Proceedings of the *15th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 2004)*, Davis, CA, USA, November 2004

Automating the Provisioning of Application Services with the BPEL4WS Workflow Language

Alexander Keller[1] and Remi Badonnel[2]*

[1] IBM T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598, USA
alexk@us.ibm.com
[2] LORIA-INRIA Lorraine
615, rue du Jardin Botanique - B.P. 101, 54600 Villers Les Nancy Cedex, France
remi.badonnel@loria.fr

Abstract. We describe the architecture and implementation of a novel workflow-driven provisioning system for application services, such as multi-tiered e-Commerce systems. These services need to be dynamically provisioned to accomodate rapid changes in the workload patterns. This, in turn, requires a highly automated service provisioning process, for which we were able to leverage a general-purpose workflow language and its execution engine. We have successfully integrated a workflow-based change management system with a commercial service provisioning system that allows the execution of automatically generated change plans as well as the monitoring of their execution.

1 Introduction and Problem Statement

The extremely high rate of change in emerging service provider environments based on Grid and Web Services technologies requires an increasingly automated service provisioning process. By provisioning, we mean the process of deploying, installing and configuring application services. A promising, systematic approach to this problem is based upon the adoption of Change Management [5]. An important prerequisite for automated Change Management is the ability of a service provisioning system to interpret and execute change plans (described in a general-purpose workflow language) that have been generated by a Change Management System. This requires adding new workflows "on-the-fly" to provisioning systems, i.e., without writing new program code and without human intervention. Second, the workflows should contain temporal constraints, which specify deadlines or maximum allowable durations for each of the activities within a workflow. Finally, once the workflows are executed by a provisioning system, the system should be able to check their status to determine if an activity has completed and, if yes, whether it was successful or not.

This paper describes our approach to addressing these requirements and its implementation. It enables a provisioning system to understand and execute

* Work done while the author was an intern at the IBM T.J. Watson Research Center

change plans specified in the *Business Process Execution Language for Web Services (BPEL4WS)* [1], an open workflow language standard, as a means to apply change management concepts and to automate provisioning tasks significantly. In addition, our system is capable of providing feedback from the provisioning system back to the change manager, so that the latter can monitor how well the execution of the change plan proceeds, and perform adjustments if needed.

The paper is structured as follows: Section 2 gives an overview of typical service provisioning systems, such as *IBM Tivoli Intelligent Orchestrator (TIO)*, and describes related work. Our approach for integrating CHAMPS, a Change Manager developed at IBM Research, with TIO and a workflow engine capable of understanding BPEL4WS, is discussed in section 3; we present the proof-of-concept implementation in section 4. Section 5 concludes the paper and presents the lessons we learned during this work as well as issues for further research.

2 Towards Automated Service Provisioning

The importance of automating the provisioning of services is underscored by a recent study [9] showing that operator errors account for the largest fraction of failures of Internet services and hence properly managing changes is critical to availability. Today, however, service provisioning systems are isolated from the change management process: They typically come with their own, proprietary workflow/scripting language, thus making it hard for a change manager to formulate reusable change plans that can be understood by different provisioning systems. Our goal is to tie provisiong systems into the change management process. By leveraging the Web Services technology and a standardized, general-purpose workflow language for expressing change plans and demonstrating the feasibility of integrating a common-off-the-shelf workflow engine with a commercial provisioning system, our approach is applicable to a wide range of provisioning scenarios.

