Autonomic Container for Hosting WSRF-based Web Services

Christoph Reich
Dept. of Computer Science
Hochschule Furtwangen University, Germany
reich@hs-furtwangen.de

Matthias Banholzer
Dept. of Computer Science
Hochschule Furtwangen University, Germany
banholze@hs-furtwangen.de

Rajkumar Buyya
Dept. of Computer Science and Software Engineering
University of Melbourne, Australia
raj@csse.unimelb.edu.au

Abstract

This paper proposes and presents an autonomic Web Service Resource Framework (WSRF) container that enables self-configuration using IBM's autonomic computing (AC) architecture and resolves resource bottlenecks by service migration. The migration manager bases its migration policy decision on an overall health status metric (H-metric) of service containers. This light-weight migration protocol permits scalability. A unified AC sensor/effector interface, protocol, and metric summarization allows to build up hierarchical WSRF container structures, a virtualized WSRF container. The experimental results demonstrate that our system scales across network of autonomic containers as service demands increases at runtime.

1. Introduction

Service-Oriented Architectures (SOA) key concepts are services, which are loosely coupled to support the requirements of business processes and software systems. SOA does not prescribe a specific implementing technology, but it is very often realized with Web services. Web services, based on many standards, are by themselves stateless. Nevertheless, workarounds like reading states from a database, cookies or WS-Session exist. The Web Service Resource Framework (WSRF) [23] standard specified by OASIS [22] defines a set of standards that allow Web services to become stateful. We believe that in the future most of the services will follow the WSRF specification, that simplifies the inter-operation and adds the possibility to use properties alongside its methods. Extending the WSRF functionality with Web Service Distributed Management (WSDM) [20] is beneficiary for management tools, allowing them to enumerate and view resources, even if they have no other knowledge of them.

The distributed nature of SOA systems makes it necessary for a good overall performance that the service containers are autonomic in respect of self-configuration, self-optimizing, self-healing, and self-adapting [18],[19]. These computing autonomic (AC) system features are implemented based on the widely referenced architecture MAPE (Monitor, Analyze, Plan, and Execute) [5] proposed by IBM. The MAPE loop control is governed by policies stated through SLAs and by performance metrics.

The proposed framework incorporates the AC features of maintaining the given performance metrics by self-managing dynamically the thread pool size (max, min spare, max spare), cache size, thread priority, etc.

If the container is not able to adapt to the changing environmental conditions and predicts a performance metric violation that might occur in the future, the container requests for service migration to solve resource bottlenecks. The migration policy could clone or move a service depending on the policy rules. In general with migration the following problems can be addressed: performance optimization, cost optimization, and fault tolerance (high availability or shutdown for system maintenance).

Our contributions: In this work we have made the following key contributions:

1. proposed and developed an autonomic container for hosting WSRF-based Web services,
2. identified parameters that characterize the performance of Web service containers and introduced migration techniques driven by them,
3. proposed a health status metric for simplifying evaluation of container performance,

4. designed and implemented innovative components using a JSR-77 [15] compliant (Geronimo/Tomcat) hosting environment and

5. carried out experiments demonstrating the potential of our work in scalable and dynamic network environments.

The rest of the paper is organized as follows: First, we discuss related works, we follow this with a presentation of the architecture of our autonomic container along with detailed description of its components. Then, we present migration capability supported in our system along with the metrics used in evaluating the health status of our container environment. Finally, we present experimental results followed by concluding remarks and future direction.

2. Related Work

In Ying et. al service Ecosystem [16] analyses the system under control and reconfigures the service-based system so that they satisfy Service Level Agreements with minimal resource consumption. Migration is a heavy-weight process and should be avoided whenever possible. Satisfying minimal resource consumption is a long term goal. Migrating services to satisfy the minimal resource consumption level can lead to unnecessary merging. The approach of this paper is merging only when resource bottlenecks occur.

Mikic-Rakic et. al. [17] present an applied self-reconfiguration approach to support disconnected operations for increasing the availability of a system during disconnection by allowing the system to monitor and automatically redeploy itself.

