Abstract

Electronic contracts play a key role in the automation of information technology (IT) processes by establishing a common context and a formal set of mutual agreements and expectations between the various parties involved in today’s enterprise applications. We describe a novel approach for applying different types of electronic contracts to a complex IT management scenario with a specific focus on change management. Our implementation of a change management system and its quantitative evaluation shows that the use of electronic contracts helps in breaking down a complex change management problem into tractable parts and improves the quality of the accomplished solution.

1. Introduction

Over the past few years, the IT Infrastructure Library (ITIL) [6], comprising several disciplines such as service management, support and delivery, has established itself as the most widely used process-based approach to IT service 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, and consequently to improve the day-to-day operations of the organization’. Change Management is central to ensuring the availability, reliability, and quality of information technology (IT) services and processes. ITIL identifies the key major activities, the roles and stakeholders that perform them and artifacts that are either consumed or produced by the various steps in the Change Management process. A simplified version of the ITIL change management process is depicted in Figure 1: it consists of a set of linked activities, each associated with a role that performs them and some of the change artifacts. The change management process starts with the submission of a Request for Change (RFC) and produces a Forward Schedule of Change. Other important change artifacts 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 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.

Figure 1 Sample roles, activities and artifacts in the ITIL Change Management Process.

Although we have left out several activities relating to the test of changes, roles and artifacts (as well as the decision logic that verifies the accuracy of the results for each activity) for the sake of space, one can see from Figure 1 that implementing a change management process in a service provider environment is a major and costly 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 rollout of complex changes in large-scale environments takes at least several days, and in many cases lasts several weeks or even months. In practice, the vast majority of the time is spent on (1) assessing the impact of a Change, (2) obtaining the necessary
approvals, and (3) creating a Forward Schedule of Change that specifies in which order tasks need to be carried out to transition a system from one workable state into another workable state. 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 involved in the process, and electronic contracts play a key role in negotiating, defining and exposing the necessary information to the various stakeholders in the change management process. This paper is about the role and characteristics of electronic contracts a change management system needs to take into account, and the lessons we learned during its implementation: over the past 2 years, we have implemented CHAMPS, a prototype of a schedule-optimizing change management system [7]. CHAMPS formulates Change Management (comprising the provisioning, deployment, installation and configuration steps) as an optimization problem and automatically generates—based on various requirements, each potentially stemming from a different party—change management workflows that address these goals. Subsequently, these workflows are executed by a common-off-the-shelf provisioning system [8, 10]. Throughout the paper, we illustrate the various types of electronic contracts used in CHAMPS by means of concrete examples and provide a critical assessment of their usability. The paper is structured as follows: Section 2 sets the stage for the following sections by providing a brief overview of CHAMPS, introducing the four different types of contracts, and presenting the scenario of provisioning a multi-tiered eCommerce application, which we will use throughout the paper. Subsequently, we will examine some of the contract types in more detail: section 3 addresses the elements of the RFCs used in CHAMPS. SLAs, expressed in the WS-Agreement language, are discussed in section 4. Section 5 contains a quantitative evaluation of the CHAMPS performance for provisioning the multi-tiered eCommerce solution. Section 6 discusses related work. Our conclusions and future directions are contained in section 7.

## 2. Change Management

### 2.1. Building Blocks of the CHAMPS Change Management System

CHAMPS is a change management system developed at IBM Research for CHAnge Management with Planning and Scheduling [7]. It 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. Its optimization techniques, based on mathematical scheduling theory, allow the CHAMPS system to come up with a high quality solution for a mathematically intractable problem.

![Figure 2 Overview of the different Contract types in a Change Management System.](image)

As depicted in Figure 2, the CHAMPS change management system consists of two major components: First, the Task Graph Builder breaks down an incoming RFC into its elementary steps and determines the order in which they have to be carried out. We call a representation of such information Task Graph. A task graph is a workflow, expressed in the WS-BPEL [2] workflow language, consisting of tasks and precedence constraints that link these tasks together. Consequently, task graphs can be modified and aggregated by an administrator using general-purpose workflow editors. In addition, they can be stored for subsequent reuse as they are not bound to specific target systems in a service provider’s data center.