2.1 Provisioning Systems: State of the Art

Typical provisioning systems, such as *Tivoli Intelligent Orchestrator (TIO)* [4] provide an administrator with a runtime environment for defining and subsequently executing provisioning scripts. Figure 1 depicts the sequence of steps for provisioning a web site that uses the *IBM HTTP Server (IHS)*, a variation of the Apache Web Server. In this example, 10 actions need to be carried out by the provisioning system, which can be summarized as follows: Copying the install image of the HTTP server into a temporary directory on a target system, launching the installation, updating the httpd.conf configuration file, installing the web site content (HTML pages, pictures etc.), starting the HTTP server, and performing cleanup tasks once the installation has been completed successfully. In TIO, such a provisioning workflow consists of a sequence of operations; these are pre-defined activities that can be adapted and customized by an administrator, as well as aggregated into new workflows. For every operation, an

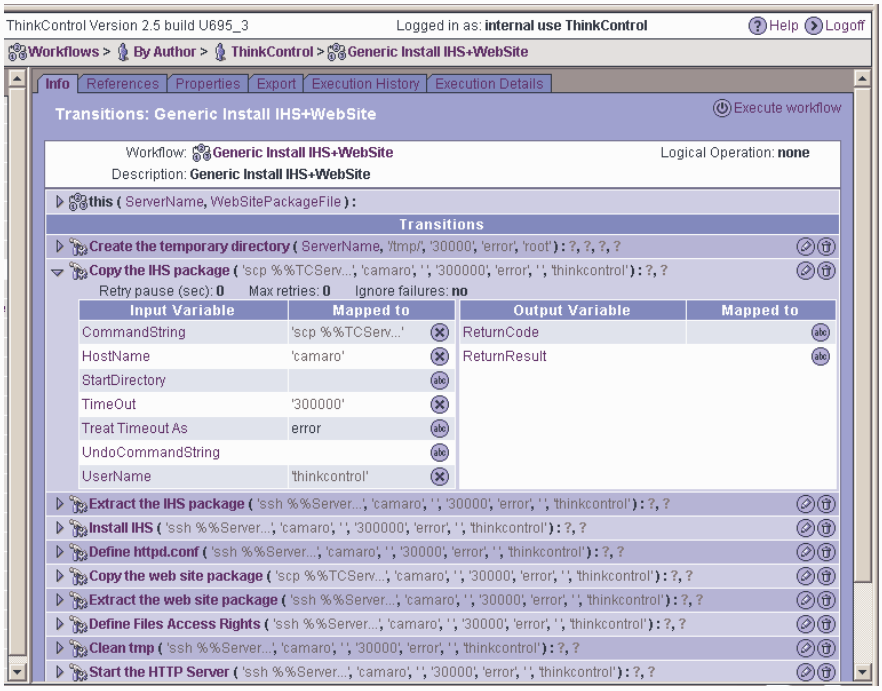

Fig. 1. Steps for Provisioning an HTTP Server from a Service Provisioning System

administrator can specify what steps need to be taken if the operation fails, such as undoing the installation or notifying the administrator. Provisioning systems that require such fine-grained definitions of provisioning workflows expect an administrator to have a detailed understanding of the steps involved in setting up the provisioning of complex, multi-tiered systems. However, the lack of knowledge about the structure of a distributed system and the dependencies between its fine-grained components often tend to make an administrator overly prudent when designing workflows, e.g., by not exploiting the potential for concurrent execution of provisioning workflows, thus resulting in inefficiencies.

Another example of a commercial service provisiong system is given in [3]. It describes a workflow-based service provisiong system for an Ethernet-to-the-Home/Business (ETTx) environment, consisting of a policy engine, a service builder, an activation engine and a workflow engine. The (proprietary) workflow engine orchestrates the execution flow of the business process, whereas the actual provisioning steps are executed by a custom-built activation engine. Our approach, in contrast, lets a common-off-the-shelf workflow engine orchestrate the actual provisioning process. Indeed, there has been interest in using workflow technologies to coordinate large scale efforts such as change management [7], and to automate the construction of a Change Plan [8]. However, no current

provisioning system is able to understand change plans that leverage the full potential of typical general-purpose workflow languages, such as the concurrent execution of tasks and the evaluation of transition conditions to determine if the next task in a workflow can be started.

2.2 Related Work

In addition to the products described above, service provisioning and change management have received considerable attention in both academia and industry. A constraint satisfaction-based approach to dynamic service creation and resource provisioning in data centers is described in [10]. Whenever a policy manager finds a match between an incoming request and a set of resource type definitions, the task-to-resource assignment is treated as a constraint satisfaction problem, which takes the service classes as well as the technical capabilities of the managed resources into account, but does not perform additional optimization. The output is consumed by a deployment system.

