV-C), the IT Knowledge Base must be able to *snapshot* its current state and *undo* assertions. In other words, it must provide versioning capabilities. At various steps in the design process, the Planner must be able to verify if conditions are satisfied in order to check what Operators and Methods are applicable. Hence, the Knowledge Base must also provide a *query* mechanism that returns tuples of objects matching a given search criterion. Finally, it must *detect causal dependencies* between the effects of Operators and the Conditions of Refinements to ensure that the generated plans are temporally consistent. This last feature is detailed in Section IV-D.

B. Change Catalogue Repository

The Change Catalogue repository stores the change template and best practices model, or in HTN terms, the Tasks, Methods and Operators, and their depending objects. This repository is accessed by change requesters when they need to author an RFC. We recognise that to make the change catalogue easily accessible to requesters, change Tasks need to be annotated and categorised, but this falls outside the scope of this paper. The Change Catalogue repository is also accessed by the Planner during the refinement process, and this simply consist of navigating through the associations of the model, for instance to find suitable Refinements for a given Task.

C. Change Planner

The Change Planner is the core component of the system and uses a variation of the HTN planning algorithm. Its input is a Task network specified by the change requester. By exploring the space of possible decompositions for the list of Tasks in a given state, the algorithm continuously decomposes Tasks by replacing them by their sub-tasks as defined in the decomposition Methods, until the initial set of goal Tasks is transformed into a list of atomic Operators (then called a list of actions or a plan), which is the algorithm's output. The Planner is tightly coupled with the IT Knowledge Base which acts as the simulation environment for the planning algorithm: Effects of Operators from partially refined workflows change the state of the Knowledge Base, queries validate the applicability of Methods and Operators and return variable bindings. A formal description of the HTN algorithm is out of the scope of this paper, since it is covered to great extent in the automated planning literature [12]. We will concentrate on our contributions which are to provide clear separation of concerns between the various information repositories, the adaptation to object-oriented models, and the consideration of duration and temporal constraints.

The classical HTN formulation [12], [13] relies on first-order logic to define Tasks, Methods and Operators; variable bindings are found and passed by unification, the state of the world is given by a set of atoms and preconditions and effects are sets of literals. Because of the predominance of object-oriented models for IT infrastructure and operations, such as CIM, we do not express the infrastructure data model in terms of first-order logic. Instead, we rely on the IT Knowledge Base which can be queried to return all objects, such as configuration items, matching a Method's or an Operator's preconditions.

Our adoption of the classical non-deterministic partial-order HTN algorithm is shown in Fig.4. Every recursive invocation of the algorithm takes as input a reference to the current state of the Knowledge Base (kb), the Task network (tn) to be refined, and a set of variable bindings from previous invocations. The algorithm picks a Task without predecessors in tn (*forward decomposition*, line 3) for further decomposition and chooses one of its applicable[1] *Refinements* (lines 4,6) and its respective variable bindings (line 7); one variable binding is selected[2] and added to the total set of bindings. If the selected Refinement is an Operator, the refined Task is removed from the Task network (line 11) and its effects are applied to the Knowledge Base (line 10). Before recursively decomposing the remaining Task network, the Temporal Reasoner and Policy Engine components are invoked (lines 13, 14) to check then preliminary plan's temporal consistency and validity with respect to policies. If both checks are successful, the Operator is added to the plan (line 16). If on the other hand the chosen Refinement is a Method, the respective Task is substituted with the Method's sub Tasks and the resulting Task network is recursively decomposed (line 18).

A deterministic implementation of the algorithm searches the whole space of possible decompositions by selecting a refinement and backtracking in the case of a failure.

D. Temporal Reasoner

Real-world change plans consist of durative and parallel activities, both features that are not supported by the classical HTN formulation where plans are sequences of atomic actions.

[1]A Refinement is defined to be applicable in the current state of the Knowledge Base if its precondition query has a non-zero number of matching tuples in the Knowledge Base.