In a second step, one or more task graphs 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 dead-
lines and maximizes the degree of parallelism for tasks according to precedence and location constraints expressed in the Task Graphs. Again, the WS-BPEL workflow language is used to express a Change Plan, which facilitates further manual modifications and extensions by an administrator, if needed. Once the change plan has been computed by the Planner & Scheduler, it is input to the Provisioning System, which executes the requested changes on the data center resources. 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.

2.2. Contract Types for Change Management

Electronic contracts that CHAMPS consumes come in various forms (see Fig. 2) and are authored by different parties at very different points in time. Each of them fulfills a specific purpose and has a different format. The information in all of them needs to be combined by the change management system to make sure that the change plan generated by the system does not violate the constraints and requirements expressed in each of them, or at least, keep eventual violations at a minimum. The four types of contracts used in CHAMPS are as follows:

Requests for Change (RFCs): ITIL specifies that the change management process starts with the submission of a Request for Change by a customer. Many RFCs may be submitted concurrently. The RFC describes what is to be done, usually in terms of hardware/software components to change, the change management operation (deploy, install, configure, uninstall), as well as the deadline by which the change needs to be completed. Examples include changing the schema of a database table in a running application, or installing a new release of a web application server in a multi-tiered eCommerce solution. We note in passing that a solution is composed of a variety of components (hardware/software/operating system/middleware etc.).

Deployment Descriptors: An important observation is that many changes are not explicitly included in the RFC. Rather, they are merely implied. For example, applications must be recompiled if they use a database table whose schema is to change. Such implicit changes are a result of various kinds of relationships, such as service dependencies and the sharing of the service provider’s resources among different customers. Dependencies express compatibility requirements between the various components of which a solution is composed. Such requirements typically comprise software pre-requisites (components that must be present on the system for an installation to succeed), co-requisites (components that must be jointly installed) as well as ex-requisites (components that must be removed prior to installing a new component). In addition, version compatibility constraints and memory/disk space requirements need to be taken into account. All of this information is typically captured during the development and build stages of components, either by the developer, or by appropriate tooling. We observe that deployment descriptors are not contracts in the classical sense: For example, they are established by a single party and their content has not been negotiated. Nevertheless, deployment descriptors contain information that influences the behavior of the change management system and thus help avoiding a negative impact on other contractual parties. We argue that deployment descriptors can be regarded as electronic contracts between the different components of which a solution is composed. It is an important duty of the change management system to perform automated matchmaking between the component requirements and capabilities, both exposed in these contracts, to achieve a workable solution. The deployment descriptors we use have been defined in the Solution Installation for Autonomic Computing initiative. Solution Install is a recent technology whose goal is to facilitate the provisioning and change management of multi-tiered distributed applications, referred to as solutions. Solutions comprise multiple levels of potentially nested Installable Units (IU) that may be distributed across different systems. 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. The level of detail included in IUDDs goes beyond traditional J2EE deployment descriptors. Furthermore, IUDDs can be applied to any kind of distributed application. For details on the IUDD schema and the IU packaging format, the reader is referred to [3] and [17].

Policies and best practices: Policies represent the business objectives of a service provider. Examples of such policies in the change management context are: minimize the downtime caused by a change, maximize the profit for a change, maximize the number of RFCs handled (aka throughput), minimize the total execution time, or a combination of any of the above. These policies may be surfaced to the customer (and then become part of an SLA, see below), but in real-life environments they are not because they reflect the core business model of a service provider. Best practices specify a variety of additional administrative constraints that need to be enforced by the change management system. Examples are: 'all database servers
assigned to a customer must run the same version of the DBMS’ or ‘if a web application server in a cluster gets upgraded, all other application servers need to be upgraded as well’. Both policies and best practices are authored by the service provider.

**Service Level Agreements (SLAs):** An important aspect of a contract for IT services between a service provider and a customer is the set of Quality of Service (QoS) guarantees and obligations. This is commonly referred to as a Service Level Agreement. Today, SLAs between organizations are used in all areas of IT services – in many cases for hosting and communication services but also for help desks and problem resolution. Furthermore, the parameters for which service level objectives (SLO) are defined come from a variety of areas, such as business process management, service and application management, and traditional systems management.

### 2.3. Case study: multi-tiered SPECjAppServer2004 provisioning

Our case study that will help us evaluate the effectiveness of our approach is based on the scenario of a customer requesting a service provider to install and configure 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 [15]. It is a complex, multi-tiered on-line e-Commerce application that emulates an automobile manufacturing company and its associated dealerships.