STRIDER [12] is a change and configuration management system targeted at detecting and fixing errors in shared persistent configuration stores (such as the Windows Registry). To do so, it follows an elaborate three-step process to analyse the state of configuration parameters, finds similar, valid configurations and subsequently narrows down the range of results to the most likely configuration. Since it deals with (re)setting configuration parameters and does not perform software deployment, the system does not make assumptions about the order in which provisioning steps need to be carried out.

Finally, the Workflakes system, described in [11], provides workflow-driven orchestration of adaptation and reconfiguration tasks for a variety of managed resources. Workflakes focuses on an adaptation controller for systems and services, where workflows describe the dynamic adaptation loop. Our work supports a change management approach, where dynamically generated workflows (describing change plans) are executed by a provisioning system.

3 Integrating Change Management and Provisioning

3.1 The CHAMPS Change Manager

The CHAMPS system is a Change Manager for **CHA**nge **M**anagement with **P**lanning and **S**cheduling [6]. CHAMPS consists of two major components: The **Task Graph Builder** breaks down an incoming request for change into its elementary steps and determines the order in which they have to be carried out. This **Task Graph** is a workflow, expressed in BPEL4WS, consisting of tasks and precedence constraints that link these tasks together.

In a second step, multiple task graphs (representing the various requests for change that are serviced by the change manager at a given point in time) are consumed by the **Planner & Scheduler**. Its purpose is to assign tasks to available resources, according to additional monetary and technical constraints, such as

Service Level Agreements (SLAs) and Policies. To do so, it computes (according to various administrator-defined criteria) a **Change Plan** that includes deadlines and maximizes the degree of parallelism for tasks according to precedence and location constraints expressed in the Task Graphs. Again, the BPEL4WS workflow language is used to express the Change Plan. Figures 5, 7 and 8 in section 4 contain various examples of instructions specified in a Change Plan.

3.2 Integration Architecture

Once the Change Plan has been computed by the Planner & Scheduler, it is input to the **Provisioning System**, which retrieves the required software packages from a Package Repository, and rolls out the requested changes to the targets in the order specified in the plan. 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 Planner & Scheduler. Being aware of the current status of the provisioning process enables the Planner & Scheduler to track the actual progress against the plan and perform on-line plan adjustment (by re-computing the change plan) in case the process runs behind schedule. In addition, such a feedback mechanism can be used to gain an understanding on how long it takes to complete a task.

Our architecture, depicted in figure 2, aims at integrating the provisioning system with CHAMPS to execute the change plans in a data center environment comprising resources such as server pools, servers, software products, switches, and firewalls. In section 2.1, we noted that current provisioning systems do not execute workflows in parallel and often do not take temporal and location constraints into account. The deployment engine of the provisioning system allows us to perform a variety of management operations on managed resources. While these operations are grouped into a single sequence on the graphical user interface (cf. figure 1), a WSDL interface exists that

