REST-Based Management of Loosely Coupled Services

Heiko Ludwig, Jim Laredo, Kamal Bhattacharya
IBM TJ Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532, USA
{hludwig,laredoj,kamalb}@us.ibm.com

Liliana Pasquale
Politecnico di Milano
Dipartimento di Elettronica e Informazione
via Golgi, 40
20133, Milano Italy
pasquale@elet.polimi.it

Bruno Wassermann
University College London
Software Systems Engineering Group
Dept. of Comp. Science
London, UK
b.wassermann@cs.ucl.ac.uk

ABSTRACT
Applications increasingly make use of the distributed platform that the World Wide Web provides – be it as a Software-as-a-Service such as salesforce.com, an application infrastructure such as facebook.com, or a computing infrastructure such as a "cloud". A common characteristic of applications of this kind is that they are deployed on infrastructure or make use of components that reside in different management domains. Current service management approaches and systems, however, often rely on a centrally managed configuration management database (CMDB), which is the basis for centrally orchestrated service management processes, in particular change management and incident management. The distribution of management responsibility of WWW based applications requires a decentralized approach to service management. This paper proposes an approach of decentralized service management based on distributed configuration management and service process co-ordination, making use RESTful access to configuration information and ATOM-based distribution of updates as a novel foundation for service management processes.

Categories and Subject Descriptors
K.6 MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS

General Terms
Management

Keywords
Loosely coupled systems, service management, REST, Discovery

1. INTRODUCTION
Today's applications make use of the distributed platform that the World Wide Web provides. In enterprises and other organizations this leads to a distributed application infrastructure in which different elements of this infrastructure are owned and managed by different organizations. For example, this is the case in companies that use a ready-made Software-as-a-Service (SaaS) such as salesforce.com, owned by the service provider, that connects to in-house applications of that company such as a general ledger application, which is owned and operated by the company itself. In another scenario, an organization uses computing infrastructure from an outside provider as a "compute cloud" and operates its own applications on an operating system image on a cloud provider's hypervisor. In another case the company builds an application on a virtualization platform such as Google AppEngine [1] or Force.com [2]. From a software stack perspective, the first scenario represents a horizontal inter-domain relationship, the second and third scenarios vertical ones. Both types of inter-domain relationships can be present at the same time. These relationships can be either quite static and tight-knit, as in the case of the SaaS integrated to the company's backend system, or very loose, such as a company using a service in a mashup. The service provider may not even be aware of the set of other organizations currently using this service.

This dependency on infrastructure outside an organization's management domain has to be taken into consideration for system and service management. Current service management approaches and systems, however, typically rely on a centrally managed configuration management database (CMDB) to store information on a service system's configuration. It is the basis on which a service management processes of a management domain run, e.g., change management, incident management and problem management. The distribution of management responsibility of Web-based applications requires a decentralized approach to service management that takes into account the distribution of management information and the execution of management processes across organizational boundaries.

For example, an application accesses a storage service of another company using a Web service interface. The properties of the storage service including its interface specification is configuration information - managed by the storage service organization but relevant for the users of the service. If the company offering the storage service changes the signature of an operation accessing the Web service the service-using organization has to change the application invoking the service correspondingly. Changes are conducted in the course of change management processes. If the storage service conducts its change process independently of it users, their service will be disrupted when the new release goes into effect until the using applications have been adapted in an – independent – change process in each of the service's user organizations. Dealing with distributed configuration information and integrating service management processes across domains is essential to avoid service outages like the one illustrated above.

Some current approaches address issues raised above. CMDB federation enables accessing configuration information held in different CMDBs [3]. However, CMDB federation requires the explicit establishment of the federation relationship on the database level and does not scale to a large service user base as common for popular service providers due to high setup costs, even ignoring incompatible CMDB products. In addition, federation requires agreement from both parties involved. However, in a loosely

WWW 2009, April 20–24, 2009, Madrid, Spain.
ACM 978-1-60558-487-4/09/04.

coupled environment such as mashups, the service provider may not be tracking closely who is using its service. Today, service process integration typically is implemented by email announcements being sent to subscribers, e.g., for planned changes, and Web-based forms or emails triggering service processes within a provider, e.g., for reporting incident. This form of process integration requires human intervention and is thus error prone, expensive and lags in time.

The objective of this paper is to outline the issues service management poses in an environment of loosely coupled, Web-based systems and to propose an approach of decentralized service management based on distributed configuration management and service process co-ordination. Our approach is based on RESTful access to configuration information across domain boundaries, RESTful representation of service process state information as a basis for service process integration and ATOM-based distribution of updates as a novel foundation for service management.

The rest of the paper is structured as follows: In the next section we discuss in more detail the issues related to cross-domain service management in loosely coupled distributed systems. Section 3 then outlines our general approach. Subsequently, we discuss the Smart Configuration (SCI) approach to distributed configuration management. In section 5 we describe the implementation of change process integration based on SCI. In section 6 we discuss related work and, finally, summarize and conclude in section 7.

2. SERVICE MANAGEMENT IN A LOOSELY COUPLED ENVIRONMENT

Service management requires specific consideration in an environment of loose coupling of applications and other resources across boundaries of management domains.

In a single management domain, e.g. a single company, service management today is mostly conducted along the lines of various sources of best practices, e.g., the IT Infrastructure Library (ITIL) [5]. Figure 1 illustrates the main concepts of this management approach.

The service infrastructure used to produce a service is complemented by a service management stack. The assets of the service infrastructure, mainly hardware and licenses, are captured in an Asset Database. The state of hardware and software entities in the service infrastructure that can be configured are represented as Configuration Items (CIs), each of which having a set of properties describing its current settings. CIs can represent a router with its routing table as a property, a Web application server with its access control method, or a database instance having properties such as its set of tables, etc. CIs can have relationships between each other, typically expressing that one CI depends on another or an asset. Assets and CIs are the information on which service management processes are based. ITIL identifies a number of processes such as incident management (tickets), problem management, change management, release management, SLA management, asset management and more.