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

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 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 application 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. Installable Unit Deployment Descriptors (IUDD), specified in [17], provide a means of expressing dependencies between software components.

Figure 3 depicts that a set of IUDDs represents a directed, acyclic graph. While dependencies between the various types of Installable Units describe a software containment hierarchy, Solution Install further classifies dependencies into various types (such as hosts, uses, contains, supersedes, etc.), which 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 component replaces the referenced software component, while a *hosts* relationship indicates that a software component resides in the referenced hosting environment. Consequently, each dependency type needs to be handled differently by CHAMPS.

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.

CHAMPS coordinates the change management operations across the hosting environments and relies on Solution Install to parse a set of submitted deployment descriptors in order to build an in-memory model of the various components and their dependencies. This dependency model is the input data to the CHAMPS Task Graph Builder, which generates a provisioning workflow to carry out the change. To do so, it carries out the necessary requirement checks, evaluates the dependencies, derives the proper order in which change management operations need to be carried out, and executes them on the target systems. 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. Based on the dependencies (or lack thereof), the Task Graph Builder is able to determine the order in which provisioning activities need to be carried out, and whether activities may happen concurrently.

While one could argue that the use of Solution Install again for a variety of change management scenarios, are created once at build time, packaged with software install images, and can be reused over and over again for a variety of change management scenarios.

3. Requests for Change

RFCs are created by a customer or an administrator and submitted to CHAMPS by means of a web-based graphical user interface that implements a wizard. An RFC is declarative in nature; it states what needs to be accomplished, but leaves the procedural details (i.e., how the change is carried out) open. Based on the information submitted in the RFC, the change management system needs to correlate the RFC with (1) SLAs the customer has with the service provider (2) deployment descriptors for the (software) components that are subject to the change and (3) evaluate what policies the service provider has in order to guide the scheduling of the change. Examples of parameters supplied in an RFC are:

- The Customer ID that identifies the submitter. It is used to correlate the RFC with the SLA(s) a customer may have with the service provider.
- The Deadline at which the change is supposed to be carried out along with a Constraint Type. Such a combination is typically used in GANTT charts to describe the constraints a task is subject to. We apply them to the overall RFC as they suit our purposes nicely. There are eight possible constraint types (as soon/late as possible, must start/finish on, start/finish no later/earlier than). Note that the deadline submitted by the customer need not be identical to the date/time when the change is actually carried out: the CHAMPS Planner & Scheduler takes the decision when to schedule a change based on the consideration of service provider policies and SLAs of customers whose systems might be impacted by the change (in case of shared resources). However, the customer can set a flag indicating whether the change is required or optional to provide CHAMPS with additional prioritization information.

Upon selection of a Change Category indicating the type of change according to a domain-specific taxonomy (e.g., service, solution, software, hardware, utility) a user is presented with a list of components that are subject to a change and for which a task graph (i.e., a workflow) already exists. In the SPECjAppServer2004 case, an item ‘SPECjAppServer2004 solution’ would appear in the list. Alternatively, a customer can indicate a location for a deployment descriptor for which CHAMPS then generates a task graph (see [7] for details on how task graphs are generated).

The Change Management Operation indicates whether the user would like to perform a new installation, start or stop an existing system, request an upgrade, or the removal of a software component.

Finally, a Change Abstract, a brief textual description of the change, is entered.

Once the customer has submitted the RFC, the change management system assigns a unique numeric identifier, RFC Number, to the RFC to track its execution.

Note that – in contrast to change management systems used today – a customer does not necessarily have to specify an expected duration of a change as CHAMPS estimates (from previous similar deployments) how long a change is likely to take. In case no such information exists, a customer is asked on whether he wishes to provide an estimate.

