RC23693 (W0508-088) August 15, 2005
Computer Science

IBM Research Report

A Best Practice Approach for Automating IT Management Processes

Aaron B. Brown, Alexander Keller
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

A Best Practice Approach for Automating IT Management Processes

Aaron B. Brown, Alexander Keller
IBM Thomas J Watson Research Center,
P.O. Box 704, Yorktown Heights, NY 10598, USA
{abbrown|alexk}@us.ibm.com

Abstract—**In an effort to bring more structure to the task of automating systems management, we introduce an approach for incrementally introducing best-practice-based automation into IT service delivery and management. Our approach is process-centric, using best-practice processes to structure the automation, scope out high-value automation opportunities, identify interaction points between automation and its broader environment, and expose the overall impact of automation on the service delivery organization. We illustrate our approach by applying it to a specific example: reducing the cost of IT Change Management via a system that automates changes in the domain of software lifecycle management (such as installation/deployment and configuration).**

Keywords-Change Management; IT Process Management; ITIL; IT Process Automation

I. INTRODUCTION

The economics of IT service delivery are rapidly becoming dominated by labor costs: as IT environments grow in scale and complexity, the human cost of the processes needed to manage them are quickly growing and will soon dwarf the costs of the environments themselves. To counteract this trend, IT departments of enterprises and service providers are increasingly looking to automation as a means of containing and even reducing the labor costs of systems management. But creating automation is not a trivial task, and successes to date have mostly been point solutions to specific labor bottlenecks.

This paper attempts to provide some needed structure to the task of automating IT service delivery. Using the key area of change management as a running example, we illustrate a six-step, process-centric methodology for going from ad-hoc, manually-implemented service delivery processes to automated best-practice service delivery. Our focus is not on total automation; rather, we recognize the difficulties of implementing such drastic changes to an established service delivery process, and instead provide a more incremental approach that targets the highest-value automation opportunities.

A. Focus on Change Management

We choose to focus on IT Change Management—a process by which information technology (IT) systems are modified to accommodate considerations such as software fixes, hardware upgrades and performance enhancements—as it is at the heart of effective service delivery and a common focus of automation efforts. The importance of change management is underscored by widely reported outages of prominent Internet sites due to misconfiguration of host systems and routers. In the case of a large Internet service provider in the US, about 50% of outages happen during maintenance windows. A study shows that operator errors account for the largest fraction of failures of Internet services [7] and hence properly managing changes is critical to availability of IT services.

We choose Change Management as well because it is a process that offers many automation challenges. The **organizational frameworks** behind existing change management processes tend to be ill-defined and highly variable in detail and formality. This makes it difficult to identify tasks for automation or to reuse standardized automation across environments within or between IT organizations (a particular problem for corporate mergers). Additionally, effective change management requires a great deal of accurate information on **technical constraints**, such as dependencies between IT services and the infrastructure supporting them. This information is rarely documented formally or kept up to date, hindering change assessment and change impact prediction. Indeed, service providers today find it necessary to set up staging areas where changes are repeatedly tested to expose their potential impacts (and adjusted accordingly) before they can be safely deployed into production environments. This kind of ad-hoc analysis is not immediately conducive to automation.

B. Use of the IT Infrastructure Library (ITIL)

In part to overcome these challenges and to provide a rigorous structure around Change Management (or any other IT service support & delivery process), our approach to automation is built upon the foundation of the Service Support Processes of the *IT Infrastructure Library (ITIL)* [5]. The IT Infrastructure Library codifies a set of industry-wide best practices by defining IT service management processes and providing detailed descriptions of the various activities in each process, along with their input and output data. It provides a universally applicable blueprint for IT service management, which has begun to receive considerable attention, both from customers and from manufacturers of IT service management systems. Note that, while we rely on ITIL in our approach, we believe that the approach will generalize to other best-practice process frameworks that provide a similar level of detail to ITIL.

C. Outline

In the rest of this paper, we first outline our best-practice approach to automation in Section II. Sections III – V describe our application of the best-practice approach to automating key

aspects of change management, focusing on process selection, delegation of activities, and capturing information needed for automation. Section VI describes CHAMPS, our implementation of automated change management that resulted from following the methodology we introduce. We evaluate the effectiveness of CHAMPS automation in Section VII, and conclude the paper with a discussion of related work, a summary of our results, and consideration of areas for further research.

II. A Process-Based Approach to Automation

We start our discussion of automating IT service delivery by outlining the best-practice approach to automation that we have developed. This approach is based on our experiences with automating change management, along with insight we have distilled from interactions and engagements with service delivery personnel.

Our approach comprises six steps for transforming existing IT service delivery processes with automation or for introducing new automated process into an IT environment. The first step involves **(1) identifying best practice processes for the domain to be automated**.

To identify these best practices we turn to the IT Infrastructure Library (ITIL), which over the past few years has established itself as the most widely used process-based approach to IT service management. ITIL comprises several disciplines such as infrastructure management, application management, service support and delivery. The ITIL Service Support discipline [5] defines the Service Desk as the focal point for interactions between a service provider and its customers. To be effective, the Service Desk needs to be closely tied into roughly a dozen IT support processes that address the lifecycle of a service. Some examples of IT service support processes for which ITIL provides best practices are: Configuration Management, Change Management, Release Management, Incident Management, Problem Management, Service Level Management, and Capacity Management.

ITIL provides a set of process domains and best practices within each domain, but does not provide much guidance as to which domain should be an organization's initial target for automation. Typically this choice will be governed by the cost of existing activities in these various process domains: for example, an enterprise that finds itself spending large amounts of labor dollars on Configuration Management should target that process first for automation. Since we lack such a specific context in this paper, we will focus on the Change Management process domain, as it is a key ITIL domain that factors into many day-to-day service operations.

After identifying a process domain and obtaining the ITIL best practice, the second step of our automation pattern is to **(2) establish the scope of applicability for the automation**. Most of the ITIL best practice processes are quite broad in applicability and cannot be reasonably automated in one fell swoop—to do so would require a tremendous automation effort and equally significant deployment and user acceptance challenges. Thus, to bound the scope of automation it is best to target it to a particular subdomain in which the best practice process is used. For example, ITIL Change Management best practices can apply across a spectrum of Change Requests: requests to install or change software, requests to alter hardware configurations, requests to deploy entirely new IT environments, facilities management requests, documentation updates, and so on.

The most successful automation will be scoped to one or a small subset of these subdomains—a "slice" through the entire change management best practice process—for example handling only software change requests automatically while deferring other change requests to manual procedures. Once one Change Management subdomain is successfully automated, the others can follow suit, often reusing components of the existing automation. Similar arguments apply with other ITIL best practices; for example, Configuration Management can apply to managing configurations of servers, software licenses, documents, networks, storage, and sundry other components of an IT infrastructure, any subset of which is likely to provide a high-value automation opportunity.