Service processes can trigger each other, e.g., a problem process can trigger a change process if fixing the problem requires a configuration change. Asset and configuration information are updated regularly in a discovery process that identifies new assets and CIs and changes in its configuration by searching for and analyzing systems on the network of a service infrastructure.

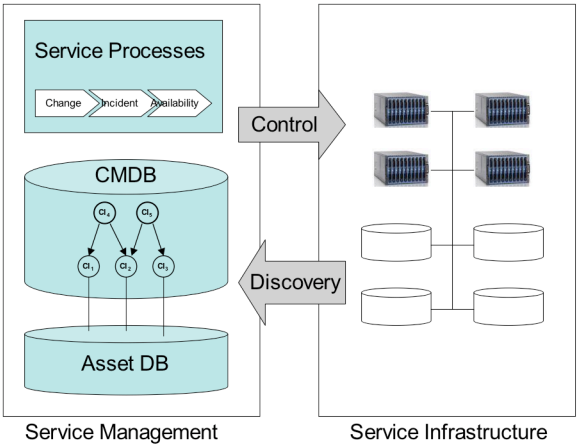

Figure 1: Single domain service management

CI and asset changes can also be triggered by service management processes, mainly the change process, updating the CMDB in the course of modifying the service infrastructure through control operations, i.e. change of configuration. In a single management domain, it is assumed that all assets and CIs relevant for the service management processes can be found in the Asset DB and CMDB and these CIs can be discovered by accessing the service infrastructure.

This assumption typically does not hold in a loosely coupled environment. The service infrastructure of one management domain accesses services in another management domain or is accessed by another. This entails CIs of one management domain depending on CIs of another. The following figure illustrates some scenarios.

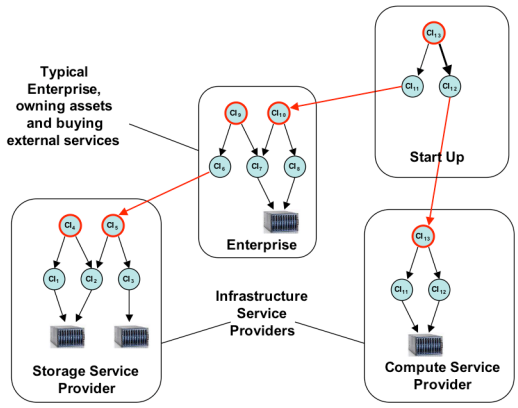

Figure 2: Distributed configuration example.

In figure 2 the rounded boxes represent the assets and CIs (circles) in different management domains – we disregard the actual service infrastructure in the remainder of the discussion. The bold circles represent CIs on which other management domains depend or may depend, e.g., a Cloud service provided by a compute service provider. It might have properties such as the type of hypervisor, the end point where to access the service, its IP address etc. In the scenario above we have two infrastructure service providers, one for storage and one for compute services. The service of the storage service provider is used by an enterprise, which has assets and CIs on its own but accesses an external storage service, e.g., for backup. A startup company may not have any assets of its own in its service infrastructure but uses services provided by others, e.g., the

compute services of an external provider and the services of an enterprise, which may be a credit card processor.

A number of issues arise in such a context for a management domain:

- It is difficult to discover assets and CIs outside one's own management domains due to the lack of scope and lack of access to the resources of another domain.
- It is difficult to keep track of which management domains are using and currently depending on a specific CI of one's management domain.
- Service management processes are typically confined to one management domain, primarily due to lack of knowledge of external CIs in use or external domains using CIs and the lack of authority to control CIs and their corresponding resources in other management domains.
- In addition, service management process implementations vary and point-to-point process integration is cumbersome, in particular in the case of loose coupling with frequently changing dependencies between CIs of different management domains.

Single domain service management cannot address these issues and a novel approach is required.

3. Overview of the Approach

The service management approach that we are proposing – termed (Service Management) SM 2.0 – rests on a number of principles:

Distributed Configuration Management: All configuration items are discovered locally on a resource and expose a RESTful interface to management processes. We call these configuration items *Smart Configuration Items* (SCIs). SCIs can be exposed locally or also to other domains. An SCI can depend on any other SCI, in its domain or another. This dependency can be either derived in the course of a local discovery process or established manually.

Distributed Service Management: All management processes access configuration information using SCIs as the common abstraction, be they local or remote. Management Processes also expose a RESTful interface that allows other to participate in a process to the extent appropriate, e.g., allowing other domains to follow a change process or enabling other domains to trigger new incident processes related to an SCI they use.

Updates: Changes of state of SCIs or processes are being distributed using ATOM feeds [6]. Service management processes can listen to feeds according to their interests. Feeds can originate from local or remote SCIs but also from processes. Different listeners can read updates and take corresponding action pertaining to their service management processes.

Domain-Aware Service Process: Service Processes need to be aware of potential coordination with processes in different domains, responding to updates on remote processes or SCIs. This can either be achieved by implementing new service processes or wrapping existing implementations.

Figure 3 illustrates the SM 2.0 approach. It shows 2 domains A and B interacting. Each domain performs local discovery of its service infrastructure and makes configuration information available as SCIs through a RESTful interface. One SCI of the right domain can be accessed by other domains and the left domain maintains a dependency to it. Also, the service processes of the right domain are exposed as resources. Feeds are generated for changes to the state of SCIs and processes of domain B and read by listeners in domain A. These listeners can then trigger responses in service management processes. This may include participating in a process of domain B. For example, a new change process may be initiated pertaining to an SCI on which domain A depends. A feed listener for change processes identifies it as relevant and triggers a new internal change process, resulting in the participation of domain A in domain B's change process. In another case, a listener of domain A could respond to a configuration change of an SCI read in a feed by creating a new internal incident process.

