Dealing with Scale and Adaptation of Global Web Services Management

William Vambenepe, Carol Thompson, Vanish Talwar, Sandro Rafaeli,
Bryan Murray, Dejan Milojicic, Subu Iyer, Keith Farkas, and Martin Arlitt

HP

[firstname.lastname]@hp.com

Abstract

Service Oriented Architectures (SOA) are becoming the prevalent approach for realizing modern services and systems. SOA offers superior support for autonomy (decoupling) and heterogeneity compared to previous generation middleware systems, resulting in more scalable and adaptive solutions. However, SOA have not adequately addressed management, while traditional management solutions do not sufficiently scale to address the needs of (global) Web services.

We propose scalable management based on models and industry standards. We discuss a use case for global service management, we present its design, implementation and preliminary evaluation. We retain all the benefits of SOA while also enabling global scale manageability. Our approach provides manageability that is comprehensible for administrators yet automated enough for integration into autonomous systems.

1 Introduction

The increasing scale and complexity of systems and services makes them increasingly difficult and expensive to administer. In addition, traditional enterprise data centers are being complemented with so called closet computers emerging from remote and home offices. New computing models, such as Utility Computing [1, 2], Grid Computing [3], and PlanetLab grow even more significantly in scale.

Service Oriented Architectures (SOA) [7] contribute to increased scale and availability of services, but they do not sufficiently address the management of services. An update at a moderately-sized data center may require changes to software on thousands of machines. In the case of global services in a large enterprise, a software update may require touching hundreds of data centers. In addition, there may be interdependencies among the services. For example, a Web-based e-commerce application may consist of a virtual store, catalog, customer relationship, and billing services, among many others. At the infrastructure level, this application is usually mapped on a three-tier system architecture, comprising the database, application, and Web server tiers. The application tier further consists of the application server, the application in question, and other services on which the application depends. Large scale data centers in financial, public and private sector, etc. can be significantly larger in size with significantly more complex services.

As services become globally deployed, the design assumptions are changing. *Scalability* requirements change as administrative boundaries are cross. *Availability* needs change as companies move from expensive, private networks with well defined policies to the to the Internet and poorly-defined policies and best practices. Such shifts require *adaptation* to unexpected loads, rebooting and upgrading of machines, networks, and services. As the systems continue to grow in size and global deployment, the traditional management approaches become less effective. To address these new requirements, we propose a new way of scalable management, based on the use of models and standards-based interfaces. The work presented in this paper is related to our work on approaches to service deployment and on scalable communication described elsewhere [8, 9].

The rest of the paper is organized in the following manner. In Section 2 we overview related standards in the management area. Section 3 presents a use case scenario. In Section 4 we describe our solution. We evaluate this solution in Section 5 followed by lessons learned in Section 6. In Section 7 we discuss how our solution can be extended and then we compare it to the related work in Section 8. Finally, in Section 9 we present the summary and future work.

2 Industry Standards Background

Our work relies on the use of industry standards in order to ensure that there is interoperability between long-lived global services as well as infrastructures they execute on. In this section we provide a summary of standards in the area of models, management, deployment and workflows.

Web Based Enterprise Management (**WBEM**), is a set of management standards for distributed computing environments, developed by the Distributed Management Task Force, Inc. (DMTF). WBEM has been designed to simplify system management across multi-

IEEE COMPUTER SOCIETY

ple computing environments. The core set of WBEM standards includes the Common Information Model (**CIM**) standard, a data model for representing common management information for systems, networks, applications, services, and the dependences between these components. CIM specifies a schema, which provides the definitions of the model, and a meta-schema, which facilitates integrating CIM with other models.

The Web Services Distributed Management (**WSDM**) technical committee in OASIS produced the Management Using Web Services (MUWS) specification to describe a standard way to advertise, expose and access manageability capabilities through Web services. The specification defines the notions such as manageable resources, manageability endpoints, and manageability capabilities. It provides a common way to handle manageability endpoints and assess their identity. Management models such as CIM can make use of WSDM MUWS to make their semantics available through the standard mechanism for exposing management information through Web services.