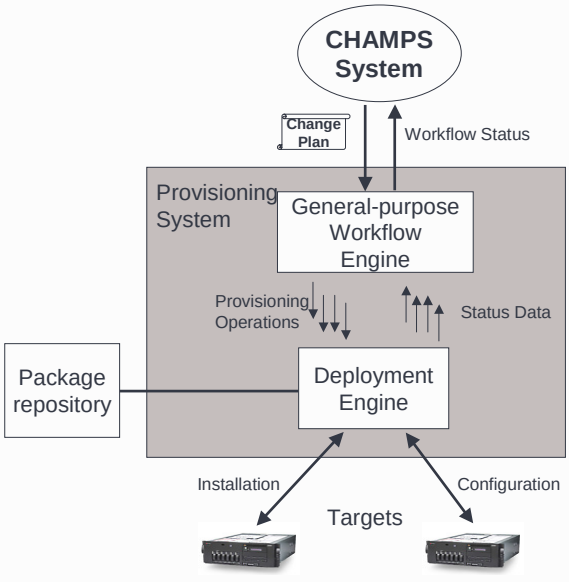

Fig. 2. Architecture for extending a Provisioning System with a Workflow Engine

allows the programmatic invocation of individual operations from an application outside of the provisioning system by means of SOAP[3] messages. We exploit this feature by feeding the Change Plans created by the CHAMPS Planner & Scheduler into a general-purpose workflow engine and invoke individual operations directly from there. More specifically, we use the BPWS4J workflow engine [2] that is able to execute workflows and business processes specified using BPEL4WS. A BPEL4WS workflow describes Web Services interactions and may comprise parallel execution (so-called *flows*), sequences, conditional branching, time-out mechanisms, as well as error and compensation handling.

By doing so, we can execute provisioning tasks defined in change plans concurrently. The architecture of the extended provisioning system (depicted in figure 2) is consequently composed of two sub-systems: the BPWS4J workflow engine and the deployment engine of the provisioning system. The former interacts with the CHAMPS system (cf. section 3.1), as follows: First, the **workflow engine** inputs the change plan provided by CHAMPS and starts each provisioning operation by directly invoking the deployment engine. These invocations are performed either in parallel or sequentially, according to the change plan. In a second step, the **deployment engine** is invoked by the workflow engine and performs the provisioning operations. It 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. Figure 2 also shows that status feedback happens at two stages:

First, the interactions between the deployment engine and the workflow engine (i.e., the invocations of provisioning operations and the assessment of their execution). 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 that would perform such checks, as these conditions are specified in the temporal constraints (so-called *links*) that connect the activities in a workflow.

The second status feedback loop comprises the interactions between the workflow engine and the CHAMPS Planner & Scheduler, i.e., submitting the change plan and receiving status feedback from the workflow engine. This is needed to perform plan adjustments in case the roll-out of changes runs behind schedule.

4 Prototype Implementation

The implementation of our prototype demonstrates the invocation of the TIO deployment engine from the BPWS4J engine, based on the change plans submitted by the CHAMPS system (see figure 3). More specifically, the implementation addresses the following aspects:

First, one needs to create Web Services Description Language (WSDL) [13] wrappers for the existing TIO invocation interface. Making TIO appear as a (set

[3] Simple Object Access Protocol.

of) Web Service(s) is a necessary step to providing a seamless integration with the BPWS4J workflow engine, as every BPEL4WS *invoke* operation refers to a Web Service. The WSDL wrappers define the allowable set of change management operations that can be used in change plans.

Once this is done, one can invoke the operations defined in the WSDL interfaces by submitting a BPEL4WS workflow (corresponding to a change plan) to the workflow engine, which allows the execution of several operations in parallel.

Third, the deployment engine needs to monitor the execution status of the change plans to determine whether they are still running, completed successfully, or completed with an error. This is important because the workflow engine depends on this information to determine if all the preconditions are satisfied before the next activity can be triggered.