Wei [12] carries out migration of weblets, specialized Web services, which can be migrated, according to the round trip time, message size, data location and load of the weblet containers.

A game-theoretic mechanism is used to find a suitable allocation. Each task is associated with a "selfish agent", and requires each agent to select a resource, with the cost of a resource being the number of agents to select it. Agents would then be expected to migrate from overloaded to under loaded resources, until the allocation becomes balanced [6].

The research of Zeid and Gurguis [24] aims at proving that with autonomic Web services, computing systems will be able to manage themselves as well as their relationships with each other. To achieve this objective, the research proposes a system that implements the concept of autonomic Web services without the possibility of merging.

3. Architecture Overview

This section describes the autonomic WSRF service container (see Section 3.1 and 3.2) with the possibility of virtualizing several containers to one virtual WSRF container (section 3.3).

3.1. Web Service Resource Framework (WSRF)

The Web Service Resource Framework (WSRF) standard [23] defined by OASIS [22] will be the basis for all future stateful Web services. Presently there are differences between the OGSI's framework [9], implemented by the Globus Toolkit (GT4) [7], and the WSRF, for example WS-Addressing (see [8] for more details). But there are indications that the two frameworks converge. We belive that the Grid community will move towards WSRF with the WS-Addressing as an alternative grid service container, because open source containers such as Apache Axis2 support complete Web services standards along with WSRF. WSRF subsumes several standards ([23]). Generally it defines how Web service properties are retrieved, set, made persistent, grouped together, and how to manage their lifetime in a standard way. There are a few WSRF implementations (see [13] for a detail comparison). This work is based on Muse [3], a Java-based implementation of WSRF and extended by specifications like: WS-Base Notification (WSN) [21], and WS-Distributed Management (WSDM) [20].

3.2. Autonomic WSRF Service Container Architecture

The WSRF services are deployed in Axis2 [1] running in Tomcat [4] embedded into the application server Geronimo [2] (see Fig. 1). JSR-77 [15] provided by JMX [14] is used to monitor the WSRF services inside the service container (e.g. request counter, processing time, etc.). Remote access is provided by a special management Web service (see Section 3.3 and 4) or by the RMI remote adapter from JMX, if there are no active firewalls. In general the remote management interaction is done by the management Web service. This management Web service contacts MBeans, JMX's management beans managed by the MBean server, to get/set management information, or to define policies, etc. The MAPE architecture is implemented by GBeans, the fundamental entity in Geronimo [11]. GBeans automatically generates MBeans, which are used by the management Web service. Using GBeans provides access to Geronimo's Inversion-of-Control approach [10], wiring MBean connections at deploy time, having a central repository database, and the ability to develop custom applications running as GBeans inside the container.

