RC24796 (W0905-036) May 13, 2009
Computer Science

IBM Research Report

Distributed Cross-Domain Change Management

Bruno Wassermann
University College London
Department of Computer Science
London, WC1E 6BT
UK

Heiko Ludwig, Jim Laredo, Kamal Bhattacharya
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
USA

Liliana Pasquale
Politecnico di Milano
via Golgi, 40
20133 Milano
Italy

Distributed Cross-Domain Change Management

Bruno Wassermann
University College London
Dept. of Computer Science
London, WC1E 6BT, UK
b.wassermann@cs.ucl.ac.uk

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

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

Abstract

Distributed systems increasingly span organizational boundaries and, with this, system and service management domains. Web services are the primary means of exposing services to clients, be it in electronic commerce, Software-as-a-Service (SaaS) or on cloud platforms and are being used and integrated with customer-managed applications as well as in complex mashups. Maturing cross-domain relationships and an increase in loose coupling and ad-hocness makes managing configuration changes, e.g., changes in interfaces or endpoints, increasingly relevant. Traditional service management processes within organizations, in particular change management, relies on a central configuration management database (CMDB) to assess the impact a change has on other components of the system. However, this approach does not work in a cross-domain environment, due to the lack of a central CMDB, centralized management processes, and knowledge by service providers which clients depends on their respective services. This paper proposes the Change 2.0 approach to cross-domain change management based on an inversion of responsibility for impact assessment and the facilitation of cross-domain service process integration. We present the requirements imposed by cross-domain change management, the Change 2.0 architecture, and a brief evaluation of its benefits.

1. Introduction

The increasing popularity of Software-as-a-Service (SaaS), Cloud computing, and the use of mashups to create novel applications on the basis of services and data from different sources entails an increase in applications that rely on infrastructure spanning multiple organizations, and with that, multiple management domains. In this context, complex dependency structures can arise. For example, an instance of a SaaS, such as Salesforce.com, may need to access a customer's backend system, such as a customer database, through a Web service interface. Today's Web service platforms not only enable complex service invocation dependencies, but also the rapid and dynamic establishment of these relationships by taking advantage of loose coupling capabilities. As a result, these cross-domain dependency relationships are subject to frequent change. The problem this poses is that changes to a service owned and managed by one domain have the potential to affect the operation of applications in other domains depending on this service. A simple change to the interface of some Web service can thus disrupt an application in another domain, which now relies on an effectively deprecated service definition.

Evolution of interfaces, endpoints and properties is a normal occurrence in the life-cycle of a service and organizations address it with the help of a *change process*. A change process helps transitioning a system configuration from its current state to a new state while minimizing its impact on service availability and taking into account the side effects that a configuration change entails [11]. The processes typically involve a number of employees of an organization that work on designing a proposed change, asses its impact on other configuration items, i.e. infrastructure components, applications and services, and then implement the change in a change window, a time interval that is convenient, e.g., at night or on weekends. Change management systems, which are available from many vendors, help drive the process and involve the necessary stakeholders.

Current change management processes and systems suffer from two key shortcomings when it comes to cross-domain applications. (1) First, they typically rely on a centrally managed *configuration management database (CMDB)* to store information on a system's configuration, its *configuration items (CIs)*. Such CMBDs are the basis on which service management processes within a management domain run. They facilitate activities such as determining the impact of a proposed change on other configuration items. (2) The second issue is that service management processes are typically not integrated between service providers and users. If a service provider knows its user base, it may send a notice of a pending change per email or post it on a Web site. This is error prone and will often lead to delays if the relationship between a service provider and its clients is not institutionalized. In many cases, changes are only noticed after their implementation leading to outages of their using applications. The distribution of management responsibility for cross-domain applications requires a decentralized approach to change management.

This paper proposes a novel approach to change management across management domain boundaries. This Change 2.0 approach addresses the issues of how to enable the various parties affected by a change to notice relevant changes and asses their impact and furthermore implements a change coordination protocol to enable the various parties to cooperate in the implementation and testing of changes.

The next section discusses the issues of cross-domain change management in more depth, derives a set of specific requirements that need to be addressed, and introduces an example on which we evaluate Change 2.0. Section 3 presents the key concepts underlying our approach to cross-domain change management. In section 4 we discuss the REST-based architecture of the current Change 2.0 prototype, before we discuss a proof-of-conept implementation and evaluate its benefits in section 5. Finally, we discuss related work and summarize our conclusions from this work.