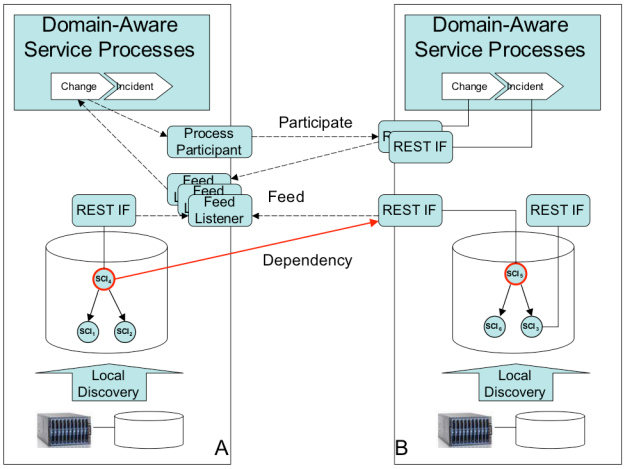

Figure 3: Overview of SM2.0

The basic tenet of the SM 2.0 approach is to use Web-based approaches where it we need to deal with cross-domain management issues due to the high level of standardization of Web-based interaction, the simplicity and flexibility of the REST-based approach and the wide availability of tooling to implement SCIs, create feeds and consume feeds and RESTfully provided information.

In the following sections we will discuss in more detail the distributed configuration management approach based on the SCI concept and distributed change management as a service process of particular relevance to service management in loosely coupled environments. This will provide a reference example for other service management processes.

4. DISTRIBUTED CONFIGURATION MANAGEMENT

The SCI distributed configuration management approach is based on discovering configuration items locally on resources and publishing the state of resources on a Web server on the resource or a Domain Configuration Server, which hosts SCI state on behalf of each resource of a domain or a part of it. Unlike traditional approaches to discovery, which mainly center around a domain-wide discovery server, in our approach each resource is configured to perform discovery locally by a discovery agent and driven by its specific needs in terms of times and frequency of scans.

In addition to state information being made available as Web resources, feeds are being created to inform about changes in configuration. Feed can be consumed by feed readers. Aggregators can combine and re-interpret feeds to provide to stakeholders of SCIs specific information about the configuration of the distributed platform according to their needs, e.g., an aggregated view of configuration changes in all domains a service infrastructure uses.

In the following subsections we will explain the architecture for configuration management and the model used to represent the configuration items and their changes.

4.1 Configuration Management Architecture Overview

Figure 4 outlines the configuration management architecture of a specific domain. Our architecture comprises two distinct sets of components:

(1) the SCI framework for configuration discovery and publishing configuration information and
(2) a feed manager for consuming and aggregating feeds.

The focus of this section is the SCI framework while we discuss the feed manager in section 4.3.

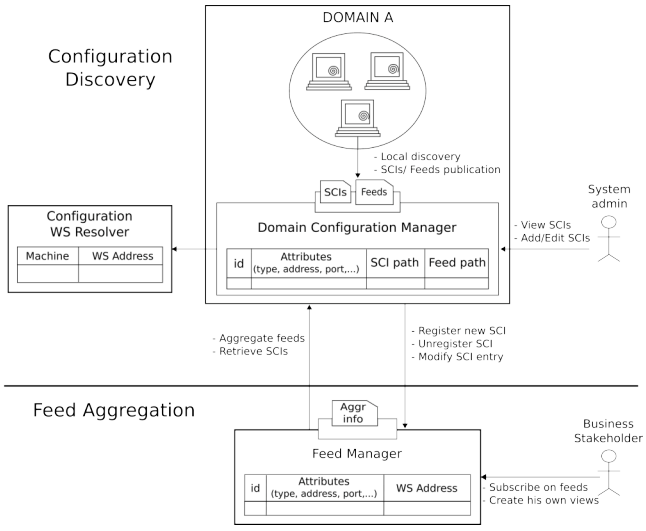

Figure 4. Configuration Management Architecture

A local discovery agent installed on each resource of the environment periodically performs configuration discovery. After local discovery, the agent generates a set of SCI files, each corresponding to a configuration item discovered on the resource. Each SCI is associated to a feed document describing its changes over time.

The generated SCIs and feeds documents are published to the Domain Configuration Manager, which is a web server authoritative for a specific domain. We may also have other scenarios in which every resource publishes its own configuration information, forming its own domain.

The Domain Configuration Manager offers RESTful services to retrieve, create, modify or delete discovered SCIs. It also provides a graphical interface system administrators can use to view all SCIs in a domain, find a local SCI, explore SCI dependencies recursively, add a new SCI or modify existing SCIs. It also enables domain administrators to add a new SCI or modify existing SCIs to represent configuration information that is not discovered automatically, e.g., a business process that is implemented based on applications running on resources.

Each Domain Configuration Manager keeps an internal registry to take trace of the available SCIs. Each SCI is associated to a unique id, a set of attributes (address, port, type, etc.) able to unambiguously identify it and two paths in the local file system pointing respectively to the location of the SCI document and the feed document containing configuration changes. Each SCI can be unambiguously identified through its URL that is constructed as follows:

http://<webServerAddress>:8080/sci?id=<id>

where *<webServerAddress>* is the address of the Domain Configuration Manager and *<id>* is the identifier of the requested SCI.

When a new SCI is created the Domain Configuration Manager adds a new entry in its internal table with a unique id associated to the new SCI, the discovered attributes and the paths to the locations of the configuration information. A new empty feed document is also created and associated to that SCI. If an configuration item is deleted in the service infrastructure the discovery agent performs a delete request to the Domain Configuration Manager, which marks the row state in its internal table as "deleted". Configuration files will be deleted after a certain time for space reasons. If an SCI is modified, the discovery agent posts a new entry in the feed document associated to that item. The possible changes are: add/delete/modify property, add/delete dependency, or add/delete a SCI pointer into a dependency.