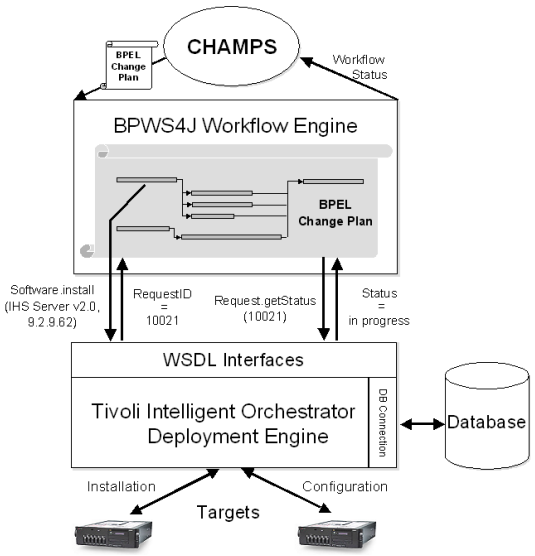

Fig. 3. Integrating the TIO Provisioning System with the CHAMPS Change Manager

Finally, a change plan may specify deadlines (e.g., task X must be finished by 8pm) that need to be enforced. The workflow engine must therefore be able to send an event back to the CHAMPS Planner & Scheduler if a provisioning activity takes longer than initially planned. The Planner & Scheduler would then decide if the provisioning process should be abandoned (and rolled back to a previous state), or continued despite the delay. In the following four sections, we will discuss how we addressed each of these aspects in more detail.

4.1 WSDL Wrappers for Logical Operations

To facilitate the invocation of provisioning operations from the outside, TIO can represent each individual operation or sequence of operations as a so-called *logical operation*. In TIO, each resource is treated as a component (i.e., Software, Operating Systems, Switches, Servers, etc.) that provides an (extensible) set of logical operations. Typically, the TIO component dealing with software provides logical operations such as *Software.deploy*, *Software.install*, *Software.start*, while its switch component provides *Switch.createVLAN*, *Switch.turnPortOn* etc. For example, the logical operation *Software.install* can be used to implement the IBM HTTP Server (IHS) install operation in the TIO sequence depicted in

figure 1. In addition, the use of logical operations ensures that the TIO database gets updated with execution status information.

A first part of our work consists in providing WSDL interfaces to facilitate the invocation of these logical operations using server IP addresses, software identifiers, or device serial numbers as inputs. As an example, we have created the following WSDL interface to perform the logical operations (*Software.Install*, *Software.Start*, etc.) on the software component:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <definitions name="SoftwareComponent">
3   <message name="installRequest">
4    <part name="softwareName" type="xsd:string"/>
5    <part name="deviceIP" type="xsd:string"/>
6   </message>
7   <message name="installResponse">
8    <part name="requestID" type="xsd:int"/>
9   </message>
10  <portType name="SoftwareComponent">
11   <operation name="install"
12    parameterOrder="softwareName deviceIP">
13     <input message="tns:installRequest"
14      name="installRequest"/>
15     <output message="tns:installResponse"
16      name="installResponse"/>
17   </operation>
    ...
18  </portType>
19 </definitions>
```

Fig. 4. Software Component WSDL interface

The listing depicted in Figure 4 shows the WSDL definition of *SoftwareComponent.install* (lines 10-17) that wraps the TIO logical operation *Software.install*. It uses the software name and server IP address as inputs (definition of the input message installRequest, lines 3-6) and returns a request ID (definition of the output message installResponse, lines 7-9). This approach can be generalized to accomodate other resources, such as switches, server pools or VLANs.

4.2 Invoking Logical Operations concurrently