The GGF's Configuration Description, Deployment, and Lifecycle Management Working Group (**CDDLM**-WG), pursues Web service deployment in the Grid space. The CDDLM deployment is an extension of the OASIS WSDM. CDDLM defines a language for specifying deployment requests, the component model that enables services to become deployable, and a set of Web services interfaces (in WSDL) for invoking deployment.

The Business Process Execution Language (**BPEL**) is a standard published by OASIS. BPEL for Web services is an XML-based language designed to enable task-sharing for distributed computing. BPEL orchestrates Web Services by specifying the order in which it is meaningful to invoke a collection of services. A Business Process in BPEL is composed of several Web Service invocations, Receptions, and Decision Points with simple conditional logic and parallel flows or sequences.

3 Use Case: Global Services Management

In this section we consider a scenario involving the deployment of a global scale, three-tier e-commerce application. The scenario consists of deploying the application onto a large number of nodes. Some nodes support the database, while others support the Web and e-commerce application. The Web application is configured to connect to the correct database node. It is possible, in case of failures, to reconfigure the Web application to migrate to a different database server. A few nodes have other services running, and these services use the default ports of the Web server and database server, which means that the deployed application has to be configured to run on a different port. The database and Web applications are customized for geographic location. For example, the application running on a node in Brazil is presented in Portuguese and should offer products that are relevant to Brazilian people. Configuring the correct language requires detecting the language to use and then activating the proper Web application files as well as filling the database with the right product catalogue.

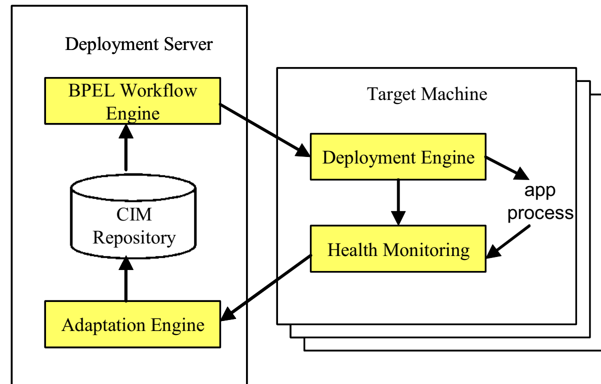

Figure 1 Solution Components

In this scenario, there is also the possibility of failures. These faults can happen to one or more Web application, database or Web servers. There is some way to monitor the deployed applications in order to detect these failures and then take some action to solve the problem. This scenario requires the following:

1. Web-services-based scalable deployment: for decoupled and scalable communication
2. Model-based configuration and adaptation: for machine-readable system configuration
3. Event-based notification: for scalable failure notification (pursued elsewhere by Adams et al. [10])

In our work, the core service model is the same in all deployments, and the localization is modeled as an extension to the core model. This allows us to configure, deploy and manage the services in a coherent manner, maintaining a consistent view of the deployments, regardless of customization. Furthermore, using the standard manageability interfaces enables the components to configure each other on as needed basis.

4 Our Solution

We have built a solution to address the problems brought about in previous sections. The solution consists of model-based services for deployment, health

IEEE COMPUTER SOCIETY

```
public class GenericRPMInstaller
{
  public boolean install(String parameters) { ....
      // download the packages
    RsyncDownloader downloader = new
    RsyncDownloader(downloadFromDir,downloadToLocation,
    new Integer(downloadBlockSize).intValue());
    downloader.download();
      // install the package
    String installCmd = rpmCmd+downloadToLocation+"/"+rpm;
    File file = new File(downloadToLocation); .....
    p = Runtime.getRuntime ().exec (installCmd,null,file); .....
  }
}
```

Figure 2. Snippet of Deployment Component

monitoring, and adaptation. (see Figure 1). We are using the solution to manage several instances of JPetStore services deployed globally on Planetlab [11]. Planetlab is a research testbed consisting of nodes over the globe. All of the nodes run a common software package that includes a Linux-based operating system and support for distributed virtualization. In summary, Planetlab provides us with a computing environment that has characteristics of *scale*, *virtualization*, and *dynamism*.