During the discovery phase the URLs of the SCIs each configuration item depends on must be identified. These SCIs can be local or they can belong to different domains. When the dependency is local, the SCI id can be retrieved from to the local Domain Configuration Manager giving in input the attributes discovered about that SCI. The Domain Configuration Manager will search in its table the rows that have attributes matching those given in input and it will return the associated ID. If the dependency is not local it is also necessary to resolve the address of the Domain Configuration Manager authoritative for the required SCI. The Configuration WS Resolver (see Figure 4) keeps trace of the addresses of the associated Domain Configuration Managers. Typically, we will expect that many cross-domain dependencies will be entered manually.

4.2 SCI State and Feed Representation

The state of an SCI is represented as an XML document that can be retrieved at the URL of the SCI using an HTTP GET request, as outlined above.

The SCI schema is extensible to address the descriptive requirements of different configuration information domains. On the top level, each SCI has three parts:

- a set of attributes,
- a list of properties and
- a list of dependencies.

Each SCI is described by a set of predefined attributes: *URI*, it is mandatory and represents the URL that univocally identify the SCI; *type*, it is mandatory and represents the component type (DBMS, application server, database, etc.); *domain*, it is not mandatory and it represents the domain name of the machine on which the SCI resides; *description*, it is not mandatory and gives a human readable description of the SCI. Each user can add new attributes of any type, if necessary.

An SCI can have any number of properties. Each property is defined by a name and an XML value. The property name is equal to the local name of the XML tag enclosing the property value. This mechanism allows users to define their own properties that can have values compliant to an arbitrary schema. Finally, an SCI has zero or more dependencies. Each dependency is specified by a type and a list of URLs identifying SCIs on which the item depends. Extension points are provided to insert new attributes and elements describing the nature of the dependency.

In the following example we show an SCI document for a DBMS. This SCI has three properties: *host-name*, *service-port*, *instance-name*. Our SCI has one dependency, of type *HostedBy* that refers to the physical machine in which the DBMS is currently hosted.

```
<sci:SmartConfigurationItem xmlns:sci="com.ibm.watson.tlaloc.sci"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="com.ibm.watson.tlaloc.sci"
 xmlns:prop="com.ibm.watson.tlaloc.sciProperties"
 xmlns:dep="com.ibm.watson.tlaloc.sciDependency"
 type="dbms"
 uri=http://169.254.212.59:8080/sci?id=1
 domain="V25was136.mkm.can.ibm.com">
<Properties>
 <Property name="host-name">
  <prop:host-name>V25was136.mkm.can.ibm.com</prop:host-name>
 </Property>
 <Property name="service-port">
  <prop:service-port>5060</prop:service-port>
 </Property>
 <Property name="instance-name">
  <prop:instance-name>DB2Node-Instance7</prop:instance-name>
 </Property>
</Properties>
<Dependencies>
   <Dependency type="HostedBy">
    <OtherSci_id>http://169.254.212.59:8080/sci?id=2</OtherSci_id>
   </Dependency> </Dependencies>
</sci:SmartConfigurationItem>
```

The SCI document format is kept extensible to accommodate configuration information on any kind of items, hardware, software, and higher level, business items. This extensibility enables the reuse of other configuration information representation models and formats such as the DMTF Common Information Model (CIM) XML representation, which provides descriptive elements for a vast set of hardware and software configurations [22].

Besides the representation of the current SCI in the Domain Configuration Manager, the discovery process produces a feed outlining the changes to the SCI state since the previous discovery. If an SCI is modified, the discovery agent posts a new entry in the feed document associated to that item. The possible changes occurring are: add/delete/modify property, add/delete dependency, or add/delete a SCI pointer into a dependency. The description of the change is represented in the change entry content. Each change is enclosed by the element *<change>*. It is described by the following attributes: *type* that represents the kind of change happened, *xpath* that points to the modified property/dependency, *feed-uri* that is the feed url and *sci-uri* that is the SCI url. Element *<change>* has two sub-elements: *<old>* that contains the previous value of the property/dependency and *<new>* that contains the new, current value of the property or dependency. If the change is an addition or a deletion of a property or a dependency the element *<old>* or ≈ *<new>*, respectively, will be not inserted in the change description.

```
<entry>
 <title>fifth entry</title>
 <id>http://example.com/property/1236</id>
 <updated>2004-12-14T18:30:02Z</updated>
 <author>
    <name>Liliana Pasquale</name>
    <email>lpasqua@us.ibm.com</email>
    <uri>http://example.com/~lpasqua</uri>
 </author>
 <content type="*/xml">
  <!-- the property value is modified -->
  <change type="ChangePropertyValue"
     xpath="/SmartConfigurationItem/Properties/Property
      [@name='server-type' ]"
     feed-uri="http://169.254.212.59:8080/feed?id=1"
      sci-uri="http://169.254.212.59:8080/sci?id=1" >
   <pc:old>
    <sci:Property name="server-type">
      <prop:server-type>WAS v5.0</prop:server-type>
    </sci:Property>
   </pc:old>
   <pc:new>
    <sci:Property name="server-type">
      <prop:server-type>WAS v5.1</prop:server-type>
    </sci:Property>
   </pc:new>
  </change>
 </content>
</entry>
```

In the example above we show an entry describing the change of the value of the property "server-type" (the change type is *ChangePropertyValue*). The property value changed from *WAS v5.0* to *WAS v5.1*.

4.3 Feed Management and Aggregation

The objective of feed management is the creation of mashups that enable interested stakeholders to gain views that transcend the perspective of a particular domain. To this end, the information exposed by each Domain Configuration Manager is aggregated by the Feed Manager. Configuration information can be manipulated by the Feed Manager in different ways: merge all SCIs and feeds of all resources of the distributed platform, show configurations or changes for a particular type of item, or show configuration and changes relative to a specific SCI and its dependencies. The platform offers users to customize the aggregation logic using standard ATOM tools such as Yahoo Pipes [21], selecting the XML sources - configuration information or other sources online, and adding the relevant logic. For example, the feed documents describing changes can be aggregated with other sources such as the Google Charts API [22] to show the statistical distribution of changes.