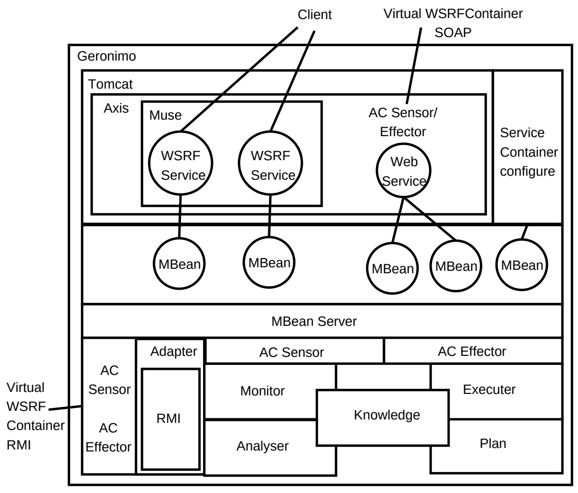

Figure 1. WSRF Service Container Architecture

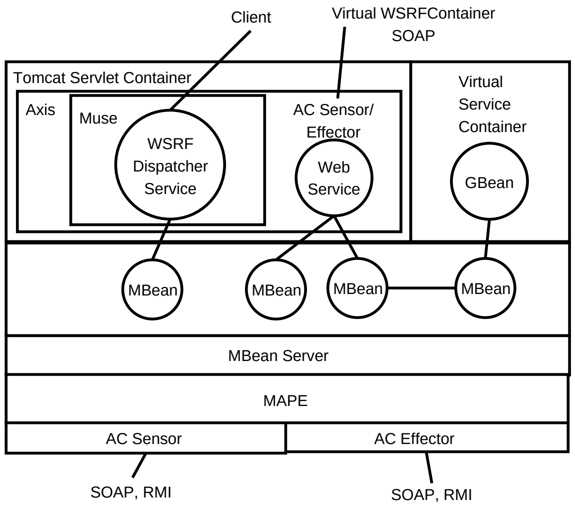

Figure 3. Virtual WSRF Container

3.3. Virtual WSRF Service Container Architecture

To virtualize several WSRF service containers a container with a dispatcher service and the same management Web service as before (see Section 3.2) is needed. Fig. 2 shows the virtualization of three WSRF service containers. The container that should be virtualized is provided by configuration during the deployment process or by a SOAP message to the management Web service. Fig. 3 shows the virtual WSRF service container architecture with the dispatcher service and the management Web service. The architecture is equal to the WSRF service container (see Section 3.2), except with a dispatcher service and different policies, for the migration of WSRF services (see Section 4), and a virtualization component to summerize the metrics of all containers. The virtualization is a general approach

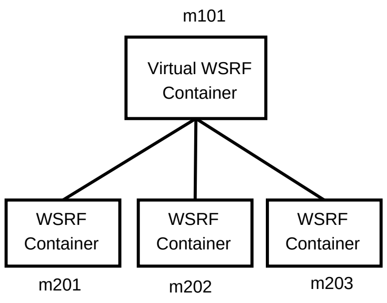

Figure 2. Virtualization of 3 Container

and can be easily extended to a more complex structure as shown in Section 5 Figure 6.

4. Migration of WSRF Services

The approach of this framework is load balancing to solve resource bottlenecks, based on measured performance metrics and the defined policies. A WSRF service is moved or cloned, when a performance metric violation is predicted and the merging strategy, implemented by the virtual WSRF service container, allows to migrate. Thus service containers can be unbalanced from the load point of view if the specified performance metrics are satisfied.

4.1. WSRF Service Performance Metrics and Policies

The deployer who wants to deploy a service has to package the service and describe it according to the necessity of the service container. Additionally the deployer has to define the desired performance metrics, stated as metric parameters, which could be of the following parameters:

metrics p_i	Description of p_i
CPU	Service has to have a specific MIPS or completion time.
Memory	The service needs a certain amount of memory.
Response Time	A service depends on a certain response time.
<others>	other metric parameter attributes defined in the future

Important for this approach is to know a minimum/maximum value of the specified metric parameter.

The minimum and maximum value is used in the H-metric (Section 4.4). For "CPU", "Memory", and "Response Time" special measurement instrumentation have been developed, to get the minimum metric parameters. The maximum metric parameters are given by the user. If other metrics have to be defined one has to make sure to provide the container with the minimum/maximum values. The policy at the moment is to suggest the service that might violate the performance measurement metric for merging.

4.2. WSRF Service Migration Management

Initially the WSRF service containers try to manage everything by themselves. When the MAPE components detect a resource bottleneck, e.g. running out of heap size, the service container has to decide by using it's policies, which of the services should be moved to somewhere else. Therefore, the WSRF service container asks the migration manager for help, given the following information:

- Endpoint Reference (EPR) of the service which has to be moved
- the container's health status metric (H-metric; see Section 4.4)
- the H-metric for the metric value, that violated the SLA, e.g. "out of Memory" and its characteristic H-metric function parameters (x_{10}, x_{90}, etc.).