The next step in our approach is to take the ITIL best-practice as scoped above and **(3) identify delegation opportunities**. Each ITIL best practice defines (explicitly or implicitly) a process flow consisting of multiple activities linked in a workflow. Some of these activities will be amenable to automation—such as deploying a change in Change Management—whereas others will not (such as obtaining change approvals from Change Advisory Boards). This third step in our approach involves identifying the candidate activities for automation, determining whether the value of automating them outweighs the cost of developing and maintaining the automation, and selecting a final set of process activities to automate. Note that in some cases activities that appear non-automatable can be delegated anyway. For example, if an automated Change Management system is trusted enough, change approvals might be delegated, with the result that the change management system automatically approves all well-formed automatable change requests.

The benefit of explicitly considering delegation is that it brings the rigor of the best-practice process framework to the task of scoping the automation effort. The best practice defines the set of needed functionality, and the delegation analysis explicitly surfaces the decision of whether each piece of functionality is better handled manually or via automation. Using this framework helps prevent situations like the one discussed in [2], where the cost of an automation process outweighed its benefits in certain situations.

With the delegated activities identified, the fourth step in our automation approach is to **(4) identify links between delegated activities and external activities, processes, and data sources**. These links define the control and data interfaces to the automation. They may also induce new requirements on data types/formats and APIs for external tools. An example in Change Management is the use of configuration data: if Change Management is automated but Configuration Management remains manual, the Configuration Management Database (CMDB) may need to be enhanced with additional APIs to allow for programmatic access from the automated Change Management activities. Moreover, data that is currently provided by humans to manual processes may need to be codified and standardized when those processes are automated.

The latter point motivates step 5 in our automation approach: **(5) Identify, design, and document induced processes needed to interface with or maintain the automation**. As described in [2], automation induces extra processes for tasks like data preparation, error recovery, and maintenance of automation. In particular, induced processes may be needed to handle the interfaces between automated activities and any activities, tools, or data sources designed for manual use. For example, if an automated software Change Management system requires that software be packaged in a special format, additional process is necessary to create the packages.

Step 5 is a key step since it surfaces some of the broader implications (and potential costs) of automation. Documenting and following those processes ensures that the automation will run smoothly over time. It also allows the induced process to be considered for its own automation treatment (following the same pattern we have laid out for the primary process). Finally, it encourages explicit consideration of the cost/benefit tradeoffs of automation [2].

Finally, the last step in our automation approach is to **(6) implement automation for the process flow and the delegated activities**. Implementing the process flow itself is best done using a workflow system to automatically coordinate the best-practice process's activities and the flow of information between them. Using a workflow system brings the additional advantage that it can easily integrate automated and manual activities within the same overall workflow.

For the delegated activities, additional automation implementation is needed. This is a non-trivial task that draws on the information gleaned in the earlier steps. It uses the best practice identified in (1) and the scoping in (2) to define the activities' functionality. The delegation choices in (3) scope the implementation work, while the interfaces and links to induced process identified in (4) and (5) define needed APIs, connections with external tools and data sources, and user interfaces. Finally, the actual work of the activity needs to be implemented directly, either in new code or by using existing tools.

In some cases, step 6 may also involve a recursive application of the entire methodology described here. This is typically the case when a delegated ITIL activity involves a complex flow of work that amounts to a process in its own right. In these cases, that activity sub-process may need to be decomposed into a set of subactivities; scoped as in steps (2) and (3), linked with external entities as in (4), and may induce extra sub-process as in (5). The sub-process may in turn also need to be implemented in a workflow engine, albeit at a lower level than the top-level best practice process flow.

One of the key advantages of our approach is that it can be recursively applied as described above. At each level, our approach produces a set of automated activities implemented in a workflow framework. These activities can potentially be reused in other process automation efforts, particularly if they define low-level automated functionality that applies across many different sub-processes. Over time, as an organization migrates to an automated best-practice process framework, a library of automation elements will be built up, providing valuable systems management assets and reducing the effort of further automation.

III. Case Study: Steps 1 and 2 – Scoping Change Management

We now turn to a detailed case study of applying the automation approach just described to Change Management. As mentioned earlier, we choose to focus on Change Management due to its importance in distributed systems management and because it offers some thorny automation challenges. This section considers the first part of the automation process: identifying a best practice and scoping the possible automation.

A. Step 1: Identifying Best Practice

Step 1 of our approach tells us to turn to ITIL for best practice guidance in implementing and automating Change Management. ITIL defines the goal of Change Management as "*to ensure that standardized methods and procedures are used for efficient and prompt handling of all changes, in order to minimize the impact of change-related incidents upon service quality […]*". Change Management is central to ensuring the availability, reliability, and quality of information technology (IT) services and processes. ITIL identifies the key major change management activities, the roles and stakeholders that perform them and input/output data that are either consumed or produced by the various activities in the change management process.

A simplified version of the ITIL change management process—along with its relationships to the configuration and release management processes—is depicted in Figure 1. It consists of a set of linked activities, each associated with a role that performs them and the shared data. The change management process starts with the submission of a Request for Change (RFC); many RFCs may be submitted concurrently. Other important inputs the change management process needs to consider are Service Level Agreements, Policies and best practices. Underlying the whole process is the Configuration Management Database (CMDB), a conceptual repository that holds the various change management artifacts and additional context information to provide, among other, a change audit trail. The CMDB also serves as the integration point of change management with other ITIL disciplines, in particular Configuration Management and Release Management.

From Figure 1, one can see that implementing a change management process is a major undertaking: Information stemming from various artifacts or stakeholders needs to be passed between the many different parties participating in the process in a coordinated way. The range of possible changes may be very broad, and the requirements of different customers with respect to the quality of service may differ greatly. It is therefore not surprising that the roll-out of complex changes in large-scale environments takes at least several days, and in many cases lasts several weeks or even months. IT organizations of enterprises and service providers are therefore eager to automate the change management process or, at least, its most costly and time-consuming activities, to accomplish labor cost reductions that are often in the double-digit million dollar range per year for large enterprises.

Automating the change management process, however, requires establishing a common context and a formal set of mutual agreements and expectations between the various parties