2. Issues of cross-domain applications change management

In a single management domain service management today is mostly conducted according to various sources of best practices, e.g., the IT Infrastructure Library (ITIL) [11] and the Control OBjectives for Information and related Technologies (COBIT) [19]. Assets and CIs are the information on which service management processes, such as change management, are based. Depending on the intended use of a CI, its relevant properties may include IP addresses of virtual machines, Web service endpoint URLs, WSDLs associated with an endpoint, information about service capacity, policy information, etc. The Distributed Management Task Force (DMTF) publishes a Common Information Model that proposes a standard set of CI types to which many vendors adhere [6]. An important aspect of CIs in a CMDB is the notion of dependency of one CI on another, e.g., a Web application being dependent on its application server. This is important for problem root cause analysis or impact assessment.

ITIL also identifies a number of service processes such as incident management (often called ticket management), problem management, and change management. 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 CIs and changes in its configuration by searching for and analyzing systems on the network of a service infrastructure. Changes to CIs can also be driven by a change management process updating the corresponding entries in a CMBD.

A change process is typically conducted in a number of steps, which may vary depending on the best practice used, the specific process of an organization and the service management system that the organization implemented. However, we will find a set of common steps in most change process implementations:

Change Design A proposed change is designed, i.e. it is specified which set of CIs will change, e.g., by installing a new version of a an application or a patch. This typically also includes a rollback plan in case of failure.

Impact Assessment It is investigated which CIs beyond the ones in the change will be impacted and remedial action will be planned, either in the course of the same change or separate ones. The key source of information for this step are the CI dependencies as found in the CMDB.

Change scheduling A suitable point in time is set to execute the change e.g., in a change window on a weekend [18].

Change implementation The actual implementation of the change, e.g., the installation of an upgrade.

Change verification and release Testing the change and, if successful, putting it in operation.

The main motivation behind the implementation of such a change management process by an IT organization is to minimize the down time and cost due to the side effects of necessary changes to one's infrastructure. In a single management domain, it can be assumed that all assets and CIs relevant for the management processes can be found in a centralized CMDB. Furthermore, these CIs can be discovered and read within a single management domain. However, in a situation where Web services are accessed across organizational boundaries in a potentially dynamic manner, we can neither assume a central CMDB shared by all participants nor an integrated change process agreed upon by everyone.

We will refer to the following example throughout this paper: A Start Up online shop relies on a payment processor Enterprise to charge its customers while the shop application itself is implemented as a cloud application by a cloud platform provider. The payment processor in turn uses a storage service provider for storing transactions data. Both, payment service and storage service expose a Web service interface. All service providers have a large number of customers.

Figure 1 outlines the corresponding configuration. The relevant CI properties for these dependencies are what we would normally find in a WSDL, such as operation signatures, service version, authentication mechanism, endpoint addresses, XML schemas, and so on. The rounded boxes represent different management domains and the circles are CIs in these domains. The edges represent dependencies between CIs. The bold circles represent CIs on which other management domains depend for the services they offer to their clients. The relevant CI properties for these dependencies are the operation signatures, service version, storage size limit, authentication mechanism, endpoint addresses, XML schemas, and so on. These are properties you would typically find in a WSDL and additional information about the service such as the storage capacity associated with the storage service subscription.

In the course of the service life-cycle, service providers may need to modify some of the CIs in their domain. For example, it may become necessary to change the authentication method of the payment service to a longer key length to comply with banking regulation. This can negatively impact clients in case changes to the client code invoking the Web service are needed and may impair the online shop, if not implemented in time. Given the example in Figure 1, we see that there are further levels of CI dependencies, which may cause a ripple effect along the chain of dependencies leading to large overall outage costs.