4. Expressing Service Level Objectives with WS-Agreement

Given its focus on Web Services, the use of WS-Agreement [1] for our purposes is a natural choice. We assume that a set of SLAs has already been negotiated between a service provider and its customers and made available to both parties. Consequently, we neither leverage the agreement template capabilities nor the lifecycle management interfaces the WS-Agreement specification defines, but rather focus on the applicability of the WS-Agreement schema to express profit and costs functions within an SLA. The WS-Agreement (in short: wsag) schema draws heavily on the previously defined Web Service Level Agreement (WSLA) language [9]. However, many of the WSLA capabilities such as compliance monitoring by third-parties, sophisticated mechanisms for expressing and aggregating resource metrics, as well as detailed constructs for specifying service level guarantees and objectives are listed as non-goals in WS-Agreement and therefore missing. They might be defined at a later point in time. In its current stage, the WS-Agreement schema provides a set of high-level constructs, whose details need to be provided by domain-specific extensions. The goal of this paper is not to define such extensions, but rather to investigate to which extent the
wsag schema can be applied ‘as-is’ to our change management scenario.

4.1. How does a Service Provider identify its Customer? wsag:Context

In WS-Agreement, a variety of (optional) constructs that relate to the involved parties, the SLA expiration date, references to the template from which it was created (TemplateName) as well as to related agreements collectively form the Context of an agreement. A service provider typically holds one agreement for each customer (termed ‘consumer’ in wsag-parlance), which contains the agreed-upon terms for all the services a customer has subscribed to. Before provisioning a service for a given customer, the provider’s Change Manager needs to look up the terms of the SLA he has with the customer. Finding the customer in a WS-Agreement, however, is not a trivial task as the WS-Agreement schema does not contain elements from which one can immediately obtain this information. Instead, it takes the approach to represent the contractual parties for both the negotiation and enforcement phases by three elements.

The first two, AgreementInitiator and AgreementProvider contain identification and contact information for contractual parties that is typically specified in a separate, domain-specific schema (both elements are of XML type any). The linkage between the negotiation and enforcement phases is done by a third element, AgreementInitiatorIsServiceConsumer, a boolean flag. By avoiding the explicit representation of information relating to both phases, there is no duplication of information relating to contractual parties. As the wsag-schema is confined to agreements between two parties, one should be able to glean which party plays the role of service provider and which one is the customer. However, as all the three elements are optional in the wsag-schema, a set of combinations (some of them inconsistent) may arise, which need to be handled by custom logic, as depicted in Figure 4.

One can see that uniquely identifying a customer while being able to detect possible inconsistencies that cannot be resolved by a syntactic XML validator requires the execution of a set of nested if-then-else statements, thus introducing some complexity for the implementer.

4.2. What are the Profit and Cost Functions? Example of a wsag:GuaranteeTerm

For our scenario, we are particularly interested in expressing profit and cost functions by means of WS-Agreement. Typical SLAs used in practice partition the SLA evaluation intervals into a set of distinct time periods to express the different profits and costs that occur within a given part of the day: For example, a customer may want to specify a different penalty for the lack of availability of a server hosting its application during a ‘prime shift’ that lasts e.g., from 9:00am to 5:00pm than for downtime during the night (5:00pm to 9:00am). In addition, maintenance intervals (e.g., from 2:00am to 3:00am) need to be defined, in which a server can be taken down for administrative tasks without causing a penalty. These different time periods can be conveniently defined in a WS-Agreement by means of the Name element of a GuaranteeTerm. (cf. Figure 5).

The ServiceScope of a GuaranteeTerm contains a reference to the service description through the ServiceName. We use the ServiceLevelObjective element to express the time period in a machine-readable format so that it can be interpreted automatically. However, a more flexible and powerful mechanism for expressing time periods, such as the policy time period conditions defined in [12], could be applied here as well by adding domain-specific extensions to the wsag schema.

For each of these time periods, both profit and cost considerations of a service are typically expressed as step functions that assign a monetary value to an AssessmentInterval. Such intervals can be cascaded to express that additional penalties are due if the problem is not addressed within a given amount of time. By doing so, a customer can specify that, e.g., for each minute of downtime, the service provider incurs a cost of USD 5.00. For every additional 10 minutes of an outage, an additional penalty of USD 50.00 comes into
effect. For every half hour of an outage, an additional penalty of USD 500.00 needs to be paid etc. In Figure 5, the values of the TimeInterval element (PT1M, PT1H etc.) correspond to the aforementioned lengths of an assessment interval. The very same concept applies to the reward a service provider collects for making the service available.

**WS-Agreement**

Service-oriented/Contracting/Service/ServiceLevel/ServiceLevelObjective

Average penalties

Average BusinessValueList

Figure 5 Sample WS-Agreement GUARANTEE TERM comprising profit and cost functions. This element is used to specify the time periods to which rewards and penalties apply.