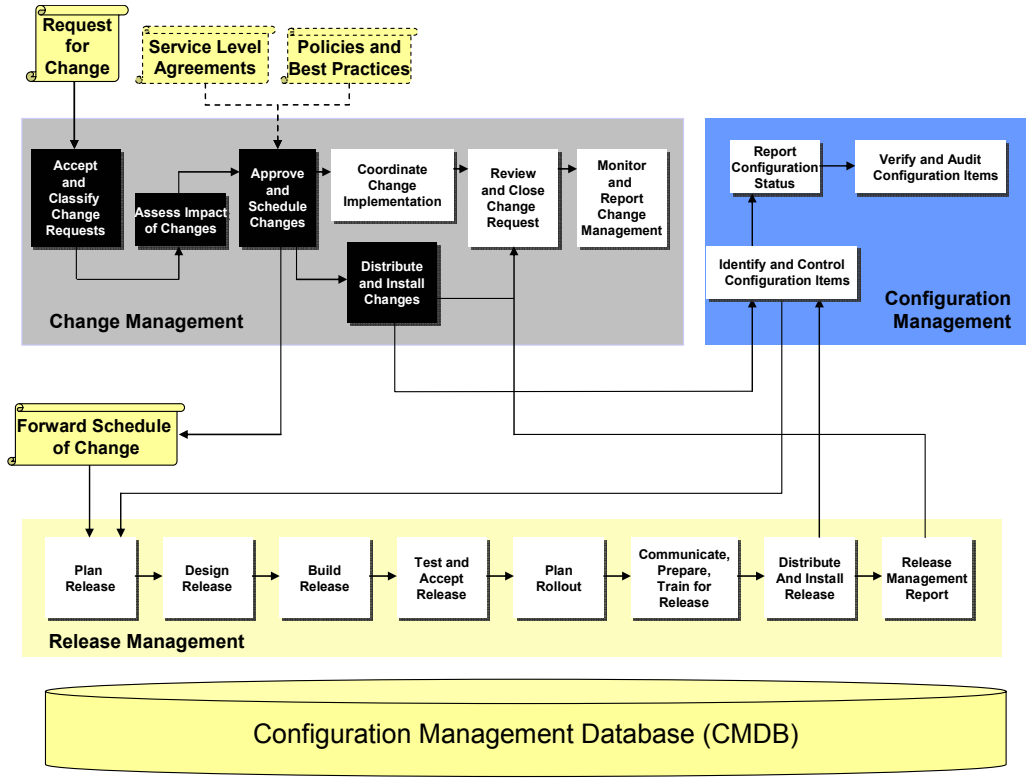

Figure 1. Activities, inputs and outputs of selected ITIL Service Support Processes

involved in the process, and the ITIL Service Support Processes play a key role in defining and exposing the necessary information to the various stakeholders. Following a standardized IT process model like ITIL yields the additional benefit of being able to exchange the systems that implement each of the building blocks of the IT service management process. This helps customers become less dependent on specific products and enables them to follow a "best-of-breed" approach by combining products from different vendors while maintaining interoperability.

B. *Step 2: Scoping the Automation*

As discussed in Section II, Change Management can cover many different kinds of changes in an IT environment. Each of these changes will follow the same overall process, but the details may be radically different—consider the difference between automating documentation updates and facilities repairs.

In our attempt to automate Change Management, we choose to focus on one small but crucially important slice of the overall problem: automating changes that affect the software lifecycle. In particular, we focus on requests for changes that involve deploying, upgrading, or decommissioning applications and their underpinning software stacks on distributed systems. In doing so, we significantly reduce the scope of our needed automation, both making it tractable to implement and making it more likely to be incrementally deployable into real IT service delivery environments.

IV. Step 3: Delegation for Software Lifecycle Change Management

With our task of automating Change Management scoped to software lifecycle changes, we can proceed with step 3 of our automation approach. Recall that this step involves identifying the key activities of Change Management to delegate to automation. Typically, the most expensive and time-consuming activities in Change Management are (1) assessing the impact of a Change, (2) obtaining the necessary approvals, (3) scheduling changes and (4) installing the changes. These activities (indicated in black in Figure 1) are the natural candidates for delegation to an *automated change management system*.

We next describe how such an automated change management system might operate, focusing in particular on how it handles the delegation of the activities mentioned above. Figure 2 depicts the interactions of such a system with the various stakeholders of the change management process in a data center that hosts web applications for customers. Every customer has signed one or more SLAs with the service provider that specify profit and loss functions for key performance indicators (response time, availability, throughput, etc.) as well as maintenance windows.

As specified in ITIL, the change management process starts with a customer submitting an RFC to the service provider's change management system. An illustrative example of such an RFC from the software lifecycle domain is a request to roll out two instances of the SPECjAppServer2004 enterprise application along with its underlying middleware in a two-tiered envi-

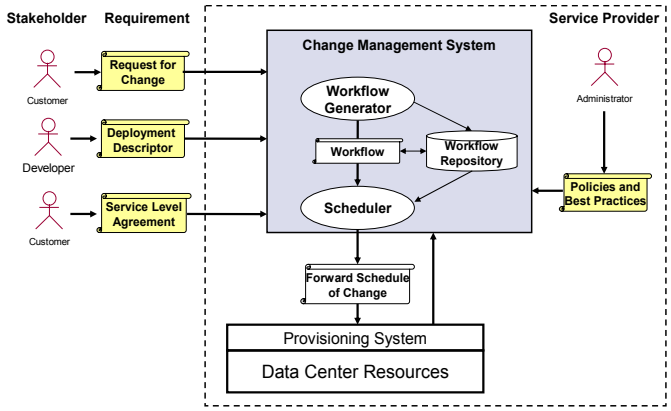

Figure 2. Stakeholders, building blocks and input data of a sample Change Management System

ronment consisting of a web application server and a database server.

The automated change management system executes the ITIL change management process (cf. Figure 1) starting with the first delegated activity: *assessing the impact of a change*, both with respect to its technical requirements and its financial impact. Examples for the former are software compatibility constraints that are expressed in deployment descriptors (see the discussion in the next section on IUDDs and SMDs), whereas the existing SLAs are used to assess the monetary impact. This latter step is needed in shared resource environments, such as on-demand data centers, since the roll-out of a change for one customer may impact the systems hosting the services of another customer.

The second delegated activity handled by the change management system is to *Approve and Schedule Changes*. We assume in this case that approval is delegated entirely to the automated system, with the understanding that in the software lifecycle domain the change management system is capable of making such decisions using its impact analysis results. The scheduling task is also delegated: the change management system must determine the best point in time to carry out a change and the best available resources on which the change will happen, according to the service provider's policies (e.g., minimize the downtime, maximize the profit, etc.) and best practices (e.g., a firewall must be placed between the web application server and database server tiers). The automated computation of the Forward Schedule of Change takes all the aforementioned parameters into account to find the best possible solution to this optimization problem.

To achieve the delegation of scheduling changes and computing the Forward Schedule of Changes, the Change Management System in our construction implements several major components, as shown in Figure 2. First, the **Workflow Generator** breaks down an incoming RFC into its elementary steps and determines the order in which they have to be carried out. The output is a workflow, expressed in a general-purpose workflow language, such as WS-BPEL [3], consisting of tasks and precedence constraints that link these tasks together. Consequently, workflows can be modified and aggregated by an administrator using general-purpose workflow editors. In addition, they can be stored in a **Workflow Repository** for subsequent reuse as they are not bound to specific target systems in a service provider's data center. If an incoming RFC relates to software artifacts for which a workflow already exists, the workflow is loaded from the workflow repository and directly submitted to the **Scheduler**. The Scheduler consumes one or more workflows and assigns tasks to available resources (obtained by querying the Provisioning System), according to the aforementioned constraints, such as Service Level Agreements (SLAs) and Policies.

Once the Forward Schedule of Change has been computed by the Scheduler, it is handed as input to the Provisioning System, which executes the requested changes on the data center resources. This accomplishes delegation of the *Distribute and Install Changes* step of the ITIL change management process. An important part of this process is the ability of the provisioning system to keep track of how well the roll-out of changes progresses on the targets, and to feed this status information back into the Scheduler.