In summary, as we have started to build applications as compositions of services from various management domains, we have increased the likelihood of outages due to changes to some CI somewhere along the chain of cross-organizational dependencies. Yet, at the same time no efficient mechanisms are available to manage change across domains. Cross-domain change management is an important ingredient in the reduction of outage costs and clients of Web services will regard this capability as an important value-add. The distribution of management responsibility of Web-based applications requires a decentralized approach to service management that takes into account the distribution of configuration information and facilitates the execution of management processes across organizational boundaries while maintaining each organization's autonomy. Enabling change management for cross-organizational applications needs to address the following issues:

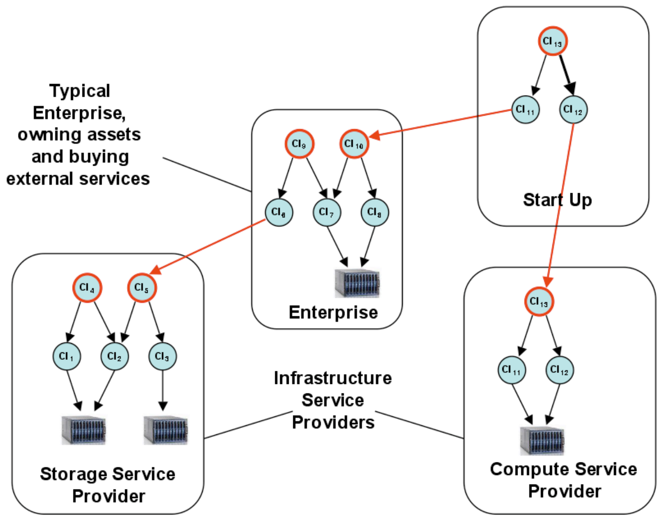

Figure 1. Overview of online shop example configuration.

- It is difficult to discover assets and CIs outside one's own management domains due to the lack of scope and access to the resources of other domains.
- It is difficult to keep track of which management domains are using and currently depending on a specific CI of one's management domain, in particular in dynamic and loosly-coupled environments.
- Service management processes are typically confined to one management domain, primarily due to lack of knowledge of and access to external CIs.
- Service management process implementations vary and point-to-point process integration is expensive.

In the next sections we outline how the Change 2.0 approach addresses these issues.

3. Change 2.0 concepts

Change 2.0 is based on a number of basic principles to overcome the issues raised above. First, we address the shortcomings of centralized approaches through the inversion of responsibility. We let clients maintain knowledge of the CIs they depend on and obtain notifications about relevant changes. Second, we have developed a decentralized change coordination protocol based on a common state model. This enables participants to cooperate on the implementation of changes while respecting their autonomy. Finally, we allow initiators of a change to fine tune the degree of influence participants have over the outcome of a change process.

3.1. Inversion of responsibility

Service providers want to ensure that all clients potentially affected by a change are notified about it so that they have an opportunity to adapt to it. However, as discussed above, inter-domain applications complicate achievement of this requirement in several ways. First, it is impractical to maintain a centralized global public CMDB and CMDB federation presents us with trust and scalability issues. Second, in a loosely-coupled environment it can be difficult to establish when and whether a particular client will use a service. And finally, there are cases where it is not obvious how to establish a link from a specific change to a set of potentially affected clients. For example, consider a change to a low-level configuration item, such as a DBMS underlying a public storage Web service, e.g., the change to the query language offered by the storage service. How could a service provider determine all of its dependent CIs across domains, e.g., those using deprecated features of the query language of a public service? We need a mechanism that doesn't require a central CMDB, that nevertheless removes the burden of identifying which clients to contact form service providers, and that scales well.

Our solution to these issues is to invert the responsibilities among clients and service providers in the inter-domain use of Web services. The two main aspects to this idea are the discovery of configuration information and notifications about changes. In the absence of a central CMDB, each client is responsible for discovering and maintaining all those CIs it depends on. We use a decentralized configuration discovery mechanism for this purpose, in which each resource carries out a local discovery and publishes CIs relevant for external clients as Web accessable resources. Furthermore, given knowledge of their dependencies, clients are able to subscribe to changes on the CIs relevant to their operation. Service providers are then responsible to publish notifications about planned changes to the CIs under their control. These notifications should describe changes in sufficient detail for subscribers to carry out their own impact analysis. The delivery mechanism for notifications must scale to a large number of subscribers and work across the Internet. The problem is thus reduced in complexity, as each client maintains its own set of dependencies and providers can remain unaware of their clients for the purpose of change management.