4.1 Deployment Service

Our deployment service is reponsible for performing the installation, configuration, activation, de-activation, and de-installation of global application services. It is primarily comprised of an infrastructure component consisting of web services-based deployment and workflow engines; a service description component consisting of language parsers and interpreters; and an eventing component consisting of event triggers and event visualization tools.

To use our deployment service for JPetStore, we first describe the logic for installation, configuration, and activation of the service as Java methods of a management component. An instance of a JPetStore testbed consists of a Tomcat server, a MySQL server, and JPetStore application files. A typical deployment process involves the download of each of these packages, the installation to appropriate folders, their configuration, and then subsequent activation. We wrote generic Java components that capture the logic for performing these actions.

The Java component is designed so that it can read many of the parameters specific to an application through a configuration file. The generic Java components we wrote include `GenericRPMInstaller`, `GenericTarInstaller`, `GenericActivator`, `GenericRSyncDownloader`, and `GenericFailure-Detector`. These components are then distributed as a library along with the deployment engine infrastructure package. (see Figure 2 for the example code snippet).

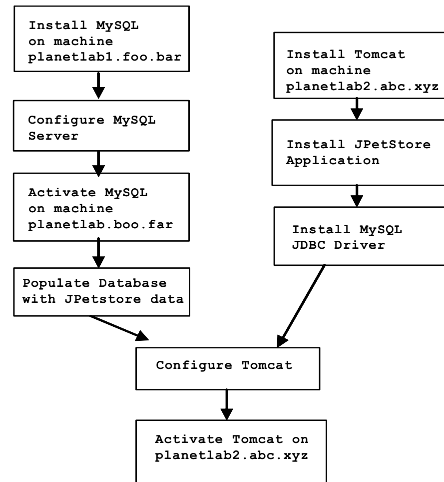

Figure 3. Workflow for the Deployment of JPetStore.

The web services based deployment engine exists on all of the deployment target nodes. It receives and processes the deployment requests given to a deployment target node. Based on a deployment request, it locates the appropriate Java component responsible for a request, and then invokes the appropriate methods on that component.

At the time of deployment, we describe the specific configuration information needed during the JPetStore deployment in a well-defined deployment language. These parameters are, for example, the name of the deployment server, the package names, the destination directories, the download byte size, etc. The language parsers and interpreters execute at the deployment target nodes. They are invoked during the execution of the appropriate Java components at the target node.

We also describe the deployment dependencies that exist among the various components of the JPetStore package as a workflow. Figure 3 shows the conceptual workflow needed for an instance of a JPetStore. This is formally represented in a workflow language, wherein we describe the destination host, the functionality to be performed, and the configuration language specification for the deployment step. In this workflow, we map the dependency requirements that the application service provider has specified to the actual instances of the packages and services within the system. (See Figure 4 for the example code).

The BPEL workflow specifies a composition of tasks to be performed by the management components and it is provided to the BPEL workflow engine. The workflow engine executes at the deployment server node. It parses and processes the deployment workflow descriptions. It

IEEE COMPUTER SOCIETY

```
<sequence name="main">
<receive name="receiveInput" partnerLink="client" portType="tns:
PLDBInstallation-Sequence" operation="process" variable="input"
createInstance="yes"/>
.....
<invoke name="invoke-1" partnerLink="deploymentengine-node-24"
operation="invokeEngine" portType="nsx24:DeploymentEngine"
inputVariable="net-xmpp_input"/>
.....
<invoke name="invoke-2" partnerLink="deploymentengine-node-15"
portType="nsx15:DeploymentEngine" operation="invokeEngine"
inputVariable="net-psepr_input"/>
......
</sequence>
```

Figure 4. Snippet of Deployment WorkFlow Specification

then invokes the deployment engines on the target nodes using SOAP. The deployment engine when thus invoked processes the deployment requests as described earlier.