4.3. Migration Protocol

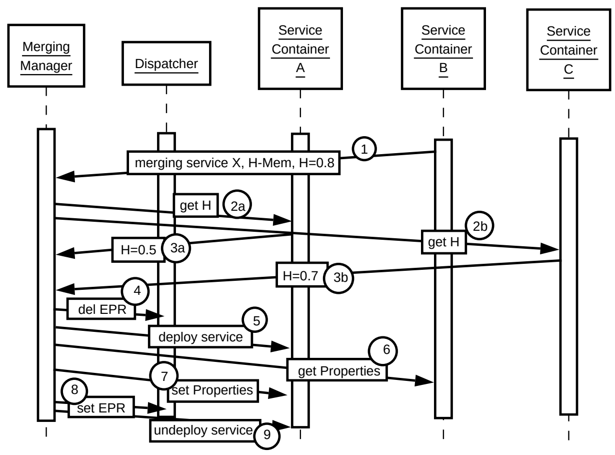

Figure 4. Merging Sequence Diagram

Figure 4 is an example for the migration of a service X from container B to A. The migration protocol is defined as follows:

1. The service container B signals to the merging manager that service X runs out of memory.
2. The merging manager asks all other service containers (A and C) for the H-metric giving them the information that the cause was the "out of memory".
3. Each container calculates the H-metric considering the memory problem and sends back it's values: Container A: $H = 0.5$ and Container B: $H = 0.7$.
4. Therefore, container A is chosen for migration and the merging manager informs the dispatch to delete the EPR for service X. Next the service X is moved from container B to A.
5. Service X is deployed at container A.
6. The Properties of service X are retrieved.
7. The Properties to service X at container A are set.
8. The EPR for service X at the dispatcher is configured.
9. The service X is un-deployed from container B. It might not be necessary; it depends on the policy. For a CPU performance problem the solution could be load balancing using round robin at the dispatch of the virtual WSRF container (see 3.3)[1].

4.4. Health Status Metric (H-metric) of a WSRF Container

Each metric parameter value (see Table in Section 4.1) is normalized (value $\in [0.0, 1.0]$) and mapped to a piecewise linear function (see Equations 1, 2,3,4,5).

$$H_{metric}() = f_{metric}(x, x_{10}, h_{10}, x_{90}, h_{90}) \qquad (1)$$

$$H_{metric}() = \begin{cases} 0.0, & \text{if } x <= p_{min} \\ f_{10}(), & \text{if } p_{min} < x <= \\ & \quad p_{min} + x_{10} * p_{max} \\ f_{10-90}(), & \text{if } p_{min} + x_{10} * p_{max} < x <= \\ & \quad p_{min} + x_{90} * p_{max} \\ f_{90}(), & x < p_{min} + x_{90} * p_{max} \\ 1.0, & \text{otherwise} \end{cases} \qquad (2)$$

$p :=$ metric parameter; $x :=$ measured metric value; $x =\in [p_{min}, p_{max}]$ $x_{10} \in [0.1, x_{90}]$ $x_{10} := 10\%$ metric default value; $h_{10} := 10\%$ H default value; $x_{90} \in [x_{10}, 0.9]$ $x_{90} := 90\%$ metric default value; $h_{90} := 90\%$ H default value; The parameter x_{10} and x_{90} extreme values are: $x_{10} = 0.1$, $x_{90} = 0.9$ (linear case) and $x_{10} = 0.5$, $x_{90} = 0.5$ (switch case).

[1]There is no status synchronization between the services.

$$f_{10}() = \frac{h_{10}x}{x_{10}p_{max}} - \frac{h_{10}p_{min}}{x_{10}p_{max}} \quad (3)$$

$$f_{10-90}() = \frac{(h_{90}-h_{10})x}{p_{max}(x_{90}-x_{10})} + h_{10} - \frac{(h_{90}-h_{10})(p_{min}+x_{10}p_{max})}{p_{max}(x_{90}-x_{10})} \quad (4)$$

$$f_{90}() = -\frac{(1.0-h_{90})x}{p_{max}-p_{min}-x_{90}p_{max}} + 1.0 + \frac{(1.0-h_{90})p_{max}}{p_{max}-p_{min}-x_{90}p_{max}} \quad (5)$$

Figure 5 visualizes the normalizing H function for a memory metric: $p_{min} = 0MB$; $p_{max} = 300MB$; $h_{10} = 0.1$; $h_{90} = 0.9$ and by sweeping $x = [0, 300]MB$ and $x_{10} = [0.1, 0.5]$ with the constraint: $x_{90} = 1.0 - x_{10}$.