V. STEPS 4 & 5: EXTERNAL INPUTS AND INDUCED PROCESS FOR AUTOMATED CHANGE MANAGEMENT

Now that we have identified the Change Management activities to be delegated and structured the architecture of an automated change management system, the next challenge is to implement step 4 of our approach. Namely, we need to identify the external information and data needed for the delegated activities to operate successfully. Figure 2 provides some guidance: the key data inputs are Requests for Change, Deployment Descriptors, and SLAs. All of these inputs need to be provided in machine-readable form or otherwise extracted from the IT environment for automated change management to function properly. In this paper, we will focus primarily on Deployment Descriptors as an illustrative example; Requests for Change in the software lifecycle domain are simply represented as a combination of software name plus lifecycle operation (e.g., install, configure, upgrade, uninstall) and there is already a significant body of existing work on machine-readable representations of SLAs.

A. *The Importance of Deployment Descriptors to Automated Change Management*

Deployment descriptors describe compatibility requirements between a variety of (software) products that can be composed into a *solution* that typically spans multiple target systems. They play a key role in guiding the automated change management system as it assesses the technical impact of changes and develops them into a feasible and efficient forward schedule of changes: the information they provide on compatibility between software products enables the change manager to establish an appropriate sequence of activities for requested lifecycle changes, and also provides the key dependency information needed to determine the impact of altering one or more pieces of software in a solution.

Deployment descriptors have some relationship to traditional software installers, such as *InstallShield* by Macrovision or Microsoft installer, but these basic installers are insufficient for multi-system, solution-oriented change management scenarios because (1) their usage is confined to single systems; consequently, installers are unable to coordinate the change

activities that happen on different systems; (2) installers address only a subset of the software lifecycle, namely the installation, upgrade and removal of individual software packages. In particular, no changes to the configuration of an already installed system can be made as the vast majority of configuration parameters are contained in response files; and (3) because single system installers only need to take the containment hierarchy of a software product into account, they represent installation topologies by means of tree-like data structures, whereas the topology of distributed systems typically is graph-like.

Thus to successfully automate software lifecycle change management, we need a more complete approach to deployment descriptors than traditional installers provide. To achieve this, we will use technology from *Solution Installation for Autonomic Computing* (Solution Install, for short).

Before diving into Solution Install, we will first introduce the SPECjAppServer2004 enterprise application, which we will use as the driving scenario to explain how the Solution Install technology can be applied to describe and process dependencies between components of a solution. Then, we illustrate how Solution Install checks can be used to describe additional, fine grained compatibility requirements that will guide the change management system in the process of assigning previously identified tasks to resources.

B. *Case Study: The SPECjAppServer2004 Enterprise Application*

Our case study is based on the scenario of installing and configuring a multi-machine deployment of a J2EE based enterprise application and its supporting middleware software (including IBM's HTTP Server, WebSphere Application Server, WebSphere MQ embedded messaging, DB2 UDB database and DB2 runtime client) on his behalf. The specific application we use is taken from the SPECjAppServer2004 enterprise application performance benchmark [9]. It is a complex, multi-tiered on-line e-Commerce application that emulates an automobile manufacturing company and its associated dealerships. SPECjAppServer2004 comprises typical manufacturing, supply chain and inventory applications that are implemented with web, EJB, messaging, and database tiers. We jointly refer to the SPECjAppServer2004 enterprise application, its data, and the underlying middleware as the SPECjAppServer2004 *solution*. Our example of a SPECjAppServer2004 solution spans a multi-system environment, consisting of two systems: one hosts the application server along with the SPECjAppServer2004 J2EE application, whereas the second system runs the database system that hosts the various types of SPECjAppServer2004 data (catalog, orders, pricing, user data, etc.). One of the many challenges in deploying such a distributed solution consists in determining the proper order in which its individual components need to be deployed, installed, started and configured. This, in turn, requires a detailed understanding on how the various components need to be 'wired together' and how their requirements and capabilities can be matched.

C. *Specifying Dependencies in Solution Install*

Solution Installation for Autonomic Computing (Solution Install) is a recent technology whose goal is to facilitate the provisioning and change management of multi-tiered application systems. Solutions comprise multiple levels of potentially nested Installable Units (IU). Each IU has an associated deployment descriptor (IUDD), an XML file that describes the content of an installable unit, its checks (required system resources, prerequisites), its dependencies and its (configuration) actions [4] [10].

There are three different types of IUs: Smallest Installable Units (SIU), Container Installable Units (CIU), or Solution Modules (SM). An SIU consists of a deployment descriptor and one software artifact; it is intended to be deployed to a single hosting environment. A hosting environment is a container in which artifacts are deployed and executed: typical examples of hosting environments for applications and middleware are operating systems, Web and EJB containers hosting J2EE applications, tablespaces of a DBMS hosting database tables etc. A CIU is an aggregated installable unit that is intended to be deployed to a single hosting environment. It consists of a deployment descriptor and one or more SIUs. A Solution Module is an installable unit that aggregates SIUs, CIUs, and other solution modules for deployment in one or more hosting environments. It is described by a deployment descriptor (SMD) that describes the aggregated installable units (SIUs, CIUs, and other solution modules) and the logical target environment for these aggregated installable units.

As an example, the overall SPECjAppServer2004 solution, comprising the J2EE application, user data, and the underlying middleware would be described by means of an SMD, which references five IUDDs for the following artifacts: the SPECjAppServer2004 J2EE application, the SPECjAppServer2004 data, WebSphere Application Server (WAS) 5.1, DB2 UDB 8.2 and the DB2 Runtime Client. Each IUDD contains CIUs that represent the logical units of deployment, which in turn aggregate SIUs that correspond to the physical deployment units. Each individual software product is represented by a separate IUDD, whereas the logical and physical components that belong to the same product are further described within the product's IUDD. In the example of WebSphere Application Server, the highest-level CIUs are the HTTP Server, the Web and EJB containers, and the messaging component, which are further broken down into the various fine-grained SIUs. As all of them make up the WAS product, their descriptions are contained in the WAS IUDD. The set of deployment descriptors for SPECjAppServer2004 and the underlying middleware stack reflects the structure depicted in Figure 3.

One can see that a set of Solution Install deployment descriptors represents a directed, acyclic graph. While dependencies between the CIUs and SIUs describe a software containment hierarchy, Solution Install provides additional IU types, such as the *referenced IU*. Whenever an IU depends on another IU belonging to a different product—and therefore specified in a different IUDD—the latter is referenced in the IUDD of the former. For example, the SPECjAppServer2004 data IU is defined in the IUDD of the J2EE application as a referenced IU.