Various event triggers are started during the deployment process. The event triggers are written to send notifications about START, FAILURE, and HEARTBEAT for the deployed process. These events are then visualized through visualization tools.

4.2 Health Monitoring Service

The Health Monitoring Service is responsible for monitoring the execution of application processes started on the target machine. The deployment engine tells the health monitoring service the name of process to be monitored and whatever happens to that process is reported to the adaptation service (see Figure 1).

The health monitoring service is formed by three WSDM-compliant Web services. The DetectFailure Web service is just a place holder for resource properties, namely WATCH and NOTIFY. From time to time, the resource properties are updated, at which time it sends notification events to its subscribers.

The WatchService Web service subscribes to the WATCH resource property of DetectFailure. When a notification is received, the WatchService starts a failure detection service for monitoring an application process. The NotifyService Web service subscribes to the NOTIFY resource property of DetectFailure. Whatever is written to NOTIFY is then provided to NotifyService that on its turn translates the WSDM event into an external event and it is sent to adaptation service.

The Health Monitoring Service is triggered by the deployment engine (see Figure 5). Once the deployment engine has started the deployment of an application, it calls the SetResourceProperty operation (WS-ResourceProperties) on DetectFailure and sets a new value to the NOTIFY resource property.

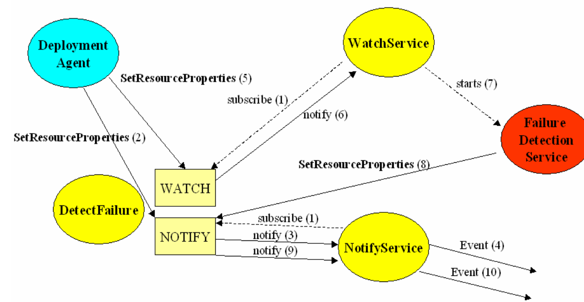

Figure 5 Health Monitoring Service

At this moment, NOTIFY is set to a STARTUP event. The NotifyService is then notified of this event and then translates it from WSDM to an external event and sends it to adaptation service. Once the deployment engine has finished deploying (started) that application, it calls the SetResourceProperty operation on DetectFailure and sets a new value to the WATCH resource property. The WatchService is then notified of the new WATCH value and based on its content the WatchService starts a failure detection service.

On its turn, the failure detection service keeps watching the application process and generates events of the current state of that process. It calls the SetResourceProperty operation on DetectFailure and sets a new value to the NOTIFY resource property. This event is received by NotifyService and passed on to the adaptation service. There are two types of events generated by failure detection service. The first one is HEARTBEAT, which tells adaptation service that the designated application is up and running. The second event is FAILURE. This event tells the adaptation service that the process is no longer running on the target machine. After generating a FAILURE event, failure detection service stops running.

4.3 Model Based Adaptation Service

The implementation of the adaptation service is comprised of CIM repositories; a discovery and eventing mechanism that populates and updates the models throughout the service lifecycle; and scalable decision making services that act upon the information in the models for adaptation.

The motivation for using models is the need to capture in a structured manner the application details, the dependencies among various application components, and their relationship with the underlying hardware. For example, in a standard three-tier application, several application servers could talk to one database server. So, if the database goes down, all of the application servers connecting to this database server would also fail. We

IEEE COMPUTER SOCIETY

look at CIM models as a way of capturing the complex relationships between different application components.

We are using the WBEM implementation for CIM repositories. We create a model of JPetStore instances. Several instances of the JPetstore testbed exist and their attributes are each customized based on geography and internationalization.