[2]Note that the concept of bindings does not increase the branching factor when compared with classical HTN, as different bindings for Refinements correspond to different possible substitutions in classical HTN.

```
1   function HTN(kb, tn)
2     if tn = ∅ then π ← empty plan; return true
3     t ← tn.getFirstTask()
4     refiners ← t.getApplicableRefiners(kb)
5     if refiners = ∅ then return false
6     r ← refiners.choose()
7     bindings ← r.computeBindings()
8     b ← bindings.choose()

9     if r is an operator then
10      kb.assert(r.getEffects(b))
11      tn.removeTask(t)
12      π.add(r)
13      if [TemporalReasoner.PathConsistency(π) and
14          PolicyEngine.Check(π)] then HTN(kb, tn)
15      else return false
16    else if r is a method then
17      r.decompose(tn, t, b)
18      if [TemporalReasoner.PathConsistency(π) and
19          PolicyEngine.Check(π)] then HTN(kb, tn)
20      else return false
```

Fig. 4. Non-deterministic algorithm for partial order forward decomposition HTN planning with Knowledge Base and STN integration. Inputs: kb: Knowledge Base, tn: goal Task network. Output: true and variable π stores a solution plan, if one exists, false otherwise.

Total order plans may be acceptable for applications in which the plan is executed in a sequential fashion (*e.g.* because all actions have the same resource requirements and therefore cannot be accomplished simultaneously) but not for change plan design: we are seeking a change plan that conserves the parallel structure of the workflow templates and only contains additional ordering constraints where imposed by interactions between effects and preconditions of Operators. For instance, Operators accomplishing the Tasks *InstallDatabase* and *InstallJ2eeContainer* (Fig.6) are usually independent and should remain temporally unconstrained if they effect different server CIs (ws and ds) as in the depicted case. On the other hand, if the database and J2EE container were chosen to be installed on the same server, additional ordering constraints are necessary to ensure that the installation activities do not come into conflict with required resources.

In order to obtain plans with parallel branches and enhance the planner with temporal reasoning capabilities for durative actions, we have incorporated in our architecture a temporal reasoning component which maintains a Simple Temporal Problem (STP) data structure. An STP [20], [21] is a triple (X, D, C) where X is a set of time point variables, D defines the domain for every variable and C is the set of binary constraints between time points. For STPs the path-consistency algorithm is a sound and complete method [20], [21] to check the consistency of the constraint problem and compute the minimal equivalent constraint network.

As the change plan is being refined, temporal constraints are added to the STP. Every plan π has an associated STP with an initial time point, INIT, and for every Operator $o_i \in \pi$ two additional time points, start(o_i) and end(o_i), representing the start and the end of the respective Operator. Whenever an Operator o_i is added to the intermediate plan by the HTN algorithm, a qualitative ordering constraint[3] is posted between o_i and all previous Operators $o_j \in \pi, j < i$ whose effects contribute to the preconditions of o_i. To allow for this mechanism, the Knowledge Base component must be able to detect causal dependencies between preconditions and effects on configuration items:

- A change record must be stored for every configuration item, specifying when and by which Operator it was changed
- The change record must be made available to the Planner for every configuration item in a query result

Furthermore, every time a Task is decomposed, temporal constraints between its sub-tasks and the remaining Tasks are posted to enforce the parallel or sequential structure of the workflow encoded in the decomposition Methods. A durative action a_i with duration d is encoded by the constraint $[0, d]$ between the time points start(a_i) and end(a_i), a Task T with a deadline t generates the constraint $[0, t]$ between the time points INIT and end(T).

After updating the STP with new time points or temporal constraints, a path consistency algorithm [20], [21] can be used to check the consistency of the STP and to compute its equivalent minimal network. Checking the consistency of the STP for every potentially new action in the preliminary plan can be accomplished in $O(n^3)$ time and thus helps to efficiently prune large chunks of the HTN search space and ensures that all plans are temporally valid.

E. Policy Engine

In every planning step (Fig.4, line 14), the Planner asks the Policy Engine if the intermediate change plan complies with policies from the policy repository: policies on the IT Operations Model or on the plan structure. Failure in the validation of such policies causes the Planner to backtrack. The benefit of this approach is that any off-the-shelf rule engine, such as the Drools system, can be used as the Policy Engine.

V. VALIDATION

In this section, we first describe CHANGEREFINERY, a working prototype based on the Java programming language that demonstrates the feasability of the approach. We then discuss qualitive and quantitative results we have obtained from this prototype.