The implementation of the Feed Aggregation user interface provides two kinds of information:

(1) SCIs configuration and
(2) SCIs changes.

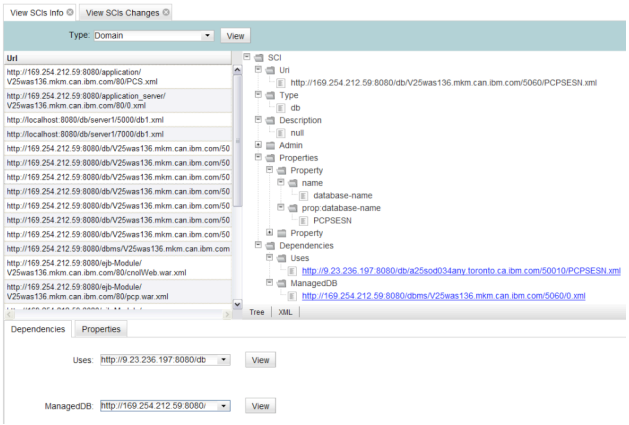

Figure 5: User interface for SCI configuration info

Figure 5 shows the user interface provided by the Feed Manager to offer information about SCI configuration. In the blue toolbar on top, the user can select the preferred visualization type: all SCIs available in the environment, all SCIs a business application relies on, all SCIs of the same type or a specific SCI. After the user selects the visualization mode, in the left side is reported a list of the requested SCIs. When the user selects one of them, the SCI document is shown in the right side. While the tabs in the bottom show the dependencies and the properties of the selected

SCI. The SCI document is visualized as a tree, as shown in Figure 5 or as XML. The SCI pointers in the dependencies are links that permit to navigate dependencies to other SCIs in any domain.

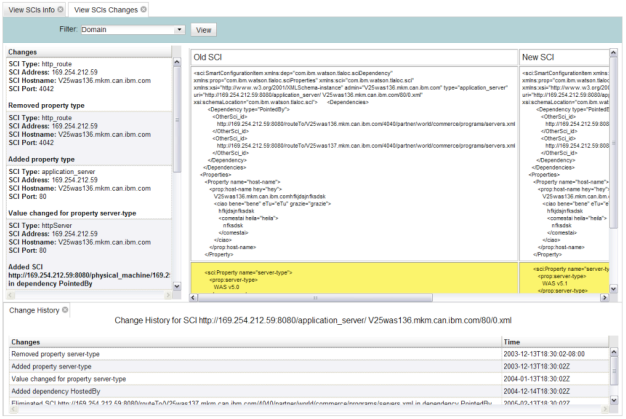

Figure 6: User interface for SCI Change info

Information about SCI changes is provided through subscriptions to Atom feeds provided by Domain Configuration Managers.

In Figure 6 we show the user interface provided by a custom feed reader. The blue toolbar allow users to filter visualized changes depending on the business application the configuration items rely on or the SCI type (e.g. only databases, or applications). If the user selects the filter mode we can see in the left pane a list of the last changes happened in the environment. When the user selects one of them, the change is shown in the right pane, highlighting the differences between the old SCI and the new SCI. In the bottom a history of the changes of the selected SCI is displayed.

To aggregate feeds coming from the whole distributed platform, the Feed Manager keeps an index of all the SCIs in an internal table: each SCI is detected through its ID, a set of attributes and the address of the Domain Configuration Manager containing its local configuration (SCI and feed documents). The Feed Manager applies filters to attributes to select feeds that need to be aggregated. For example, if a user wants to see changes that happened in all DBMS the Feed Manager will aggregate only those feeds of type *DBMS*. The Feed Manager also provides suitable interfaces to register and un-register an SCI when a new entry is created or deleted in the authoritative Domain Configuration Manager.

5. CROSS-DOMAIN CHANGE MANAGEMENT

Change management is an important aspect of IT service management. It enables the stakeholders of a system to deal with changes in a controlled manner and thus maintain consistency and remain operational. The ability to manage change is a fundamental requirement for loosely-coupled applications that are comprised of services drawn from service providers in multiple management domains. Service providers need to ensure that they can carry out necessary changes to the systems that host their service offering without breaking their clients' operations. And service consumers need an opportunity to try and adapt to changes in the services they use.

As outlined in section 4, applications built over the Web, present us with a number of challenges when it comes to managing change. Due to loose coupling and the potentially large number of applications and clients, it is impractical for a service provider to maintain up-to-date information on all clients that can be affected by a particular change. This raises the question of how to identify all clients of a service provider across administrative domains that may be impacted by a particular change without resorting to broadcasting the change proposal to everyone. Furthermore, a change process must respect the autonomy of the various management domains and accept that change management implementations vary across domains. Nevertheless, there need to be mechanisms that allow service providers and their clients to cooperate in the implementation of changes to SCIs in their domains.

How then can we enable change management in such environments? Our solution is based on a number of simple concepts whose implementation exploits standard Web 2.0 technologies. Figure 7 presents an overview of the components comprising the Change 2.0 architecture. Many of these components make use of the functionality described in the previous section.

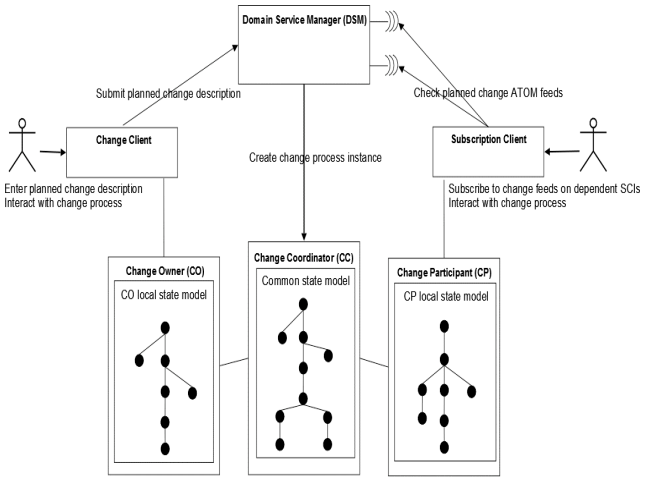

Figure 7: Change 2.0 Architecture Overview

In the following subsections we discuss the concepts behind our solution and show how these concepts have been realized in a working system.