An eventing mechanism is used to receive change events from the Health Monitoring Service. This communication between the Adaptation Service and the Health Monitoring Service happens through an external publish/subscribe event system. What happens is that instead of subscribing to DetectFailure's NOTIFY resource property, the adaptation service subscribes to a single given topic on this event system. Whatever information is written to NOTIFY is translated from a WSDM event to this event system format. The design option of using an external publish/subscribe system instead of directly using WSDM's WS-Notification mechanism is driven by the scalability required by highly distributed systems.

Using WS-Notification, the adaptation service would have to subscribe to all NOTIFY resource properties on every target machine being deployed. This clearly does not scale to a large number of target machines (or nodes). However, by allowing the adaptation service subscribe to only a single topic, the burden of managing all events generated is passed to the publish/subscribe system infrastructure being used. It is assumed that such system can handle the expected number of events generated by the health monitoring service.

On receiving these change events, the model is updated to reflect the changes. The information is then acted upon by decision making engines. In our implementation, we perform a redeployment in case of failures. Such a redeployment action takes into consideration the dependencies that exist among various application components. In many cases, the decision making engine needs knowledge about the current state across multiple distributed nodes.

The whole process is prone to failures during deployment time, which means that our adaptation service could never receive any FAILURE events because the Health Monitoring Service had not been launched for the given deploying component. For these cases, we start a timeout for every node being deployed. When the timeout expires and no FAILURE or HEARTBEAT events have been received, the adaptation service assumes the node has failed completely and it starts a process of redeploying the component on another node.

We simulate failures of MySQL servers and demonstrate the adaptation service executing a redeployment action and the MySQL servers are restarted.

5 Evaluation

In this section we evaluate our solution by presenting our experience in developing the solution and evaluating the scalability of our prototype.

5.1 Experience in Global Service on PlanetLab

Scale, complexity, and dynamism of the PlanetLab environment resembles the systems of future. PlanetLab is an evolving research testbed, and so are the next generation distributed services. Because we based our design on standard solutions for the various aspects of the system, we were able to build our prototype in less than two weeks. Our experiments consist of deploying the JPetStore application onto 50-100 nodes. On each node, the JPetStore can be customized based on its geographic location. This customization requires initializating each database with its respective products.

The whole deployment process is visualized with a tool (showing dots on the screen as deployment completes or fails). The following events took place during our experiments:

- Planetlab nodes were constantly going up and down. Our initial list of nodes to be deployed during the experiment had 100 nodes. However, a large number of those nodes went down, and as a result, our number of nodes shrank from 100 to 36, then to 22 and then to 12. In less than 36 hours, our setup was reduced to one tenth of its original size.
- Our management service is not currently capable of adapting to such varying conditions, nor is the underlying infrastructure capable to provide resource guarantees. Just before we started running our experiment, the configuration of our communication infrastructure was modified. This change was not formally captured by our system, and as a result, the configuration changes made to it were not propagated to our management service and the experiment broke unexpectedly.
- The Planetlab network and nodes were highly loaded leading to unpredictable service response times. Our management service is neither currently handling adaptation to such changing conditions, nor the underlying infrastructure provides any sort of resource guarantees. As a result, the time for the deployment events to get propagated through the communication infrastructure to the visualization tool was much poorer

IEEE COMPUTER SOCIETY

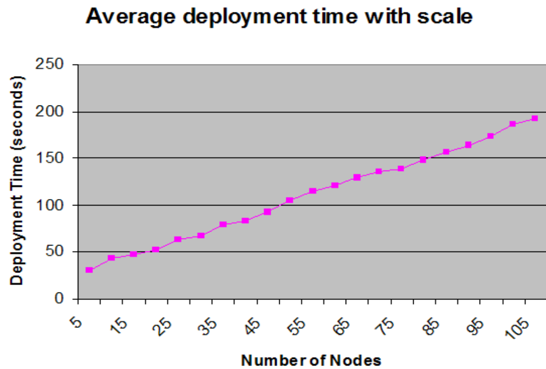

Figure 6 Scaling Web Services Deployment

than expected, with some events taking more than five minutes to complete.