The BPWS4J workflow engine [2] allows us to invoke several logical operations simultaneously through the above WSDL interfaces. As mentioned in section 3.1, the CHAMPS system uses the BPEL4WS [1] workflow language to describe change plans: the invocations of logical operations are done through our WSDL interfaces and by using the *invoke* construct of BPEL4WS; parallel and sequential execution paths map to the *flow* and *sequence* structured activities.

The deployment engine is driven by the workflow engine and thus able to execute tasks concurrently, such as the installation of the IHS server and the deployment of the web site content (HTML pages, pictures etc.). An example, briefly mentioned in section 2.1, is given in figure 5. It depicts a part of the change plan, defined in BPEL4WS and rendered in the BPWS4J workflow editor, for the simultaneous installation and configuration of two websites with different content, along with IHS servers on two different systems running Linux: The website with the name WSLA (together with the HTTP server) is to be provisioned on the system 'cuda' having the IP address 9.2.9.64 (dashed lines in the figure), while the system 'charger' with the IP address 9.2.9.63 will host another HTTP server and the website DSOM2003 (dotted lines in the figure).

One can see that using the BPEL4WS *flow* construct yields the advantage of decoupling the provisioning processes on a per-system basis: if the provisioning process on one system encounters a problem, the provisioning of the second system remains unaffected by this and continues. Concurrent invocations of the change management operations can be carried out because the invocation of a logical operation on the provisioning system through its WSDL interface is non-blocking.

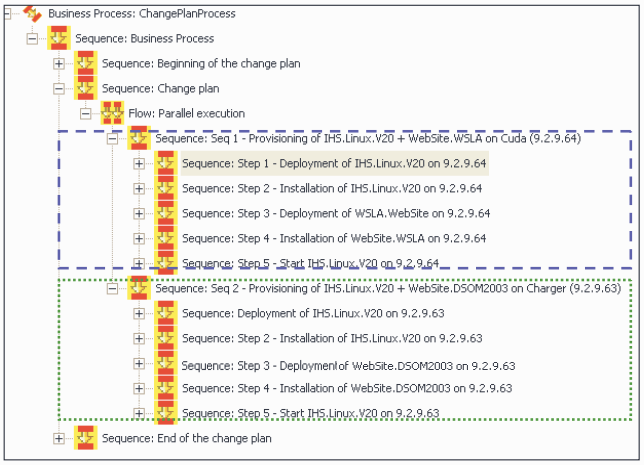

Fig. 5. Concurrent Provisioning of 2 Websites

4.3 Monitoring the Execution of Change Plans

In addition to tasks that can be carried out in parallel, a Change Plan contains temporal constraints between various tasks that need to be taken into account as well. As an example, every *invoke* operation within a *sequence* can only start if its predecessor has finished. To retrieve the execution status of a logical operation from the provisioning system, we have created a second set of WSDL interfaces (listed below). This information is needed by the workflow engine to determine if a task within a *sequence* is still running, or whether it can proceed with the execution of the next task. As an example, in figure 6, the operation *Request.getStatus* (lines 11-17) returns the start time and the status of an execution (definition of the getStatusResponse message, lines 6-8) from a request ID (definition of the getStatusRequest, lines 3-5).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <definitions name="Request">
3   <message name="getStatusRequest">
4    <part name="requestID" type="xsd:int"/>
5   </message>
6   <message name="getStatusResponse">
7    <part name="startTime" type="xsd:date"/>
8    <part name="status" type="xsd:string"/>
9   </message>
10  <portType name="Request">
11   <operation name="getStatus"
12    parameterOrder="requestID">
13     <input message="tns:getStatusRequest"
14      name="getStatusRequest"/>
15     <output message="tns:getStatusResponse"
16      name="getStatusResponse"/>
17   </operation>
18  </portType>
19 </definitions>
```