5.1 Inversion of Responsibility

As mentioned above, without a centralized CMDB the burden of knowing which clients to inform about a particular change proposal lies with the service provider. Apart from the potentially large number of clients, a service provider may not know when a client will need to use one of its services. Furthermore, consider the case in which a modification is to a low-level configuration item, such as a DBMS. How should the link to its dependent configuration items be established? Given the impracticality of relying on a centralized CMDB, it is difficult to ensure that all relevant clients are contacted. Service providers shouldn't have to maintain detailed lists of which clients to contact for every possible change. Similarly, we want to avoid simply broadcasting change proposals to everyone. What is required is a mechanism that scales well and that can work with the reality of a loosely coupled application environment.

We overcome this issue through the inversion of responsibilities among the participants in a change. Given that service providers cannot identify the set of identified stakeholders, clients will have to know which SCIs to watch. In section 4 we have described how clients can discover the configuration items they depend on across domains. Given knowledge of their dependencies, clients are able to subscribe to changes on the SCIs relevant to their

operation. Service providers are then responsible to publish notifications about planned changes to the SCIs under their control. These notifications describe changes in sufficient detail for subscribers to carry out an impact analysis and are made available via ATOM feeds. This simple idea is the first necessary step to enable change management in service mashups.

In our architecture, there are three components that are primarily responsible for implementing inversion of responsibility. First, we have the Domain Service Manager (DSM), which is a central domain service and acts as the entry point to change management functionality. The DSM hosts a set of RSS feeds describing the latest changes to SCIs in its domain. The Change client is, in its current incarnation, a web-based UI that runs on an end user's machine. The Change client allows users to access a list of SCIs (e.g., all SCIs that belong to this application or all SCIs that represent databases under my control) through the Feed Manager and describe the planned changes to these SCIs. A description of the change is then submitted to the DSM and published in its feeds. The counterpart of the Change Client is the so-called Subscription client. Via its UI users can view the SCIs under their control and ask the Subscription client to subscribe to the change feeds of its dependencies at the DSM. Knowledge of these dependencies is the result of our local discovery as described in section 4. The Subscription client then displays new changes of the dependencies similar to a RSS reader and can also notify its users via email or SMS. Users can then view the details of any new changes, carry out an impact analysis and decide, again via functionality provided by the Subscription client, whether or not to join in the change process. By exploiting our dependency models and relying on change feeds via ATOM we remove the burden on service providers of identifying whom to notify of a planned change while furthermore avoiding the need to broadcast change notifications indiscriminately.

5.2 Change Coordination

The next question is how to implement the actual change process among a service provider and the set of clients affected by a particular change. We cannot impose the same process on everyone across administrative domains and organizations. We need to enable cooperation while not violating the autonomy of the various parties involved in a change process. Therefore, a global change process is not suitable for change management on the cloud. Instead change in such application environments needs to work like a decentralized coordination protocol. Change coordination is based on a common state model. Our state model provides for coordination at various stages of the change process and represents the least common denominator of necessary synchronization points. This leaves participants the freedom to implement the various phases of the change process at their end as they see fit.

Our state model is encapsulated in the Change Coordinator (CC) component. The CC runs the state model for each change, collects votes and status updates from participants and notifies them about transitions in the change process. Interaction with the CC is via its REST interface, which allows participants to register, submit votes, enquire about the current status of the change process, and so on. The Change Owner (CO) component and the Change Participant (CP) component represent the initiator of a change and a client affected by this change. The CO and CP implement the local part of the change process and coordinate with each other via the CC. The owner and participant have a great degree of flexibility in how they implement the CO and CP components.

Figure 8 presents an overview of our state model. Its states loosely follow the ITIL service management process [9]. During authorization participants vote whether they agree for the change to proceed. Once the CO (Change Owner) commences with the implementation, the common state reflects this and prompts all participants to carry out the necessary changes at their end. The CO and all participants then synchronize on the completion of the implementation phase in order to allow for testing the change. The result of verification can either be to undo the change or have it committed by the CO and all participants.

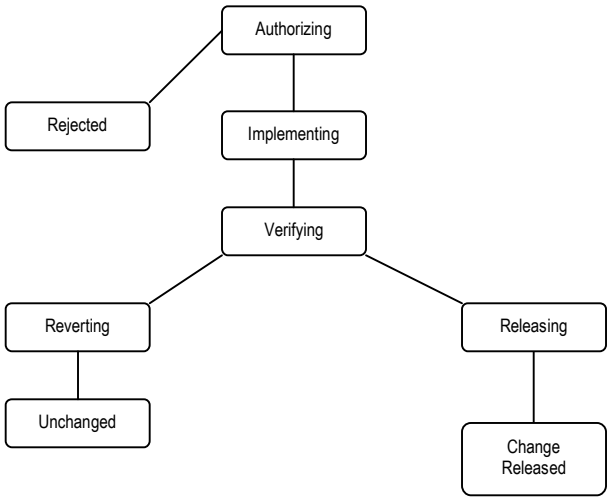

Figure 8. The common change coordination state model.

CO and CP need to be able to follow the various states of the protocol, carry out some local action and respond accordingly. For this the CC must be able to update them of the change process status via RESTful calls and receive votes as well as status updates (e.g., implementation complete, change released) from them in the same manner. However, beyond these obvious requirements to be able to follow the change protocol, there is no prescription of how the local actions should be carried out. That is, as long as participants can synchronize with the CC when required, internally they can still employ an existing manual process or some automated system. In this way we can achieve coordination while maintaining the largest possible degree of freedom on how to implement the local part of a change process.