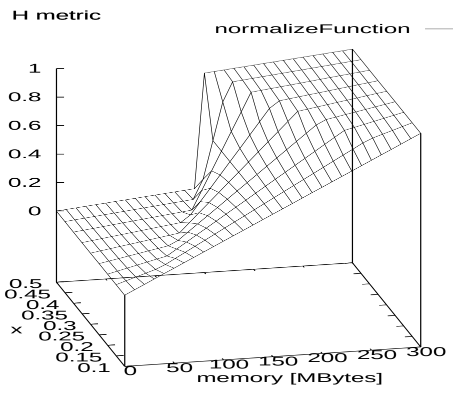

Figure 5. Normalizing Function for the Policy Parameters

To calculate the overall health status metric of a WSRF container the normalized metric parameter values are accumulated and normalized again (see Equation 6 by considering the maximum parameter of all service containers (see Equation 7).

$$H = \frac{\sum_{m=1}^{m=n} w_m * \frac{H_{metric}()_m}{p_m^{MAX}}}{n * \sum_{m=1}^{m=n} \frac{w_m}{p_m^{MAX}}} \quad (6)$$

With m := machine with WSRF Container and w := weights (1.0, if equal machines).

$$p_{MAX} = max(p_{m,max}) \quad (7)$$
$$m \in [1, number - of - machines]$$

p_{MAX} is the maximum value of all given policy parameter maximum values. p_i are policy parameter values. w_i are weight values, to emphasize a particular parameter, such as memory. By default all w_is are set to 1.0. If the overall H metric has to be calculated for a specific problem, e.g. response time, the weight value w_m for that H_{metric} will be doubled and the others adjusted accordingly.

If the virtual WSRF container has to inform another virtual WSRF container about its health status it is reporting the maximum value of H of the service containers under control (see Equation 8).

$$H = max(H_i) \text{ with } \mathrm{H_i} \in [\mathrm{H_1}, \mathrm{H_2}, ..., \mathrm{H_n}] \quad (8)$$
$$n := number - of - machines$$

5. Experimental Evaluation

5.1. Implementation

As indicated earlier (Section 3.2) we have implemented the autonomic container manager inside Geronimo with GBeans using Java. For implementation of WSRF-based services we used the libraries (muse-core, muse-wsrf, muse-util, muse-wsdm, etc.) from Axis2 and Muse. For the merging manger additional libraries for remote deploying based on Geronimo are used (geronimo-kernel, geronimo-util, geronimo-deployment, etc.). The status information of the WSRF-based services are retrieved by using the standard set/get-Property method calls, which are defined in the WSDL document of the services. The dispatcher is implemented as a servlet using the Tomcat servlet library.

5.2. Setup

The machines used for the evaluation have been set up with a Intel Pentium 4 CPU with 2.66G-Hz, 1G-Byte memory. The measured MIPS value for this machines was: 276984. We used version 1.1.1 of the J2EE application server Geronimo [2] with Tomcat [4] included. To realize WSRF services we installed Axis2 [1] and used Muse 2.0 [3] to generate the subs, skeletons, and java interfaces. Figure 6 shows the configuration setup of the machines. For the experiments we used 3 different kind of services with the following metric values defined:

- CPU-Service: CPU average load: $p_{min} = 0.1$, $p_{max} = 3.0$
- Memory-Service: Memory: $p_{min} = 10MByte$, $p_{max} = 100MByte$, and
- Counter-Service: Response Time for all services: $p_{min} = 6ms$, $p_{max} = 160ms$.

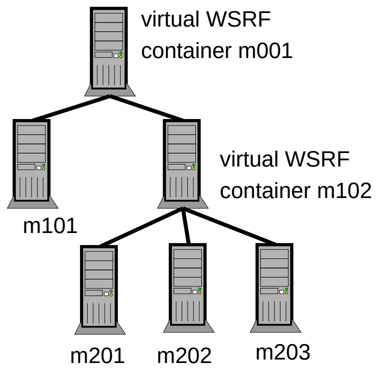

Figure 6. Cascading Service Container Hierarchically

- Merging-Service: Response Time for all services: $p_{min} = 6ms$, $p_{max} = 160ms$.
- Merging-Service SLA: The merging service should be below 140ms average response time.