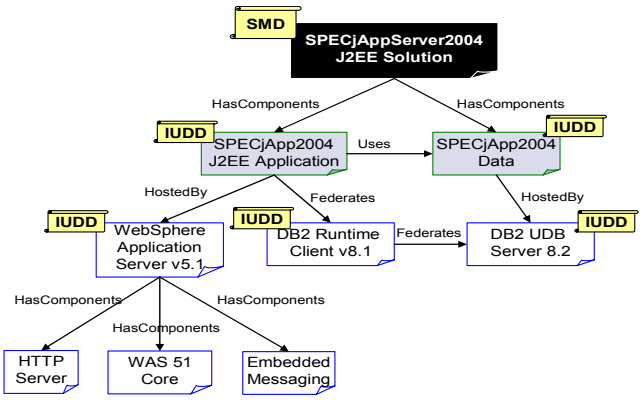

Figure 3. Modeling SPECjAppServer2004 and its underlying Middleware with Solution Install

SI classifies dependencies into various types (such as hosts, uses, contains, supersedes, etc.). Such dependencies, which apply to a multi-system scenario (and therefore don't exist in currently available single-system installers), provide a means of augmenting dependencies with meta-information that helps a change management system take the proper decisions. E.g., a 'supersedes' relationship indicates that a software artifact replaces the referenced software artifact, while a 'hosts' relationship indicates that a software artifact resides in the referenced hosting environment. The 'uses' and 'federates' relationship types are often applied in conjunction with referenced IUs. Consequently, each dependency type needs to be handled differently by a change management system.

In addition to specifying a common format for deployment descriptors, Solution Install defines a set of change management operations (create, update, delete, configure, verify, etc.) and provides a runtime environment. The runtime takes a deployment descriptor as input, decomposes aggregate units, and builds an installable unit graph.

D. Expressing Requirements in Solution Install

The previous section has shown how dependencies between various resources can be expressed in our SPECjAppServer2004 scenario. However, in order to achieve a workable solution, dependencies need to be refined further to address the compatibility requirements each software artifact has on other installable units and hosting environments. Solution Install provides this by defining various types of checks that are subsequently executed against the hosting environment. A check may either refer to various capacity parameters of the overall system such as CPU speed, consumption parameters such as RAM and disk space, properties of the hosting environment, or other software along with its possible version ranges and fixpack levels. Figure 4 depicts how Solution Install allows one to express that the DB2 Runtime Client 8.1 has the following requirements on other resources: The CPU clock speed must be at least 1.6 GHz, there have to be at least 130MB free space in the temporary file system, the operating system needs to be Windows 2000, whereas its version needs to be in a specific range. In addition, the DB2 runtime client requires the presence of DB2 UDB Server, whose version must be within the range of 7.2 and 8.1.

```
<SIU IUName="DB2UDB81_Client.binaries" hostingEnvType="OSRT:Operating_System">
  <identity>
    <name>com.ibm.db2.client.advanced_binaries</name>
    <UUID>68c28f57092a4593a45a6a666a9feb5a</UUID>
    <version>8.1</version>
  </identity>
  <checks>
    <capacity checkId="ProcessorSpeed_Check" type="minimum">
      <propertyName>Processor/CurrentClockSpeed</propertyName>
      <value>1600</value>
    </capacity>
    <consumption checkId="Temporary_Disk_Space_Check" temporary="true">
      <propertyName>FileSystem/AvailableSpace</propertyName>
      <value>130</value>
    </consumption>
    <property checkId="Windows_2000_Check">
      <propertyName>OsType</propertyName>
      <value>Windows 2000</value>
    </property>
    <version checkId="Windows2000_version_Check">
      <propertyName>version</propertyName>
      <minVersion>5.00.2195.2</minVersion>
      <maxVersion>5.00.2195.4</maxVersion>
    </version>
    <software checkId="DB2_for_Windows_Check">
      <UUID>12345678901234567890123456789012</UUID>
      <name pattern="true">(DB2|Universal Database)</name>
      <minVersion>7.2</minVersion>
      <maxVersion>8.1</maxVersion>
    </software>
  </checks>
```

Figure 4. Expressing requirements of the DB2 Runtime Client 8.1 on other artifacts

Sometimes, there are several alternative configurations that meet the requirements, which broadens the options of a change management system to fulfill a Request for Change. For example, a web application server can specify that it is able to interact with any database system implementation (DB2, Cloudscape, etc.) as long as the latter can be accessed via JDBC. Several alternative database systems can be expressed in an IUDD as well as requirements on the state in which the database system must be so that the operation succeeds: Figure 5 provides an example of a requirements definition that provides alternatives for prerequisite software along with their mandatory state. In order to configure the JDBC data source of an web application server, the database system referenced by the data source definition must not only be installed, but also running.

Despite the fact that the example depicted in Figure 4 is fairly simple and leverages only a few of the checks that we have defined for each of the components of our SPECjAppServer2004 example (for the sake of space), one can see that Solution Install allows the definition of very fine-grained requirements for a wide variety of parameters. The Solution Install runtime environment automatically performs this requirements checking and is therefore able to match, e.g., the version definition of one artifact with the version requirements of another.

```
<requirement name="ConfigureJDBC_connectors" operations="Configure">
  <alternative name="isDB2_Installed" priority="1">
    <checkItem checkIdRef="DB2_InstalledCheck"/>
    <checkItem checkIdRef="DB2_StartedCheck" testValue="started"/>
  </alternative>
  <alternative name="isCloudscape_Installed" priority="2">
    <checkItem checkIdRef="Cloudscape_InstalledCheck"/>
    <checkItem checkIdRef="Cloudscape_StartedCheck" testValue="started"/>
  </alternative>
  <alternative name="isDBZ_Installed" priority="1">
    <checkItem checkIdRef="DBZ_Check"/>
  </alternative>
</requirement>
```

Figure 5. Compatiblity and Lifecycle requirements in Solution Install

E. Induced Process Implications

From our discussion, it is clear that the creation of IUDDs for large software products may require a significant effort to ensure all the necessary information is present and accurate. The processes necessary to create these IUDDs, and the SMD ultimately submitted to the change management system, represent induced process that results from our delegation of the core change management activities to automation. Thus we have arrived at step 5 in our methodology, where we must consider the impact of these additional induced activities and understand whether they decrease the value of our automation.

In the case of generating IUDDs, it is likely that the required process can itself be relatively easily automated. In analogy to J2EE deployment descriptors, the fairly complex and detailed IUDDs for individual products can be automatically generated during the product's build process and bundled with the product install image. IUDDs are also defined once at build time and can be reused over and over again in processing multiple change requests, amortizing any extra development cost.

Thus only the SMD—the topmost deployment descriptor containing references from the overall solution to the products it is made of—is defined manually by an administrator at solution design time. This is extra induced process, but its impact is mitigated by use of a graphical editor, such as *SolutionArchitect* by ZeroG/Macrovision, to simplify SMD construction. Also, as with IUDDs, SMDs can be defined once and reused over and over again, amortizing their creation cost. However, while the extra effort of creating a single, fairly simplistic SMD for provisioning a multi-tiered solution seems negligible—especially in large data center environments with significant reuse—there still may be cases where the extra effort does not pay off, as discussed in [2].