3.2. Decentralized change coordination

Successful adaptation to changes often requires cooperation among a service provider and the set of clients affected by a particular change. However, we cannot expect everyone to adhere to and implement the same change process. Point-to-point integration is not feasible on a large scale and therefore a more general solution would be desirable. In particular, such a solution must provide for cooperation in implementing a change without violating the autonomy of the various parties involved in a change process.

Instead of a global change process, change should be a decentralized coordination mechanism coordinating independent processes. Coordination protocols are often applied in situations where a dynamic number of participants are being coordinated, e.g., in transactions executing a 2 phase commit protocol orchestrated by a transaction manager [9], [12]. Coordination means in this case distribution of state updates to the participants involved and advancing the state of the coordination according to a state model. These models are typically very simple, e.g., the 2 phases of the 2 phase commit.

We apply this general approach on coordination to the issue of cross-domain change management. In Change 2.0, coordination is based on a common state model representing the least common denominator of the states a distributed change processes assumes, corresponding to the phases or steps of a change process. Participants are free to implement the various phases of a change process as required at their end but are able to synchronize on the progress of the common change process.

Figure 2 shows the common state model of the Change 2.0 approach. The initial state is Authorization during which participants join and have an opportunity to voice their opinion whether or not to go ahead with the change. If a participant votes to reject the change, the state of the change process will reflect this (i.e. Rejected) and all participants are notified of the decision. Otherwise, if no one rejects the change by the planned start time of the change, the change owner is free to indicate when it has started with the implementation and the common state will reflect this prompting all participants to carry out the necessary changes at their end. Once the service provider has completed its implementation of the change and all participants have been adapted accordingly the common state model transitions to the verification phase. This allows for testing the changes and can either result in undoing the service change or putting the new version into production. These states loosely follow the ITIL service transition process [11]

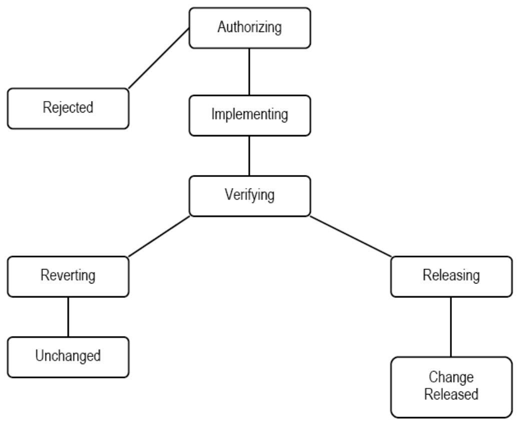

Figure 2. The Change 2.0 common state model.

and represent the common states necessary to allow participants to synchronize on implementation and testing of changes.

3.3. Modes of collaboration

Given the cross-domain and cross-organizational nature of the change processes considered here, it is important to ensure that service providers can maintain control over their resources and still cooperate with those affected by its changes. For this purpose we have introduced the concept of collaboration modes. These represent different levels of influence a participant is granted by the change owner over the outcome of the change process. We have defined the following modes of collaboration

- 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. 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 affords collection of information about how clients adapted to a change and it may 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. A co-authorizing participant can vote to reject the change during the authorization phase.
- Co-Verifying: As above, but in addition the change participant's vote during the verification phase is taken into account. A co-verifying participant whose adaptation to a change failed can thus cause the owner and all other participants to revert the changes.

Different levels of cooperation can be granted to different service users according to their importance. While high-level involvement is often desired, it limits the service provider. In cases of a large user base, providers will opt for the informative or consultative mode of collaboration, which scales well to large numbers.

4. Change 2.0 Architecture

The issues discussed above point to a number of key requirements for cross-domain change management. It must be possible to carry out change notifications in a scalable manner, to allow clients to easily integrate their management systems with the change coordination protocol, and to respect the autonomy of the various domains involved in a change. The Change 2.0 change process includes two major steps: the dissemination of information on impending changes and the coordination of the change process itself.

4.1. Change dissemination

A prerequisite step of disseminating information on impending changes to CIs is the access to CIs pertaining to a Web service by its clients. This requires the discovery of configuration information and its publication to clients in a convenient way, as RESTful Web resources. As alluded to by inversion of responsibility, instead of relying on a centralized process for this, each resource or domain needs to perform a discovery of its CIs through locally installed agents. The dependency discovery approach creates a set of so-called *Smart Configuration Items (SCI)*. Each SCI contains the properties of the configuration items it represents and a set of dependencies on other SCIs, which can be in the same or in some remote domain. This approach for local discovery and publication of SCIs has been discussed in detail in [13]. In the Change 2.0 architecture, the *Feed Manager (FM)* is responsible for maintaining a registry of all SCIs that a domain has discovered locally and makes this information available to clients with relevant dependencies. Each domain is thus responsible to maintain its own set of dependencies.