The IT Knowledge Base component implements the CIM-based IT operations model and exposes the required interfaces assert(), rollback() and query() (see Section IV-A) to the Planner component. We use the Hibernate object-relational persistence service to realise an object-oriented database for configuration items. Queries against the Knowledge Base are expressed in the Hibernate Query Language (HQL), whose object-oriented query interface aligns nicely with inheritance-based modelling concept of CIM.

The assert() and rollback() interfaces are implemented via database snapshots: Every assertion triggers the creation of

[3] A qualitative ordering constraint between two actions, a_i and a_j, is posted as a constraint $[0, \infty]$ between the time points end(a_i) and start(a_j).

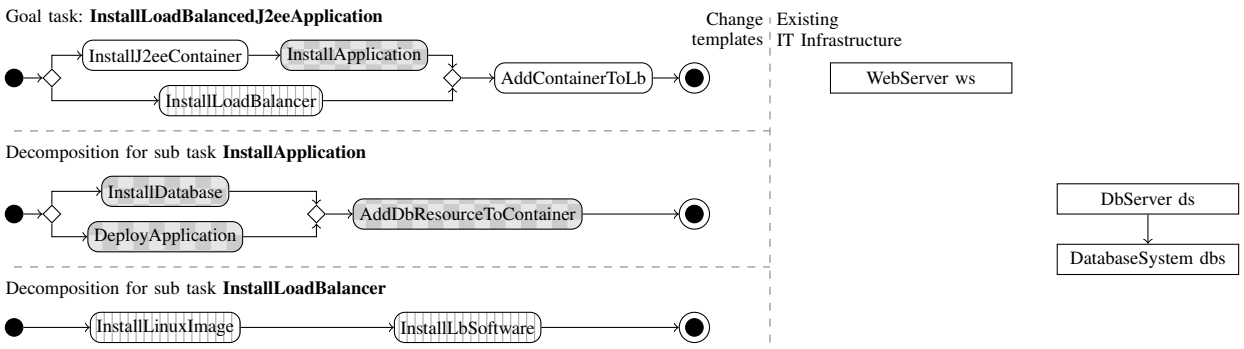

Fig. 5. Simplified example for change templates and IT infrastructure before plan execution

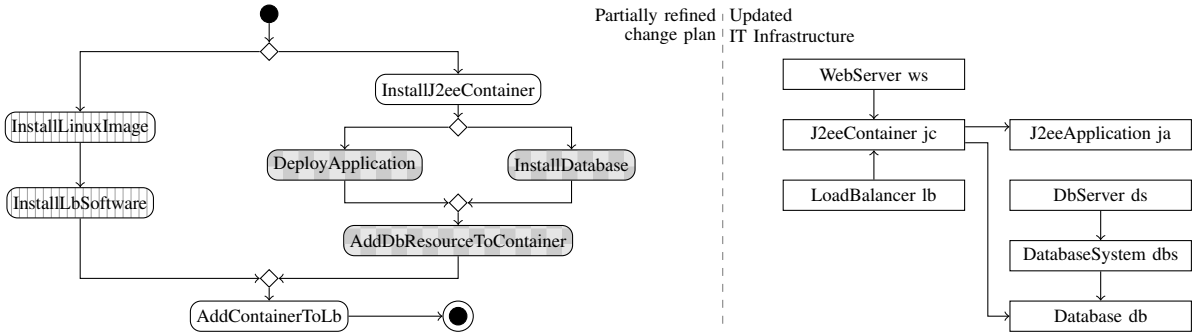

Fig. 6. The goal Task *InstallLoadBalancedJ2eeApplication* (Fig.5) was refined by workflows for the Tasks *InstallApplication* and *InstallLoadBalancer*. The integrated knowledge representation for workflows and IT infrastructure allows to compute the effects on the infrastructure as shown on the right side: The existing CIs ws, ds, dbs are integrated with additional new configuration items which entered the Knowledge Base by virtue of effects of the refined sub-tasks.

new snapshot and every rollback reverts the database to a previous state.

We use a backtracking implementation (see Fig.4) of the partial-order forward decomposition HTN algorithm [12], modified to support partial-order plans and temporal enhancements by integration of an STP data structure as explained in Section IV.