VI. Step 6: Implementation: CHAMPS – A Schedule-optimizing Automated Change Manager

Step 6 of our automation approach is to implement the automation for the delegated activities. In Section III we began outlining a high-level architecture for an automated change management system—starting with Workflow Generator, Workflow Repository, and Scheduler components. Figure 6 fleshes that architecture out in terms of an implementation developed at IBM Research. Our automated change management system, CHAMPS (CHAnge Management with Planning and Scheduling), automates change impact assessment, approval, and scheduling, and is able to generate a Forward Schedule of Change with a very high degree of parallelism for a set of change management tasks by exploiting detailed factual knowledge about the structure of a distributed system from dependency information, provided by Solution Install deployment descriptors. Its optimization techniques are based on mathematical scheduling theory. Once the forward schedule of changes is generated, CHAMPS interfaces closely with a multi-layer provisioning system to automate the implementation of that forward schedule of changes. For the detailed architecture of CHAMPS and the mathematical formulation of the change-scheduling optimization problem, the reader is referred to [6].

Figure 6 illustrates as well the pattern laid out in Section II, where workflow technology is used to coordinate overall automation of a best-practice process such as Change Management. The left-hand side of the figure shows a workflow engine being used at the top level to coordinate the flow of work between manual and automated activities: a Request for Change kicks off the ITIL-based Change Management workflow, which coordinates manual tasks such as change classification. If the change is classified as a software lifecycle change, the workflow engine can directly invoke CHAMPS to automatically handle the next several change management activities (accomplishing the desired delegation), returning control back to the workflow for the final manual step in change management (reviewing and accepting the change). For Requests for Change in other (non-software-lifecycle-related) domains, the workflow engine will coordinate the manual execution of the change management process without using CHAMPS.

Note as well that Figure 6 illustrates the use of the Configuration Management Database (CMDB) to persist all the relevant artifacts involved in the automated change management process, including the Request for Change, the Forward Schedule of Changes, and any relevant SMDs and IUDDs. This follows the ITIL best practice for change management, where ITIL Configuration Management and the CMDB are used to store and persist process-related artifacts.

A. The CHAMPS Change Manager: Generating the Forward Schedule of Changes

The CHAMPS change manager incorporates the Workflow Generator, Workflow Repository, and Scheduler components mentioned earlier in Section III. CHAMPS relies on Solution Install to parse a set of submitted deployment descriptors in order to build an in-memory model of the various artifacts and their dependencies. This dependency model is the input data to the CHAMPS Workflow Generator component, which uses the dependency model to assess the technical impact of the software change request and thereby to structure an initial *change workflow* that can effectively implement the desired change. To do so, it carries out the necessary requirement checks, evaluates the dependencies, and derives the proper order in which change management operations need to be carried out. Based on the dependencies (or lack thereof), the Workflow Generator is able to determine the order in which change implementation activities need to be carried out, and whether activities may happen concurrently. The change manager uses the Change Plan Repository to store generated change workflows for possible future reuse; if an appropriate change workflow is found in the repository for a given change request, that plan can be used directly, bypassing the workflow generator.

The CHAMPS Scheduler component takes advantage of the requirements expressed in Solution Install when trying to assign tasks identified by the Workflow Generator to suitable resources. For an identified candidate resource, the check definitions within each deployment descriptor are submitted to the Solution Install runtime to carry them out. If a check fails, the Scheduler picks a different, more suitable resource. Once the scheduler completes its work, it has transformed the initial

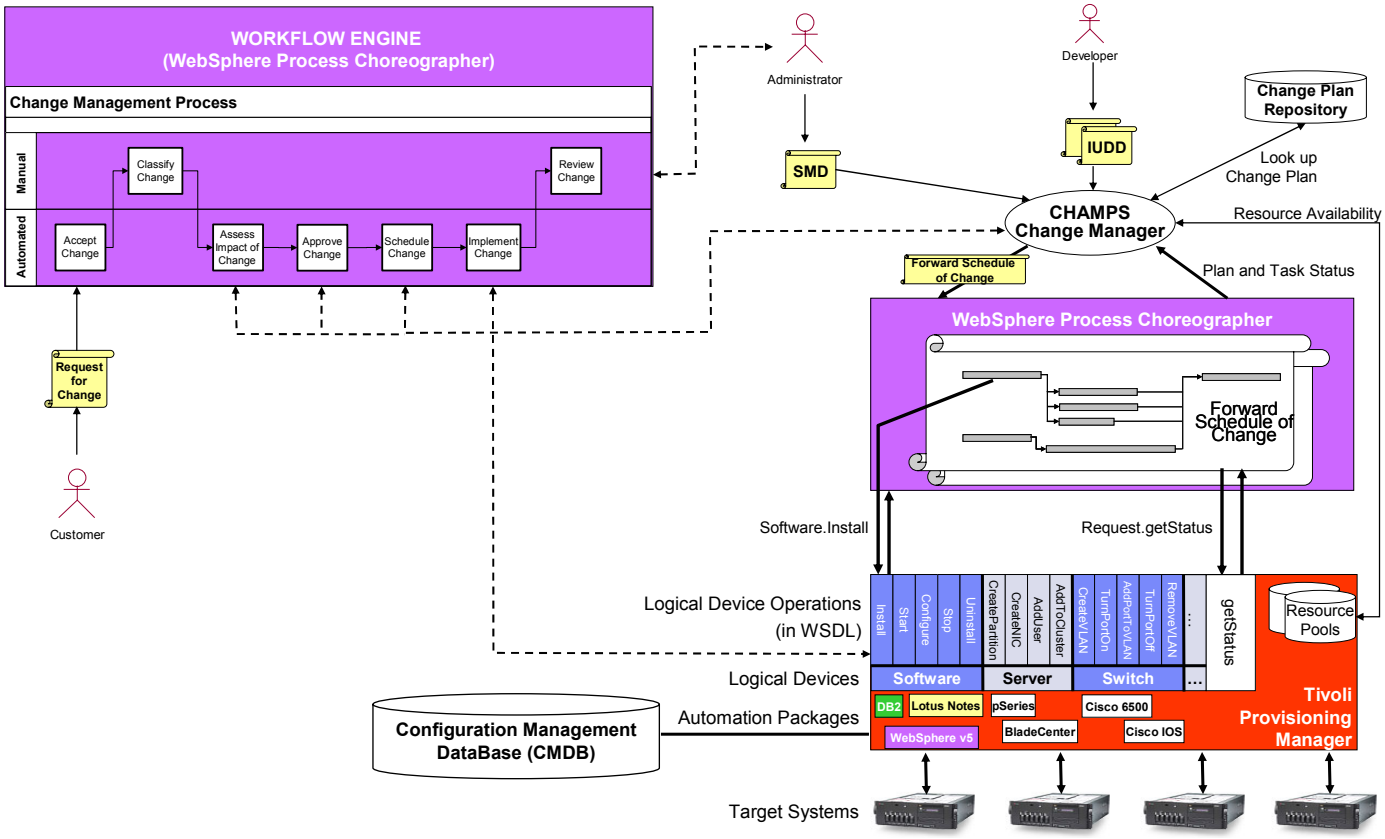

Figure 6. Architecture of a workflow-driven change manager prototype

change workflow into a Forward Schedule of Changes, which we refer to as the *change plan* in our implementation.