Subscriptions and notifications of planned changes are mainly dealt with by four components in our architecture as illustrated in Figure 3.

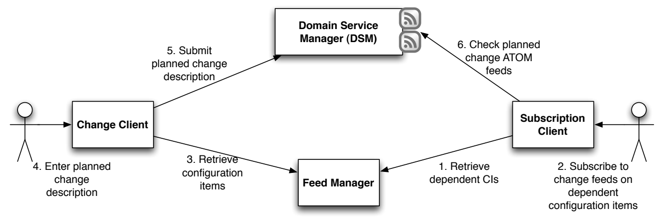
Domain Service Manager (DSM)
5. Submit planned change description
6. Check planned change ATOM feeds
Change Client
Subscription Client
4. Enter planned change description
3. Retrieve configuration items
Feed Manager
1. Retrieve dependent CIs
2. Subscribe to change feeds on dependent configuration items

Figure 3. Subscription and notification of changes.

Having identified the SCIs they depend on, clients subscribe to information on planned changes pertaining to these SCIs using a *subscription client*. Planned changes are made available as a feed at the *Domain Service Manager (DSM)*. The DSM is a central domain service and acts as the entry point to change management functionality for a particular domain. It serves feeds that describe the latest planned changes to SCIs in its domain to interested subscribers (i.e. all those resources that discovered a changing SCI as one of their dependencies). Service clients use the subscription client to monitor changes to their dependencies. The subscription client polls the relevant DSM instances in different domains for changes to these dependencies and notifies its users if a change is encountered that may potentially impact a service client. Users can then retrieve all necessary information to carry out an impact analysis for the given change and take the appropriate action such as conducting their own change process.

On the service provider's side, a *change client* is used to connect to the DSM and submit a newly planned change, including the set of SCIs to which it is germane. The change feed for each of the affected SCIs is amended with the planned change, to be retrieved by interested service clients. The change feed describes the planned changes to SCI properties, e.g., the change of a WSDL or the update of the DBMS underlying the storage service.

4.2. Change process coordination

The three primary components that implement change coordination are the Change Coordinator (CC), the Change Owner (CO) and the Change Participant (CP). The CC encapsulates our common state model and enables owners and participants to coordinate the implementation of a change among each other. The latter two represent the initiator of a change, the service provider, and a client affected by this change, respectively. In our current architecture the CC resides in the domain initiating the change, but it could equally well be hosted as a service by a third party.

Figure 4 shows how the various components of the

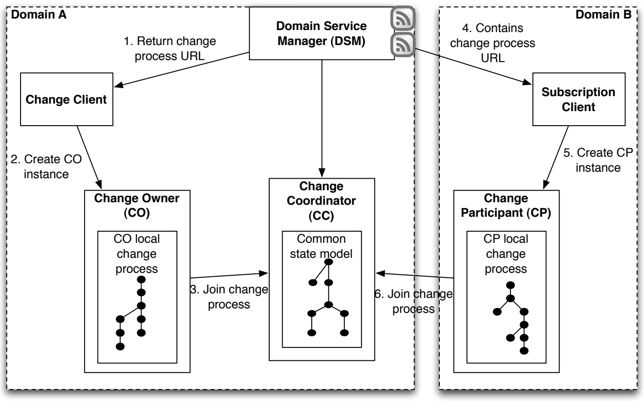

Figure 4. Creation of a new change process.

Change 2.0 architecture working together. When a change client submits a new planned change to the DSM, the DSM not only reflects this in its change feeds, but it also causes a new change instance to be created at the CC, which we refer to as the *change context*. The DSM then informs the change client of the newly created change context by returning its URL. Furthermore, it adds this URL to the corresponding change feed entry. The change client can then create a Change Owner instance, which will represent the service provider of the change during the change process. Similarly, the subscription client creates a Change Participant instance, which then joins the change process at the CC. The CO and CCs can be integrated to the proprietary, domain-internal change management systems of their respective organizations.