For the causality dependency checking between effects and conditions, we have implemented a "CI locking" technique. A natural alternative approach to build such causal relationships as proposed in [22] cannot be incorporated in our case, because it relies on a logic language representation of the state of world and makes assumptions that are too strong on the representation of the Knowledege Base. Our pessimistic configuration item locking scheme enforces that the Effects of new Operators cannot invalidate the preconditions of any previous Operator. This pessimistic solution has the merit of being simple to implement and was adequate for our experiments. Finer-grained locking to the level of associations and attributes would require a deeper modeling of effects and conditions and was out of the scope of this paper.

Regardingg the use of IT policies, we have experimented with policies that affect CIs (*e.g.* the database server *DbServer* can only be Oracle and it must run on a Windows 2003 or HPUX 11i machine) and with policies that affect the change process (*e.g.* an *Approval* step must be present for changes that affect a business critical application). When the Change Planner explores the change search space, it calls the Policy Engine to filter out plans in which at least one IT Policy is violated. In our current approach, we evaluate the whole set of IT Policies at each step of the HTN planning process. Performance problems may arise on large rule sets and one may need to be more selective in the set policies that are evaluated.

We now discuss qualitative and quantitative results we have obtained from our experiments with CHANGEREFINERY. Using the proposed approach, we have successfully modelled several real-life examples of change (installation, migration and maintenance involving hardware and software components). Throughout this paper, we have described the simple case of installing a J2EE application (Fig.5 and Fig.6). Allowing a choice of Methods and Operators for a single Task has proved to be a very useful feature to express the variations in changes based of various circumstances (security or availability requirements, hardware and software dependence).

In terms of quantitative results, we have run the planner on examples including up to 50 tasks involving an IT infrastructure of up to 100 components. If the Change Planner is left running in fully automated mode, without any user intervention, it is capable of automatically generating plans for realistic enterprise examples in the order of seconds or minutes, depending on the size of the IT model. While automatically generating change plans is useful for performance and system testing, our goal is to assist an IT practitioner in

refining plans. The user interface of CHANGEREFINERY provides *mixed-initiative* interaction of the IT practitioner with the planning system. At each step of the HTN planning process, the user can let the planner exlore the search space and is presented with the set of feasable decomposition Operators or Methods for the refined Task. Also, when different sets of variable bindings are returned from Knowledge Base queries, the user can choose which binding she prefers. In this usage model, we never observed computation greater than a few seconds at each step of the planning process. Indeed, by making decisions on Refinements and Bindings, the user is prunning the search space, making the planning process easier. In all our experiments, we observed that the Knowldge Base, and in particular our database snapshot mechanism, was the bottleneck. This problem would need to be addressed to be able to scale to models of large enterprises and datacentres with thoudands of configuration itemns.

VI. CONCLUSION AND FUTURE WORK

We have proposed a generic architecture for integrating various information sources necessary for IT change design. The Hierarchical Task Network planning paradigm augmented with temporal constraints is used to model and hierarchically refine change Tasks into consistent change plans, considering the involved IT infrastructure and policies or constraints on such plans.

By building a prototype and testing it with real-life examples, we have demonstrated the viability of the approach. To substitute the IT Operations model of our prototype with real IT Operations data sources (CMDB, service management or asset management products) would require to solve several information management problems, such as model mapping and integration, and the ability to have a Knowledge Base that scales to large data sets. Because of the mixed-initiative nature of the solution, measuring the *efficiency* of the tool (its speed for generating plans) is not the most significant metric. Further validation would require to deploy the tool in a live environment, and to measure its *effectiveness*, *i.e.* the improvement of productivity of IT practitioners in designing IT changes and the improvement of quality of the IT plans.

Our immediate next steps are to introduce decision-support features in our mixed-initiative scenario. In this paper, an IT practitioner guides the design of the change plans by choosing Refinements and variable bindings. While we still want to leave the decision to the human operator, we would like to assist her in making the best choices by presenting her metrics, such as time, cost or risk, and to help her understand the trade-offs of various change designs.

Finally, we also intend to expand the proposed solution in two ways. First we will investigate the feasibility of a hybrid planning approach, *i.e.* interleaved state-space and HTN planning, to be able to reason on incomplete domain knowledge. We will also study how our solution can be combined with proposed change scheduling solutions (see section II) and how to resolve temporal and resource constraints into implementation schedules.

ACKNOWLEDGMENT

The authors would like to thank C. Bartolini, W. Cordeiro and A. Boulmakoul for valuable comments and suggestions.