B. *Implementing the Change Plan*

Recalling the architecture depicted in Figure 2, the change plan is next submitted to a provisioning system for implementation of the delegated *Distribute and Install Changes* ITIL process activity. The provisioning system coordinates the change management activities documented in the change plan across the hosting environments that comprise the target IT environment affected by the change. In our prototype system, we chose to treat the change plan as a workflow, represented in WS-BPEL, the Business Process Execution Language, and we use a general-purpose workflow engine—the IBM WebSphere Process Choreographer—to interpret and execute the change plan.

The Process Choreographer in turn invokes a lower-level provisioning system, IBM Tivoli Provisioning Manager, which in turn maps the actions defined in the change plan to operations that are understood by the target systems. As visible in Figure 6, the Provisioning Manager's object-oriented data model is a hierarchy of logical devices that correspond to the various types of managed resources (e.g., software, storage, servers, clusters, routers or switches) present in the target IT environment. The methods of these logical device types correspond to Logical Device Operations (LDOs) that are exposed as WSDL interfaces, which allows their inclusion in the change plan as BPEL partnerLinks. The IBM Tivoli Provisioning Manager uses so-called "automation packages", which are product-specific implementations of logical devices: e.g., an automation package for the DB2 DBMS would provide scripts that implement the software.install, software.start, software.stop, etc. LDOs. An automation package consists of a set of Jython scripts, each of which implements an LDO. Every script can further embed a combination of PERL, Expect and bash shell scripts that are executed on the remote target systems.

The workflow engine inputs the change plan and starts each provisioning operation by directly invoking the LDOs of the provisioning system. These invocations are performed either in parallel or sequentially, according to the flows, sequences and links defined in a change plan. A major advantage of using a workflow engine for our purposes is the fact that it automatically performs state-checking, i.e., it determines whether all conditions are met to move from one activity in a workflow to the next. Consequently, there is no need for us to develop additional program logic to perform such checks. Additionally, the provisioning system reports the status of each operation execution back to the workflow engine. This status information is used by the workflow engine to check if the workflow constraints defined in the plan (such as deadlines) are met and to inform the change manager whether the roll-out of changes runs according to the schedule defined in the change plan. Note that when an error occurs during the change management operations or when the deployment is canceled, the runtime environment ensures that the various operations are rolled back across the involved hosting environments.

C. Discussion: Recursive Application of the Automation Pattern

Our automation implementation essentially makes use of three levels of workflow engine: once for the top-level coordination of the Change Management workflow, again for implementation of the generated change plan, and finally a third time inside the provisioning manager where the LDOs implement miniature-workflows within their defined scripts. This use of multiple levels of workflow engine illustrates a particular pattern of composition that we expect to be common in automation of best-practice IT service management processes, and recalls the discussion earlier in Section II of recursive application of the automation approach.

In particular, the delegated Change Management activity of Distribute and Install Changes involves a complex flow of work in its own right—documented in the change installation plan produced by CHAMPS. We can see that our approach to automating this change plan follows the same pattern we used to automate change management itself, albeit at a lower level. For example, the creation of the change workflow is a lower-level analogue to using ITIL best practices to identify the Change Management process activities. The execution of change plan tasks by the provisioning system represents delegation of those tasks to that provisioning system. The provisioning system uses external interfaces and structured inputs and APIs to automate those tasks—drawing on information from the CMDB to determine available resources and invoking lower-level operations (automation packages) to effect changes in the actual IT environment. In this latter step, we again see the need to provide such resource information and control APIs in the structured, machine-readable formats needed to enable automation. The entire pattern repeats again at a lower level within the provisioning system itself, where the automation packages for detailed software and hardware products represent best practice operations with delegated functionality and external interfaces for data and control.

One of the key benefits of this type of recursive composition of our automation approach is that it generates *reusable automation assets*. Namely, at each level of automation, a set of automated delegated activities is created: automated ITIL activities at the top (such as Assess Change), automated change management activities in the middle (such as Install the DB2 Database), and automated software lifecycle activities at the bottom (such as Start DB2 Control Process). While created in the context of change management, it is possible that many of these activities (particularly lower-level ones) could be reused in other automation contexts. For example, many of the same lower-level activities created here could be used for performance management in an on-demand environment to enable creating, activating, and deactivating additional database or middleware instances. It is our hope that application of this automation pattern at multiple levels will reduce the long-term costs of creating system management automation, as repeated application will build up a library of reusable automation components that can be composed together to simplify future automation efforts.

VII. Evaluation

To evaluate the efficiency and effectiveness of our automation approach with respect to scheduling and rolling out changes, we measured the time it would take to manually provision SPECjAppServer2004 (excluding human think time) and compared it to a CHAMPS-driven version of the SPECjAppServer2004 scenario. In the CHAMPS scenario, a user first fills out an RFC and submits an associated deployment descriptor, supplies the values for the requested parameters, and kicks off the automated process by submitting the RFC.

In the first case, we assume that an administrator—not being aware of the dependencies between the various parts of the overall SPECjAppServer2004 solution—would carry out each individual step in the provisioning process only after the prior step has completed successfully, e.g., first, the database server would be set up, and only once this is done, an administrator would initiate the deployment, installation and configuration of the application server. The total time for such a strictly sequential flow is 50 minutes and 25 seconds, which is an optimistic number, given the fact that we do not take the think times of the administrator into account.

In the second case, CHAMPS generates the provisioning workflow by taking the (non-)existence of dependencies between the artifacts of the solution into account, which are expressed by deployment descriptors. The possibility of carrying out multiple activities in parallel, which CHAMPS automatically identifies, allows it to significantly reduce the total time for provisioning the SPECjAppServer2004 application and its middleware stack because the deployment and installation of the large middleware packages (the size of the install image for DB2 UDB v8.1.6 is 535 MB, the size of WebSphere Application Server v5.1 is 415 MB) are by far the most time-consuming activities. Using two 2.4 GHz Intel Pentium machines, we were able to collect the times depicted in Figure 7 for the various activities of the workflow. The machines were connected through 100MBit/s Ethernet to a 2-way SMP Intel 2.4 GHz Pentium system running CHAMPS that also hosted the install images of the software.

As depicted in Figure 7, the realized time savings (a reduction of more than 30% overall provisioning time on average) are quite significant as the times for the deployment and installation activities are fairly high, compared to the durations of the start and configuration activities. However, the former activities can be carried out concurrently as they happen on two different systems. The total provisioning time for the execution of the overall workflow is 34 minutes on average over multiple runs, thus realizing a speedup of 1.5.

In addition to evaluating the performance improvements from automated software change management, we also in previous work examined the implications of automation on administrative complexity—complexity that contributes to human administrative burden, required human skill, and the possibility of erroneous configurations. Our results showed a significant complexity reduction from CHAMPS-based automation of installing and configuring SPECjAppServer2004; a detailed evaluation is provided in [1].