On request by the DSM the CC initiates a change context for each change and then transitions through the state model by collecting votes and status updates from the change owner and participants. Each change context at the CC implements a state machine representing our common state model. The state machine implementing the state model inspects all incoming requests and determines whether or not to transition to the next state. For example, it will transition to verifying, once the CO and all CPs indicate that they have completed the implementation or adaptation to the change. Upon a state transition, the CC updates participants of this and then awaits their reaction. For example, the CC informs everyone that the Verifying phase has been reached and thus indicates that it now awaits votes from the owner and participants whether or not they verify the change.

Once the change has progressed to a terminal state, Unchanged or Change Released, the change coordination is completed and the change context in the CC can be archived or dismantled. The CO and CP instances in the service provider and the clients can be dissolved and the domain-internal change processes in all participants can completed.

4.3 Interfaces

Facilitating change management across domain boundaries benefits from using existing standards of interoperability and requiring the least additional standardization, thus enabling adopters to use existing systems to deal with Web services as far as possible. The Change 2.0 approach to interfaces is is based on Web 2.0 technologies to facilitate integration and interoperability. All components expose their functionality via RESTful interfaces [8]. We rely on ATOM feeds [2] for the notification of planned changes, which lends itself well a the large number of subscribers. The various data types used by the Change 2.0 components (e.g., change feeds and change protocol messages) are published as XML Schema Definitions.

Table 1. Excerpt of the CC's REST interface.

Resource	GET	POST	DELETE
/chgctxs/id/participants	List participants	Register participant	Participant withdraws
/.../id/status	Protocol status	Unused	Unused
/.../votes/authorization	List submitted votes	Submit authorization vote	Unused

An example of the REST interface as defined by the CC is shown in table '1. For each change, there is a uniquely identifiable change context through which a number of resources are organized. These resources allow participants to register, submit votes and inquire about the current status of an ongoing change process. The payload of requests to and responses from the CC are defined as data types published by the CC as an XSD. Using RESTful interfaces and ATOM feeds enables service providers and service client organizations to use existing tools such as ATOM readers by Google or various e-mail systems to follow changes or to implement COs and CCs based on widely available toolkits to deal with REST, ATOM feeds and the like.

As stated above, it is an important requirement to respect the autonomy of participants in a change process. There are a number of requirements on the implementation of a CO and CP. First, they need to understand the data types published by the CC and the feed format used by the DSM. There are a handful of status messages (e.g., implementation complete, verifying) they need to understand. Second, COs and CPs need to offer some URL on which they accept status updates from the CC. Finally, they must be able to inform the CC of their current progress and of their votes at certain points in the change process. It is important to note that beyond these requirements participants are free to implement the change process at their end in whatever way they see fit. Integration thus does not impose more than the bare minimum and the interface standards used keep the implementation burden low.

5. Implementation and evaluation

We implemented a proof of concept of all components involved in Change 2.0. We use WebSphere sMash as a lightweight Web server supporting RESTful implementations. We use Dojo [1] to implement the change and subscription client end user components. Since all interfaces are REST, it is easy to also read status in a regular Web browser or follow feeds using a feed reader, which enabled us to make use of many popular tools such as Google Reader for subscriptions and reduce adoption cost, both in terms of implementing the solution as well as in training costs.

While a large-scale trial of Change 2.0 is still outstanding, we are able to start sizing its benefits. The main global benefit is the reduction in change-related outage costs of service clients. The reduction for each client depends on the collaboration mode of the client. Co-authoring and co-verifying clients should see a large reduction of outage cost, close to zero in the latter case. The benefit of clients in an informative and consultative mode depends on the extent to which they are affected by the change and whether the time window allocated for the change is sufficient for their change implementation and verification. It is difficult to size the improvement over email or Web-based updates on service changes. We are not aware of any studies on the perception rate of these announcements in organizations though, anecdotally, it is low. The benefit for service providers is the superior service they can offer to clients, compared to peer services that don't offer change management support.