REFERENCES

[1] *IT Infrastructure Library*. Office of Government Commerce, UK, 2003.
[2] D. Dubie, "Itil adoption increases in u.s., proficiency still lacking." [Online]. Available: http://www.networkworld.com/news/2008/022908-itil-adoption.html
[3] R. Rebouças, R. Santos, J. Sauvé, and A. Moura, "The HP-Bottom Line Project, IT Change Management Challenges - Results of 2006 Web Survey, Technical Report DSC005-06," Computing Systems Department, Federal University of Campina Grande, Brazil, Tech. Rep., 2006.
[4] *IT Infrastructure Library, "ITIL Service Transition"*. Office of Government Commerce, UK, 2003.
[5] A. Keller, J. Hellerstein, J. Wolf, K. Wu, and V. Krishnan, "The CHAMPS System: Change Management with Planning and Scheduling," in *9th IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2004, pp. 395–408.
[6] W. Cordeiro, G. Machado, F. Daitx, C. Both, L. Gaspary, L. Granville, A. Sahai, C. Bartolini, D. Trastour, and K. Saikoski, "A Template-based Solution to Support Knowledge Reuse in IT Change Design," in *11th IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2008, pp. 355–362.
[7] J. P. Sauvé, R. Rebouças, A. Moura, C. Bartolini, A. Boulmakoul, and D. Trastour, "Business-driven decision support for change management: Planning and scheduling of changes," in *17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM)*. Springer, 2006, pp. 173–184.
[8] R. Rebouças, J. P. Sauvé, A. Moura, C. Bartolini, and D. Trastour, "A decision support tool to optimize scheduling of it changes," in *10th IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2007, pp. 343–352.
[9] D. Trastour, M. Rahmouni, and C. Bartolini, "Activity-based scheduling of it changes," in *International Conference on Autonomous Infrastructure, Management and Security (AIMS)*. Springer, 2007, pp. 73–84.
[10] T. Setzer, K. Bhattacharya, and H. Ludwig, "Decision support for service transition management," in *IEEE Network Operations and Management Symposium (NOMS)*, 2008.
[11] L. Ramshaw, A. Sahai, J. Saxe, and S. Singhal, "Cauldron: A policy-based design tool," in *7th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2006)*. IEEE Computer Society, 2006, pp. 113–122.
[12] D. Nau, M. Ghallab, and P. Traverso, *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., 2004.
[13] K. Erol, J. Hendler, and D. S. Nau, "Htn planning: Complexity and expressivity," in *Twelfth National Conference on Artificial Intelligence (AAAI-94)*. AAAI Press/MIT Press, 1994, pp. 1123–1128.
[14] T. Estlin, R. Castano, R. Anderson, D. Gaines, F. Fisher, and M. Judd, "Learning and Planning for Mars Rover Science," in *IJCAI 2003 workshop notes on Issues in Designing Physical Agents for Dynamic Real-Time Environments*, 2003.
[15] J. Fernández-Olivares, L. A. Castillo, Ó. García-Pérez, and F. Palao, "Bringing users and planning technology together. experiences in siadex," in *Sixteenth International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI, 2006, pp. 11–20.
[16] Y. Gil, E. Deelman, J. Blythe, C. Kesselman, and H. Tangmunarunkit, "Artificial intelligence and grids: Workflow planning and beyond," *IEEE Intelligent Systems*, vol. 19, no. 1, pp. 26–33, 2004.
[17] B. Srivastava, J. Vanhatalo, and J. Koehler, "Managing the life cycle of plans," in *AAAI*, M. M. Veloso and S. Kambhampati, Eds. AAAI Press / The MIT Press, 2005, pp. 1569–1575.
[18] "Common Information Model (CIM)," Distributed Management Task Force, 2007. [Online]. Available: http://www.dmtf.org/standards/cim/
[19] "RuleML." [Online]. Available: http://www.ruleml.org/
[20] R. Dechter, I. Meiri, and J. Pearl, "Temporal Constraint Networks," *Artificial Intelligence*, vol. 49, no. 1-3, pp. 61–95, 1991.
[21] R. Dechter, *Constraint Processing*. Morgan Kaufmann, 2003.
[22] L. A. Castillo, J. Fernández-Olivares, Ó. García-Pérez, and F. Palao, "Efficiently handling temporal knowledge in an htn planner," in *Sixteenth International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI, 2006, pp. 63–72.