5.3 Modes of Collaboration

Not all clients are created equal and our change process needs to take account of this. Given the cross-domain and even cross-organizational nature of change processes, we must ensure that a provider is not prevented from committing a change merely because one of the participants finds itself unable to authorize it. However, there are clients whose continued operation it is important to guarantee. A service provider will not want to go ahead with a change, if such a client reports problems during the verification of a change. The question is how a service provider can maintain control over its resources, while at the same time cooperating with those affected by its changes. We address this issue by defining different modes of collaboration that a change participant can be granted by the change owner. These collaboration modes provide for various levels of influence a participant has on the outcome of the change process. The collaboration modes we have developed so far are as follows.

- **Informative:** The change participant is notified of progress made as the change process runs (i.e. authorized, implementing), but has no influence over the change process

and does not supply any feedback. This represents the most basic form of support given to someone affected by a change. A participant is enabled to follow the change process and adapt to it, but no further cooperation can take place.

- **Consultative:** As for informative, however, a consultative participant is asked to provide feedback about the change process, such as whether it could have verified the change or how long it took to implement the necessary changes at its end. This differs from the informative mode in that it enables a service provider to collect information about how its clients cope with its changes. It can be useful to aggregate such information over time.
- **Co-Authorizing:** As above, but in this case the change participant can influence the change process through its authorization vote. The Change Coordinator will ask the owner and all other participants to abort the change, if a co-authorizing participant votes to reject in the authorization phase.
- **Co-Verifying:** As above, but in addition the change participant's vote during the verification phase is taken into account. If a co-verifying change participant indicates that it couldn't verify a change (i.e. adaptation to the change at its end wasn't successful), then the Change Coordinator will ask the owner and all other participants to revert the changes. A vote to reject a change can contain additional information explaining the reasons behind this decision.

The Change Coordinator can be configured to grant a particular collaboration mode to certain groups of clients. In addition, Change owners can be notified of the collaboration mode a particular participant requests during registration in a change and decide whether or not to grant it. During the change process, the Change Coordinator will react to incoming messages from participants according to the collaboration mode granted to them. The various modes are not mutually exclusive within a change process.

The concept of collaboration modes affords a service provider control over its resources while at the same time reflecting the various degrees of influence a provider may want to grant to certain clients from different domains. Furthermore, collaboration modes define a framework for cooperation in a change process ranging from simply providing information about the progress, over collecting feedback and finally to having the ability to abort the change process, if these changes would break the service of an affected client.

Change management is an important activity in loosely-coupled applications that consist of compositions of services available through a marketplace of service providers. In this section we have identified the key issues that complicate change management in complex, cross-domain application environments, such as Web 2.0 and cloud computing. We have shown how these issues can be addressed through a few simple concepts, which have been implemented using standard Web 2.0 technologies, such as ATOM feeds and RESTful interfaces. This should simplify the integration of Change 2.0 into a wide array of systems. Having overcome the most pressing technical issues in enabling change management for these environments, it will be interesting to investigate which additional challenges will become apparent from applying our architecture/solution on some real world cases.

6. IMPLEMENTATION

The SM 2.0 Framework for distributed, cross-domain management was implemented based on WebSphere Smash, a development and runtime environment for RESTful services, mash-ups and AJAX-type Web applications [8].

As explained above, the distributed configuration management is comprised by several components: the local configuration discovery agent, the Domain Configuration Manager and the Feed Manager. The configuration discovery performed by the local agent is based on Galapagos [7]. Galapagos is a model-driven approach combining models of software components with a distributed crawling (graph traversal) algorithm to discover end-to-end, multi-tier dependencies between application and data in a distributed system. The local agent uses the results produced by the discovery performed by Galapagos to generate the SCIs XML representations. The Domain Configuration Manager and the Feed Manager are implemented as WebSphere Smash applications. Both expose REST interfaces as described for the consumption of SCIs and feeds and the definition of new SCIs. The graphical user interfaces provided by the local Domain Configuration Managers and the Feed Manager are based on Dojo [9], an open-source JavaScript toolkit for building Ajax web applications.

As a representative of cross-domain service integration, the Change 2.0 approach was likewise implemented using WebSphere Smash, building on the distributed configuration framework. All components are Smash applications, Change Coordinator as well as the Change Owner and Change Participant components and their user interfaces.

7. RELATED WORK

Several efforts in the area of distributed systems management using services have been described and standardized. The approach of exposing a resource's properties to describe its details is proposed in standards such as Web Services Distributed Management (WDSM) [11][12], and Web Services for Management (WS-Management) [13]. To define the actual components, WS-Management endorses the use of WS-CIM [7], and WSDM requires key tags to specify attributes about each property. Both standards allow for extensions to embed custom definitions. Each standard proposes a different set of service interfaces to access these definitions. WSDM recommends the use of the Web Services Resource Framework (WSRF) [10], and WS-Management recommends WS-Target [17]. WSRF originated as part of the Open Grid Services Architecture [20], which identified the need for special treatment of management in a cross-domain environment [14].

In contrast, SCIs have a very simple and extensible schema to describe properties and expose them using a RESTful interface. To manage events WSDM uses WS-BaseNotification [18] and WS-Management uses WS-Eventing [19]. Both standard offer a publish/subscribe mechanism to send and receive events. These protocols are not as widely spread and pose a greater obstacle to adoption than the proposed ATOM/RSS feed approach of the SM 2.0 framework.

Dependencies are key within the SCI approach and the management processes which are based on it. We need to be able to trace SCIs on which we depend and receive notification when they are about to change. In WSDM, the concept of Relationships is presented to express any type of relation, including dependencies. We claim that dependencies are all we need to maintain and we can do the majority locally. It is worth noting that WS-M does not cover the aspect of maintaining relationships or dependencies. Also, within the Grid world, dependencies are not paramount, locating

services is more important, yet via a peer to peer approach all resources can be identified and indexed.