Overall, WS-Agreement provides a sound basis for defining simple SLAs whose rewards and penalties are based on step functions, as in our example. While it is possible to define any kind of penalty/reward functions (e.g., linear or polynomial functions), one has to keep in mind that evaluating these values requires additional custom logic because WS-Agreement does not provide arithmetic operators.

5. SPECjAppServer2004 provisioning with CHAMPS: Evaluation

To evaluate the efficiency and effectiveness of CHAMPS with respect to scheduling and rolling out changes, we compared the times it takes a provisioning system to provision SPECjAppServer2004 and compare them to a CHAMPS-driven version of the SPECjAppServer2004 scenario: The 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.

<table>
<thead>
<tr>
<th>Activity type</th>
<th>Average time [minutes:seconds] for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration (incl. DB creation and loading of data)</td>
<td>n/a</td>
</tr>
<tr>
<td>Start</td>
<td>1.36</td>
</tr>
<tr>
<td>Total time (CHAMPS workflow)</td>
<td>34 min 00 sec</td>
</tr>
<tr>
<td>Total time (strictly sequential, no think time)</td>
<td>50 min 25 sec</td>
</tr>
</tbody>
</table>

Figure 6 Times for CHAMPS-based SPECjAppServer2004 provisioning based on multiple runs.

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, the 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 into account the (non-)existence of dependencies between the components of the solution, 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 6 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.

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 dura-
tions of the start and configuration activities. However, the former activities can be carried out concurrently as they happen on 2 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.

6. Related Work

Our work falls in the area of Customer Service Management [14], which combines concepts from electronic contracting [11] with traditional Network and Systems Management [5]. Both disciplines have assembled an extensive body of knowledge; however, very little work exists that combines them. There are two major reasons why this is so: First, the notion of customers guiding the orchestration of changes, specifically in shared resource environments, is still foreign to the systems management community: traditionally, the users of management systems are expert administrators who have acquired many years of expertise and who are reluctant to delegate parts of their responsibilities to automated systems that take customer concerns into account. Second, the vast majority of SLAs in use today are very often documents written in natural language, and therefore not amenable to automated interpretation. Technologies that allow a customer the precise formal specification of SLAs, such as WS-Agreement and WSLA are just beginning to emerge.

That said, there are pieces of our approach that do connect with existing work in the field. Most closely related is [13], which advocates that non-compliance with a given customer SLA may very well be a viable business option if the SLAs of different customers compete for a limited set of resources. This is comparable to our notion of having the change management system decide if and when an RFC of a customer will be executed (depending on resource availability and other SLAs) while still letting the customer specify the timeframe. In addition, [4] describes an approach to annotate WS-BPEL activities with WSLA-based service level objectives. Finally, our approach of workflow-driven resource orchestration bears some resemblance to the Workflakes system [16], which uses workflows to perform adaptation and reconfiguration tasks as well. However, Workflakes requires an administrator to manually create workflows, while CHAMPS is able to generate them.

7. Conclusions and Outlook

We have presented a novel approach to automated change management based on electronic contracts. Our research is motivated by the desire of enterprises and service providers to accomplish significant labor cost reductions and drastically reduce the time it takes to roll out changes in large-scale corporate and service provider environments, literally from months into minutes.

By leveraging a set of open standards such as Solution Install Deployment Descriptors and WS-Agreement, we were able to implement the CHAMPS change management system that can be considered as a first step towards this goal. During our implementation, we found that establishing a common context and a formal representation of mutual agreements and expectations (all expressed as electronic contracts) between the various parties involved in the change management process have been critical factors for the efficiency and effectiveness of the system. In particular, we were able to accomplish a reduction of about 30% in provisioning a multi-tiered eCommerce site that implements the SPECjAppServer2004 benchmark application.

The results presented in this paper are a starting point. Further research is needed to address the specification of more complex reward and penalty functions in WS-Agreement, and develop a better formalism to extend the language to set up fine-grained service level objectives. In addition, more work is needed to tie SLAs closer to resource instrumentations as well as support for the negotiation of Service Level Agreements. Finally, the standardization of an open format for expressing Requests for Change is an important prerequisite to facilitating the interoperability between a growing base of customers and service providers.

8. References