Thus, even though the deployment service code was functioning correctly and it executed the deployment of JPetStore, the interesting characteristics of the computing and service environment caused a disruption in the experiment. The lesson learned is that the management service needs an adaptability layer, which can adapt to the Planetlab environment conditions. We also require a more structured formal representation (i.e., model) of the overall system, including characteristics such as ranges of expected performance and response times, to allow the management system to deal with expected behavior, and to identify anomalous behavior.

5.2 Quantitative Evaluation

We conducted several additional experiments to validate the scalability of our Web Services based JPetStore deployment. We conducted a scale experiment for deploying JPetStore on up to 105 PlanetLab nodes (See Figure 6). Nodes were chosen at random from around the world. The infrastructure on each node for our deployment service included Apache Tomcat Web Server, Apache Axis Web Service Container and our deployment Web Service component. We created a work flow for deploying JPetStore (See Figure 3). We then conducted a scale experiment for the deployment and collected average deployment time. Our measurement infrastructure consists of a process monitoring the start and finish times of each JPetStore deployment. The data is then analyzed to calculate the average deployment time with increasing scale. As you can see from the graph, the average deployment time does not increase linearly with the scale. Although we would like to conduct scale experiments with hundreds if not thousands of nodes, we expect to see a similar pattern for the graph.

6 Lessons Learned

We have learned the following lessons while developing our approach to scalable management:

- *Planetary scale requires careful interaction between applications, service oriented architectures, and management stack.* At the scale of millions of nodes, transparently extending architectures designed for small scale will not work. Our preliminary experience indicates that such applications must be designed with planetary scale on mind. Rather than transparently hiding scalability, in many cases it must be exposed, by explicitly designing for redundancy and distribution where needed.
- *There exist new and different challenges for scalable management with respect to reliability and availability.* Because of the complexity of services, the dependency matching and redeployment of services becomes a critical part of the system. Consequently, separating it from the deployment system (and thus decoupling it from the health of the deployment system) is essential even if it introduces additional problems in maintaining this additional state.
- *Models are only as good as what we want to do with them.* We are relying on the use of models. However, distributed models also pose new challenges for maintaining their consistent state across distributed service deployments. Therefore, the models are not contributing anything in their own right, the key benefit is in the use of models in a manner that serves the given purpose best. This usually means making decision based on incomplete knowledge of the system.
- *Transparency of the WSDM interfaces.* We have used scalable eventing for communication. WSDM was designed with point-to-point management, whereas the eventing was designed for one-to-many communication. WSDM interfaces accommodated for it without any changes to existing design and implementation.
- *Move the BPEL workflow to target machines.* In our current implementation, the deployment workflow specified in the BPEL language is processed by a centralized BPEL workflow engine hosted at the deployment server. Such a design enables processing of cross-node dependencies at a single workflow engine. However, such a centralized orchestration has the limitations of scale. There is a need to partition and distribute the BPEL workflow description to workflow engines on the target machines. Challenges exist to achieve decentralization, such as determining the partition boundaries, and co-ordination among the distributed workflow engines.

7 Extending the Solution

As noted earlier, our scalable management solution is built around an adaptation service that uses the structural information stored in a CIM model repository to automate management actions. We are currently extending this work in two important areas: workflow generation and model implementation.

Owing to the complexity of the compute infrastructure and the services that are run or will be run on top of it, we believe that workflows must be automatically generated. We are currently exploring what information needs to be captured to facilitate such automation, and how to capture it in models. One early by-product of this work is the recognition of the types of dependencies that need to be represented within the model representation. These dependencies are for capturing relationships for installation, activation, de-activation, de-installation, and geography-based customization (e.g., internationalization).