Fundamental to a SCI is its local awareness, it allows identifying its dependencies and at the same time detecting changes and notifying them via feeds. We proposed a mechanism that provides discovery capabilities from the genesis of a host, jointly with a comparison mechanism that is able to detect when components are provisioned, modified or destroyed. In WS-Management, an identification service, namely Identify, allows to discover any component that supports such interface, in WSDM, relationships may be used to traverse and find other components, yet there is no prescribed method to first establish those relationships.

While there is initial work on CMDB federation whitepaper [3], it is not applicable to a dynamically changing environment as it is found in today's SaaS and Cloud usage patterns.

Significant work has been conducted on cross-domain process integration in general [23] and dynamic integration in particular [24]. However, many approaches rely on detailed, message-oriented integration of bilateral parties, in pre-designed processes such as expressed with WS-BPEL [25], or rely on contracts to specify the details of a relationship between interaction partners. None of these approaches applies to the large scale and dynamic environment of the SM 2.0 framework.

8. SUMMARY AND CONCLUSION

In this paper we discussed issues and challenges posed to service management by distributed, Web-based applications spread over multi organizations and proposed the SM 2.0 approach to overcome these issues.

Environments that make use of Cloud infrastructure or platform services, the integration of SaaS in an organization's service infrastructure, as well as the use of Web services in mashups lead to distributed ownership of resources and corresponding distributed service management responsibility. Current service management approaches are primarily based on the availability of a central CMDB as a repository of configuration information on which all service management processes are based. Moreover, service management processes are aimed at being conducted in a centralized way and do not address interaction with other organizations' service management processes. However, this is not viable in the case of cross-domain integration of service infrastructure.

The proposed SM 2.0 approach addresses these issues based on a distributed approach to configuration management and cross-domain process integration. Configuration items are discovered locally and exposed using a RESTful interface as SCIs, which provide an abstraction of local and remote configuration information for management processes. Also, service management processes expose a restful interface that enables other domains to participate. We detailed the approach to process integration using the change management process as example. The SCI framework, the feed aggregator and the change coordinator have been implemented on the basis of WebSphere Smash.

While some prior work addresses issues of CMDB federation or distribution of management interfaces on resources, there is no approach, to our knowledge, that addresses the issue of loose coupling and its ensuing dynamics and the integration of management services exposed as a RESTful entity.

The SM 2.0 approach is aimed at a cross-organizational scenario. However, it is also applicable to large enterprises that have different domains of responsibility between the IT organizations and the lines of business.

In our next steps, we will work on implementing other management processes on the basis of the SM 2.0 approach and devising novel ways of aggregating feeds from different SCIs and processes for advanced analytics of service management behavior. In addition, we will continue to evaluate SM 2.0 against existing centralized approaches, ease of integration by consumers of the information produced by SM 2.0, cost and frequency of aggregations and quality of the data that is available.

9. Acknowledgements

The research leading to these results is partially supported by the European Community's Seventh Framework Programme (FP7/2001-2013) under grant agreement n° 215605.

10. REFERENCES

[1] Google AppEngine. http://code.google.com/appengine

[2] Force Platform. http://salesforce.com/platform

[3] D. Clark et al.: The Federated CMDB Vision: A Joint White Paper from BMC, CA, Fujitsu, HP, IBM, and Microsoft, Version 1.0. 2007. (http://www.cmdbf.org/CMDB-Federation-white-paper-vision-v1.0.pdf)

[4] R. Fielding: Architectural Styles and the Design of Network-based Software Architectures. Dissertation, University of California, Irvine. 2000.

[5] Service Transition, Information Technology Infrastructure Library, (May 2007).

[6] The Atom Syndication Format. *http://www.ietf.org/rfc/rfc4287.txt*

[7] K. Magoutis, M. Devarakonda, N. Joukov, N. Vogl: Galapagos: Model-driven discovery of end-to-end application-storage relationship in distributed systems. IBM Journal of Research and Development, June 6, 2008.

[8] ProjectZero. http://www.projectzero.org/

[9] Dojo. http://dojotoolkit.org/

[10] Web Services Resource Framework (WSRF) Primer v1.2. OASIS Committee Draft 02 - 23 May 2006.

[11] Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.1 OASIS Standard, 01 August 2006

[12] Web Services Distributed Management: Management Using Web Services (WSDM- MUWS 1.1) Part 1, 2 OASIS Standard, 01 August 2006

[13] Web Services for Management (WS-Management) Specification Document Number: DSP0226 DMTF 2008-02-12 Version: 1.0.0

[14] A. Iamnitchi and I. Foster: On Fully Decentralized Resource Discovery in Grid Environments. International Workshop on Grid Computing, Denver, CO, November 2001.

[15] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman: Grid Information Services for Distributed Resource Sharing, In Proceedings 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), August 2001.

[16] Web Services Common Information Model (WS-CIM) DMTF.

[17] Web Services Transfer (WS-Transfer) W3C Member Submission 27, September 2006.

[18] Web Services Base Notification 1.3 (WS-BaseNotification) OASIS Standard, 1 October 2006.

[19] Web Services Eventing (WS-Eventing) W3C Member Submission 15, March 2006.

[20] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, J. Von Reich. The Open Grid Services Architecture, Version 1.0. Informational Document, Global Grid Forum (GGF), January 29, 2005.

[21] Yahoo pipes. http://pipes.yahoo.com/pipes/

[22] Distributed Management Task Force: Specification for the Representation of CIM in XML. Version 2.2, January 9, 2007.

[23] C. Bussler. B2B Integration: Concepts and Architecture. Springer-Verlag, 2003.

[24] P. Grefen, K. Aberer, H. Ludwig, and Y. Hoffner. Cross-organizational workflow management for service outsourcing in dynamic virtual enterprises. IEEE Data Engineering Bulleting, (24)1, 2002.

[25] OASIS. Web Services Business Process Execution Language Version 2.0, 2007.