Fig. 6. WSDL interface for Status Monitoring

To determine the execution status of a logical operation, the workflow engine periodically invokes the monitoring WSDL interface. An example of how this can be expressed in BPEL4WS is depicted in figure 7. First, the change management operation *Installation of IHS on 9.2.9.64* is invoked through the WSDL interface corresponding to the appropriate *Software.install* logical operation. In a second step, the request ID returned by the invocation is used to check periodically (through the monitoring WSDL interface implementing the method *RequestComponent.getStatus*) the status of the running logical operation, until it completes. If this is the case, the next change management operation of the workflow is started. Our implementation distinguishes between 3 states for an executing workflow: *in progress*, *completed* (with success), and *failed* (the return message includes an error code). If an error occurs during the execution of a logical operation, the workflow engine returns an error message back to the CHAMPS Planner & Scheduler, which then needs to determine how to proceed further. This may involve rolling back and subsequently retrying the operation, or bringing the target system(s) back to a well-defined state.

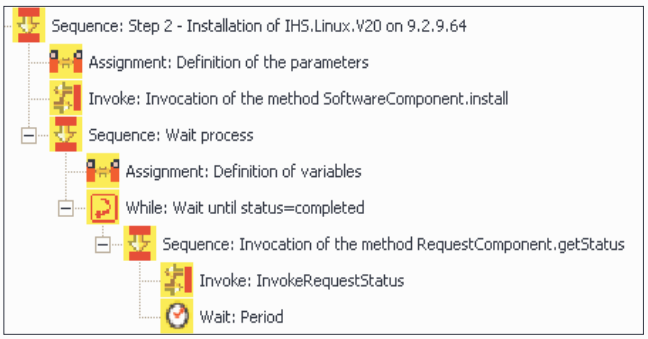

Fig. 7. Monitoring Task Execution Status

By using the WSDL monitoring interface, we are able to enforce temporal constraints defined in Change Plans such as: *the logical operation X must be finished before the logical operation Y can start*, or *the logical operation X must not start before the logical operation Y has started*. For a detailed discussion of the various temporal constraints in Change Plans, the reader is referrred to [6].

4.4 Enforcing Deadlines and Durations

An additional requirement is the enforcement of deadlines for change management operations that are given in a Change Plan. To do so, the workflow engine needs to understand what these deadlines are and notify the CHAMPS Planner & Scheduler in case a change management operation runs behind schedule. The Planner & Scheduler would then decide if the change management operation should be abandoned (and roll back the system to a known state), or if it should continue despite the delay.

Yet again, we are able to exploit the features of the BPEL4WS language to specify time constraints on the provisioning workflow. Activities corresponding to invocations of logical operations can be grouped together by means of the *scope* structured activity. An *event handler* is then attached to a *scope* activity, which

may contain one or more alarms. Each alarm is defined by both a constraint and an escape activity, which is performed when the constraint is violated. This mechanism works for single activities as well.

We use alarms to define time constraints (the BPWS4J workflow engine comprises a timer) so that we can specify deadlines ("must be finished by 8PM") as well as impose limits on the duration of an activity ("must take less than 45 minutes"). The escape activities allow us to notify the CHAMPS system whenever an activity violates its time constraints. In figure 8, we place an activity (a software installation) within a *scope* and define the time constraint *duration* $<$ *5 min*. If the duration exceeds the time period defined in the change plan, the escape activity attached to the alarm invokes a WSDL method of the CHAMPS Planner & Scheduler to report the violation. Note that a notification does not mean that the change plan is automatically aborted. Instead, the Planner & Scheduler will determine how to proceed, according to the overall system state, other (competing) change plans, as well as penalties specified in Service Level Agreements or general Policies. It will then decide if the current change plan can continue, if it has to be cancelled, or if a new change plan must be generated later.

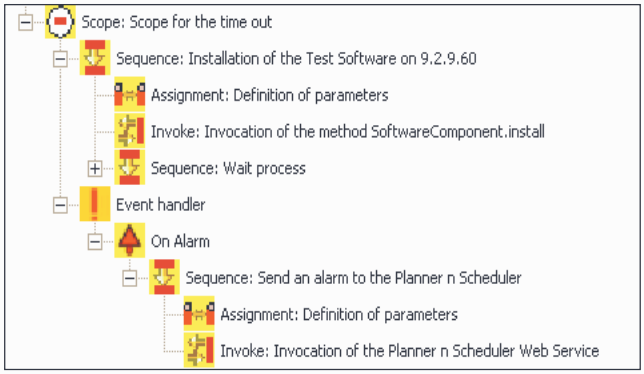

Fig. 8. Enforcing Deadlines and Durations

5 Conclusion and Outlook

We have presented a novel approach for integrating a change manager with a service provisioning system to facilitate the workflow-based provisioning of application services. Our work was motivated by the extremely high rate of change in emerging e-Commerce environments and the need for integrating service provisioning into the change management process. By using a standardized, general-purpose workflow language for expressing change plans and demonstrating the feasibility of integrating a common-off-the-shelf workflow engine with a commercial provisioning system, our approach is applicable to a wide range of provisioning scenarios.