Model implementation, which is our second focal area, embraces a number of issues relating to model structure, model access and security, and model maintenance. Model structure is important because we believe a single model repository running on a single server in a single administrative domain is not scalable, sufficiently robust, nor practical for the wide-area compute environments. We are thus investigating how to decompose the models into multiple repositories located on multiple servers running within multiple administrative domains.

To support this federation, we are exploring approaches for stitching the distinct model repositories into one or more logical views while resolving possible conflicts in the information stored in different models, thereby facilitating access to the stored information. Equally important are mechanisms for handling unplanned model-view outages, and the resynchronization of models once the outages are restored. Complexity in building such views results from the likelihood that multiple modeling frameworks (e.g., CIM, Glue Schema[1], or home-grown framework) are used, or that a given administrative domain will extend a given framework to facilitate local needs. We are thus considering how to bridge the semantic and syntactic differences in model frameworks, for which techniques such as ModelGen [12], will likely be useful. To address model-framework extensions, we believe policies for model governance must be established.

Federation also requires mechanisms for limiting access to a given repository and the information it maintains, and for updating the model repository. Related to model updates is the problem of keeping the model repositories consistent with each other and synchronized with reality. We are exploring mechanisms for automatically discovering models and, as noted above, constructing the layered views of all the repositories.

Similarly, we would like to investigate the possibilities for converging the service models that are used for deployment, health monitoring and adaptation, so that all of the information necessary for the effective management of the service remains consistent, and can be leveraged across the lifecycle of the service.

8 Related Work

The related work falls into categories of deployment, model-based automation, and workflows. In the area of deployment, several tools exist. Deployme system for package management and deployment supports creation of the package, distribution, installation, and deleting old unused packages from remote hosts [14]. Magee et al. describe CONIC, a language specifically designed for system description, construction, and evolution [15]. Cfengine provides an autonomous agent and a middle to high level policy language for building expert systems which administrate and configure large computer systems [19]. A number of other tools are surveyed in [17].

Existing management solutions similarly address functionalities in other areas of our interest, e.g., adaptation to failures and to performance violations ([4],[5],[6]). The effectiveness of these traditional solutions in large distributed systems is significantly reduced by a number of properties of these solutions. These are centralized control, tight coupling, non-adaptivity, semi-automation. Furthermore, these solutions do not adequately address the needs and characteristics of large- scale distributed services.

We base our work on standards evolving in SOA [21-24]. SOA represents a tie between various areas, such as Grid computing, autonomic computing, and enterprise computing, by enabling underlying mechanisms for implementing policies and controls for these different domains. A number of projects use workflows for orchestrating tasks in large scale dynamic environments, such as [25, 26]. Our work has a lot of similarities with all above areas, however, our primary focus is on very large scale, global services.

9 Summary and Future Work

In this paper we have described an approach for managing planetary-scale services. Our approach is based on the use of models and standards. We have demonstrated

1. *http://www.cnaf.infn.it/~sergio/datatag/glue/*

a use case and then presented our solution to it, followed by an initial evaluation. We claim that adopting this approach will enable easier management of global services and reduce development and adoption barriers. In summary, it will reduce the total cost of ownership of large computer systems running globally-scale services.

In the future, we plan to pursue the following opportunities. For the *deployment*, we plan to use WSDM MOWS to manage the deployment service e.g., to handle failures occurring during deployment process. We will also further integrate the GGF CDDLM language parser, deployment APIs, and component models. We will finally, distribute workflows so that we eliminate it as a potential bottleneck for very large scale deployments. In the area of *models*, we shall research federation of models in order to enable more effective decentralized decision making. We will also explore loosely coupled communication among models and prototype example model-based automation services, e.g. adaptation services. Finally, similarly to deployment, we shall explore compliance with standards, in particular using WSDM/CIM interfaces.