6. Related Work

Much work has been done in the area of distributed systems management to expose configuration information to the public and on the Web. Both standards, Web Services Distributed Management (WDSM), and Web Services for Management (WS-M) propose a Web services interface to access this data [15], [7], the latter of which proposing an XML rendition of CIM [6] as a model for the CI content. While WSDM has a concept of relationships, which could be amended to represent dependencies, WS-M lacks such an explicit concept which is crucial for tracking cross-organizational changes. To distribute information, WSDM uses WS-BaseNotification [16] and WS-M uses WS-Eventing [4]. Both standards offer a publish/subscribe mechanism to send and receive events. An ATOM/RSS feed approach, however, can be used with any RSS reader and rely on its subscription and filtering capabilities.

CMDB federation enables accessing configuration information held in different CMDBs [5]. However, CMDB federation requires the explicit establishment of the federation relationship between interested parties 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. Therefore, this approach is not applicable to dynamically changing environments as it can be found in today's Web services usage patterns.

Significant work has been conducted on cross-domain process integration in general [3] and dynamic integration in particular [10]. However, many approaches rely on detailed, message-oriented integration of bilateral parties, in predesigned processes such as expressed with WS-BPEL [17], or rely on contracts to specify the details of a relationship between interaction partners. None of these approach does applies to the large scale and dynamic environment of our change issue in the same way than a coordination-based approach, on which we based our work [12].

Initial work on automatic service invocation adaptation remains confined to specific scenarios of change [14].

7. Conclusions

As Web service use and integration across the Internet in a loosely-coupled manner proliferate, managing change in a controlled way becomes increasingly important. Otherwise, we are facing a situation in which regular maintenance and evolution of services leaves behind a trail of disruption and service outages on a regular basis.

We propose the Change 2.0 approach to address this issue. In this approach, the initiator of a change is relieved of having to identify the set of potentially affected clients. Instead, each domain maintains its dependencies and change owners publish change notifications. The use of ATOM feeds enables the publication of such notifications to a potentially large number of subscribers across the Internet. We rely on widely adopted Web 2.0 technologies to facilitate integration and interoperation. We avoid imposing a single change process through our coordination-based approach that imposes a common state model, which encapsulates the minimum number of synchronization points necessary to achieve meaningful cooperation. Participants are left with ample freedom to implement a change process at their end, or use the change system they have in place. Finally, we enable service providers to maintain fine-grained control over their resources through various modes of collaboration. To our knowledge, this solution is the first to address the key issues of cross-domain change management that we have outlined at the beginning of this paper.

The next stage of our work is to continue to deploy our prototype in real settings. Furthermore, we will investigate to conduct other service management process across domain boundaries based on a similar, coordination-based approach.

References

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

[2] The Atom Syndication Format. http://www.ietf.org/rfc/rfc4287.txt, 2005.

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

[4] D. Box et al. Web Services Eventing (WS-Eventing) W3C Member Submission 15, 2006.

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

[6] Distributed Management Task Force, Inc. (DMTF). Common Information Model (CIM) Specification, Version 2.2, 1999.

[7] Distributed Management Task Force, Inc. (DMTF). Web Services for Management (WS-Management), Version: 1.0.0, 2008.

[8] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California Irvine, 2000.

[9] J. Gray and A. Reuter. *Transaction processing : Concepts and techniques*. Morgan Kaufman, 1993.

[10] P. Grefen, K. Aberer, H. Ludwig, and Y. Hoffner. Cross-organizational workflow management for service outsourcing in dynamic virtual enterprises. *IEEE Data Engineering Bulletin*, 24(1), 2001.

[11] Information Technology Infrastructure Library. Service Transition, May 2007.

[12] L. F. Cabrera et al. Web Services Coordination (WS-Coordination), version 1.0, 2005.

[13] H. Ludwig, J. Laredo, K. Bhattacharya, L. Pasquale, and B. Wassermann. Rest-based management of loosely coupled services. In *Proc. of the World Wide Web Conference*, 2009.

[14] H. R. Motahari Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi-automated adaptation of service interactions. In *Proc. of the World Wide Web Conference*, 2007.

[15] OASIS. Web services distributed management: Management using web services (wsdm- muws 1.1), 2006.

[16] OASIS. Web Services Base Notification 1.3 (WS-BaseNotification), 2006.

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

[18] T. Setzer, K. Bhattacharya, and H. Ludwig. Decision support for service transition management. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium*, 2008.

[19] The IT Governance Institute. COBIT 4.1, 2007.