Our prototype demonstrates that change plans, generated by the CHAMPS change management system, can be executed by the TIO deployment engine and that the BPEL4WS workflow language can be used effectively to describe change plans. While this advantage is likely to apply to other workflow languages

as well, BPEL4WS has the additional benefit that it is specifically targeted at Web Services. Second, the use of a workflow engine yields the advantage that the task of checking the execution status of activities in a distributed system (to decide if the next activity in a workflow can start) can be completely offloaded to the workflow engine. Finally, we are able to achieve a very high degree of parallelism for a set of tasks: In the running example we used throughout this paper, provisioning a single website (server software and web content) took 185 seconds on average, whereas provisioning additional websites added less than 5% of overhead in terms of provisioning time per site.

While these initial results are encouraging, there are several areas of further work: As an example, we are currently working on extending our approach to address the deployment of more complex multi-tiered application systems involving Web Application Servers and Database Management Systems. Further promising research topics are advanced error-handling and rollback facilities, and the automated service composition and aggregation.

References

1. Business Process Execution Language for Web Services Version 1.1. Second Public Draft Release, BEA Systems, International Business Machines Corp., Microsoft Corp., SAP AG, Siebel Systems, May 2003. http://www-106.ibm.com/developerworks/library/ws-bpel/.
2. *Business Process Execution Language for Web Services JavaTM Run Time (BPWS4J)*. http://www.alphaworks.ibm.com/tech/bpws4j.
3. M. Cheung, A. Clemm, G. Lin, and A. Rayes. Applying a Service-on-Demand Policy Management Framework to an ETTx Environment. In R. Boutaba and S.-B. Kim, editors, *Proceedings of the Application Sessions of the 9th IEEE/IFIP Network Operations and Management Symposium (NOMS'2004)*, pages 101 – 114, Seoul, Korea, April 2004. IEEE Publishing.
4. E. Manoel et al. *Provisioning On Demand: Introducing IBM Tivoli Intelligent ThinkDynamic Orchestrator*. IBM Corporation, International Technical Support Organization, Research Triangle Park, NC 27709-2195, December 2003. IBM Redbook, Order Number: SG24-8888-00.
5. IT Infrastructure Library. *ITIL Service Support*, June 2000.
6. A. Keller, J.L. Hellerstein, J.L. Wolf, K.-L. Wu, and V. Krishnan. The CHAMPS System: Change Management with Planning and Scheduling. In R. Boutaba and S.-B. Kim, editors, *Proceedings of the 9th IEEE/IFIP Network Operations and Management Symposium (NOMS'2004)*, pages 395 – 408, Seoul, Korea, April 2004. IEEE Publishing.
7. F. Maurer and B. Dellen. Merging Project Planning and Web-Enabled Dynamic Workflow Technologies. *IEEE Internet Computing*, May 2000.
8. J.A. Nilsson and A.U. Ranerup. Elaborate change management: Improvisational introduction of groupware in public sector. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, 2001.
9. 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.

10. A. Sahai, S. Singhal, V. Machiraju, and R. Joshi. Automated Policy-Based Resource Construction in Utility Computing Environments. In R. Boutaba and S.-B. Kim, editors, *Proceedings of the 9th IEEE/IFIP Network Operations and Management Symposium (NOMS'2004)*, pages 381 – 393, Seoul, Korea, April 2004. IEEE Publishing.
11. G. Valetto and G. Kaiser. Using Process Technology to control and coordinate Software Adaptation. In L. Dillon and W. Tichy, editors, *Proceedings of the 25th International Conference of Software Engineering (ICSE 2003)*, pages 262 – 272, Portland, OR, USA, May 2003. IEEE Computer Society.
12. Y-M. Wang, C. Verbowski, J. Dunagan, Y. Chen, H.J. Wang abd C. Yuan, and Z. Zhang. STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support. In *Proceedings of the 17th Large Installation Systems Administration Conference (LISA 2003)*, pages 159 – 172, San Diego, CA, USA, October 2003. USENIX Association.
13. Web Services Description Language (WSDL) 1.1. W3C Note, Ariba, International Business Machines Corp., Microsoft Corp., March 2001. http://www.w3.org/TR/wsdl.