Acknowledgments

Parts of this work were conducted in the broader context of the Scalable Management project, with Robert Adams and Paul Brett.

References

[1] Wilkes, J., Mogul, J., Suermondt, J., "Utilification," Proc. ACM European SIGOPS Workshop, September 2004.

[2] "Utility Computing," IBM Systems Journal special issue 43(1), 2004.

[3] Foster, I. et al., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration.", Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[4] HP OpenView http://www.managementsoftware.hp.com/

[5] IBM Tivoli, http://www.tivoli.com/

[6] Computer Associates Unicenter,http://www3.ca.com/solutions/solution.asp?id=315

[7] Huhns, M.N., and Singh, M.P., "Service-Oriented Computing: Key Concepts and Principles," IEEE Internet Computing, vol. 9, no. 1, 2005, pp. 75-81.

[8] Talwar, V., et al. "Approaches for Service Deployment", to appear in IEEE Internet Computing, vol. 9, no. 2, March/April 2005.

[9] Adams et al., "Scalable Management—Technologies for Management of Large-Scale, Distributed Systems", to appear at International Conference on Autonomic Computing (ICAC) 2005.

[10] P. Brett, et al., "A Shared Global Event Propagation System to Enable Next Generation Distributed Services", WORLDS'04: First Workshop on Real, Large Distributed Systems, San Francisco, CA, December 2004.

[11] Peterson, L., et al., "A Blueprint for Introducing Disruptive Technology, "PlanetLab Tech Note, PDN-02-001, July 2002.

[12] Atzeni, P., et al., "ModelGen: Model Independent Schema Translation," Unpublished Report. http://www.dia.uniroma3.it/~atzeni/didattica/SINF/20042005/modelgen.pdf

[13] Dunagan, J., et al., "Towards A Self-Managing Software Patching Process Using Black-Box Persistent-State Manifests," Proceedings of the International Conference on Autonomic Computing, pp 106-113, May 2004, New York, NY, USA.

[14] Oppenheim, K., and MCormick, P., "Deployme: Tellme's Package Management and Deployment System," Proceedings of the Usenix IVth LISA Conference, December 2000, New Orleans, pp187-196.

[15] Jeff Magee, Jeff Kramer, and Morris Sloman. Constructing Distributed Systems in Conic. IEEE Transactions on Software Engineering, 15(6):663--675, June 1989

[16] Goldsack, P., et al., "Configuration and Automatic Ignition of Distributed Applications", 2003 HP Openview University Association conference.

[17] Anderson, P., et al., "SmartFrog Meets LCFG: Autonomous Reconfiguration with Central Policy Control," Proc. USENIX LISA'03 pp 173-180, Oct 2003, San Diego, CA.

[18] Wang, Y.M., et al., "STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support," Proc. of the USENIX LISA'03, pp 159-172, October 2003, San Diego, CA.

[19] Mark Burgess, "A Site Configuration Engine", USENIX Computing Systems, Vol8, no 3, 1995, http://www.cfengine.org

[20] Aiber, S., et al., "Autonomic Self-Optimization According to Business Objectives," Proceedings of the International Conference on Autonomic Computing, pp 206-213, May 2004, New York, NY, USA.

[21] OASIS WSDM WG Charter http://www.oasis-open.org/committees/wsdm/charter.php

[22] DMTF CIM, http://www.dmtf.org/standards/cim/

[23] OASIS BPEL Working Group Charter: http://www.oasis-open.org/committees/wsbpel/charter.php

[24] CDDLM Charter Document, https://forge.gridforum.org/projects/cddlm-wg

[25] Krammer., P., et. al, "Supporting distributed workflow using HTTP", Proc. of the Fifth International Conference on the Software Process, pages 83-94, Lisle, IL, June 1998

[26] Vidal, J.M., et al., "Multiagent systems with workflows", IEEE Internet Computing, 8(1):76-82, Jan/Feb 2004.