Activity type	Average times [minutes.seconds] for:		
	DB2 UDB 8.1 / RT Client	WAS 5.1	SPECjAppServer
Deployment (file transfer and unpacking of install image)	5.56 / 1.14	5.52	1.34
Installation	6.42 / 1.34	14.31	4.54
Start	0.16	0.48	n/a
Configuration (incl. DB creation and loading of data)	5.28	1.36	n/a
Total time (strictly sequential, no think time)	50 min 25 sec.		
Total time (CHAMPS workflow)	34 min 00 sec.		
Total savings (time seq.–time CHAMPS)/time seq.	**16 min 25 sec (~33%)**		

Figure 7. Times for CHAMPS-based SPECjAppServer2004 provisioning based on multiple runs

VIII. Related Work

While there is a great deal of existing work describing ad-hoc automation of aspects of service management, there is very little work that (like this paper) describes a general automation approach starting from best practices such as ITIL. One relevant piece of work in the latter category is eTOM, the enhanced Telecom Operations Map, which provides a top-down hierarchical view of business processes. eTOM does not itself address how processes are supported by human or automated actions, although work is underway as part of the broader Next Generation Operations Support System (NGOSS) program of TM Forum to address this. While the focus of ITIL is on the operational side, it is fair to say that eTOM is more geared towards product/service aspects. One of the notable differences is that eTOM does not have the concept of a Configuration Management Database (CMDB); instead, eTOM assigns configuration items to categories that reflect managed resource types. Nevertheless, both approaches overlap in scope. For a detailed discussion of eTOM and ITIL and a mapping between the two approaches, the reader is referred to [11].

Another relevant piece of work is the Quartermaster system from HP, which provides tools for design, deployment, and operation of utility-computing applications [8]. Quartermaster in part provides a framework for automation of service delivery in a utility/on-demand environment, although it focuses more on composition, resource allocation/scheduling, and system modeling than the operational aspects of service management covered by ITIL best practices. Quartermaster offers some change management capabilities and maintains an equivalent of the CMDB in its CIM-based system models; it offers scheduling and resource allocation capabilities along the lines of those in CHAMPS. But, unlike the automation components developed for CHAMPS, Quartermaster is not built around standard best-practice activities for IT service management such as those defined by ITIL. Thus while Quartermaster provides a rich framework for utility computing, it does not directly address the problem that we tackle here, namely automating IT service delivery in the context of a best-practice process framework.

IX. Summary and Conclusions

With labor costs increasingly dominating the economics of IT service delivery, automating IT service management has never been more important. But automation is a tricky business: without the right organizational framework and standard process behind it, automation can "bake in" suboptimal processes and furthermore can be difficult to reuse and apply broadly. We have attempted to address those concerns in this paper by introducing a best-practice approach to automating IT service management, starting with ITIL best-practice processes and proceeding through a six-step approach for refining those ITIL processes to practical automated service management implementations.

We demonstrated this approach through a case study of automating IT Change Management in the software lifecycle domain; IT Change Management has become one of the most labor-intensive and time-consuming activities in Service Management and thus automating it is a top priority for many IT service delivery environments. Our approach brings to automated change management an organizational framework that clearly defines standardized processes for carrying out changes, a standardized method of providing needed information on technical constraints such as software compatibility requirements by means of formal deployment descriptors, and an implementation architecture based on multi-level workflows produced through recursive application of our automation pattern.

The system that resulted from our automation efforts is CHAMPS, a schedule-optimizing Change Manager. CHAMPS takes an optimization-centric approach to developing and scheduling the ITIL Forward Schedule of Changes process artifact, and is able to generate change plans with a very high degree of parallelism for a set of change management tasks by exploiting detailed factual knowledge about the structure of a distributed system from dependency information at runtime. Our empirical results suggest that exploiting parallelism can lead to substantial time savings, such as about 33% reduction in the installation time of a complex Java based enterprise application.

While these results are encouraging in showing the value of process automation and the effectiveness of our approach, they are a starting point. Significant work is needed to apply our methodology to automate the many more remaining processes and steps in the ITIL Service Support best practices. Furthermore, within the change management domain itself, more work is needed to extend the automation we describe here, and the Solution Install framework on which it is built, to other entities of the IT infrastructure beyond software applications, including hardware components, storage systems, operating systems, and networking components.

REFERENCES

[1] A. Brown, et al. A Model of Configuration Complexity and its Application to a Change Management System. In *Proceedings of the 9th IEEE/IFIP Network Operations and Management Symposium* (IM'2005), pages 631 – 644, Nice, France, May 2005. IEEE Publishing.

[2] A. Brown and J. Hellerstein. Reducing the Cost of IT Operations--Is Automation Always the Answer? In *Proceedings of the 10th Workshop on Hot Topics in Operating Systems (HotOS 2005)*, Santa Fe, NM, June 2005.

[3] Business Process Execution Language for Web Services (WS-BPEL) Version 1.1. Second Public Draft Release, BEA Systems, International Business Machines Corp., Microsoft Corp., SAP AG, Siebel Systems, May 2003. http://www6.software.ibm.com/developer/software/library/ws-bpel.pdf.

[4] H. Chu (Editor), Installable Unit Package Format Specification, Version 1.0., W3C Member Submission, IBM Corp., ZeroG Software, InstallShield Corp., Novell, July 2004, http://www.w3.org/Submission/2004/SUBM-InstallableUnit-PF-20040712

[5] IT Infrastructure Library. ITIL Service Support, version 2.3, Office of Government Commerce, June 2000.

[6] A. Keller et al. The CHAMPS System: Change Management with Planning and Scheduling. In *Proceedings of the 9th IEEE/IFIP Network Operations and Management Symposium (NOMS'2004),* pages 395 – 408, Seoul, Korea, April 2004. IEEE Publishing.

[7] D. Oppenheimer, A. Ganapathi, and D.A. Patterson. Why do internet services fail, and what can be done about it? In Proceedings of the 4th Usenix Symposium on Internet Technologies and Systems, Seattle,WA, USA, March 2003. USENIX Association.

[8] S. Singhal, et al.: Quartermaster – a Resource Utility System. In *Proceedings of the 9th IEEE/IFIP Network Operations and Management Symposium* (IM'2005), pages 265 – 278, Nice, France, May 2005. IEEE Publishing.

[9] SPECjAppServer2004 Design Document, Version 1.01, January 2005, http://www.spec bench.org/osg/jAppServer2004/docs/DesignDocument.html

[10] M. Vitaletti (Editor), Installable Unit Deployment Descriptor Specification, Version 1.0., W3C Member Submission, IBM Corp., ZeroG Software, InstallShield Corp., Novell, July 2004, http://www.w3.org/Submission/2004/SUBM-InstallableUnit-DD-20040712/

[11] An Interim View of an Interpreter's Guide for eTOM and ITIL Practitioners, *enhanced Telecom Operations Map (eTOM)*, Document GB921V, version 1.1, release 4.6, TeleManagement Forum, February 2005