The parameters for the F_{metric} functions are chosen for the linear case to have a better understanding of the simulation results. Therefore they are for all metric parameters: $x_{10} = 0.1$, $h_{10} = 0.1$, $x_{90} = 0.9$, $h_{90} = 0.9$

5.3. Performance Results

Figure 7 shows the CPU average load, Figure 8 the heap size and Figure 9 the health status metric of all 4 machines. Figure 10 shows the average response time of the migrate service at machine m202 and machine m101. It can be seen that after about 5s the average response time is approaching the 90% metric violation and that the merging is initiated. Since the H-metric of the machines m201 and m201 are higher than that of the machine m202, it is not sensible to migrate it to one of these machines. Therefore the virtual WSRF container m102 tries to move the service somewhere else and asks the virtual container m001 which now moves the merging service to m101. This can be seen in Figure 10 at 20s, when the move of the service is finished.

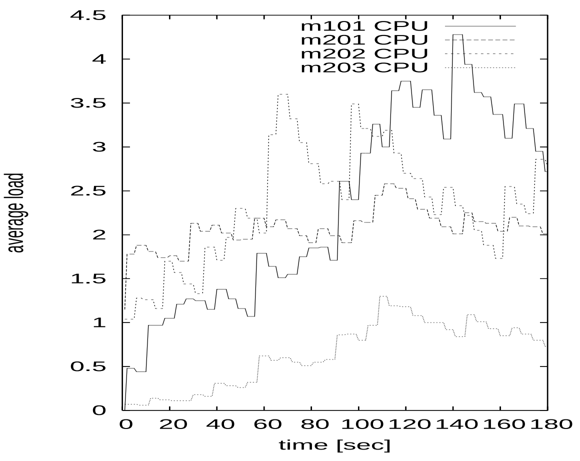

Figure 7. CPU average load metric

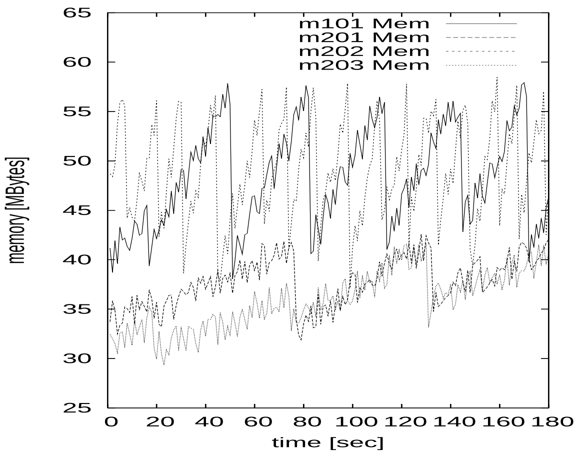

Figure 8. heap size metric

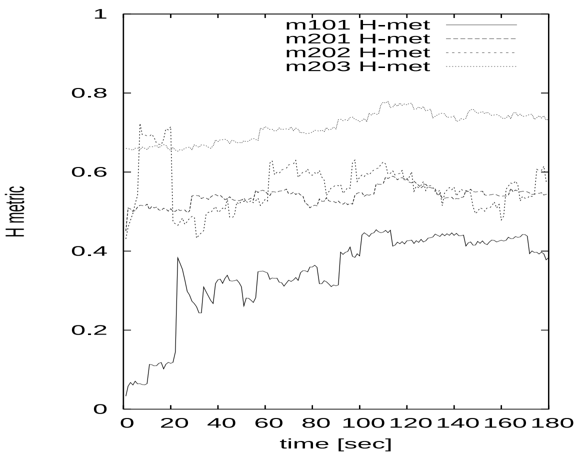

Figure 9. health status metric

6. Conclusion and Future Work

Reconfiguration of a set of server by migrating services statefully is needed to solve resource bottlenecks that a single WSRF container can no longer accommodate by itself. The single overall health status metric (H-metric) of the ser-

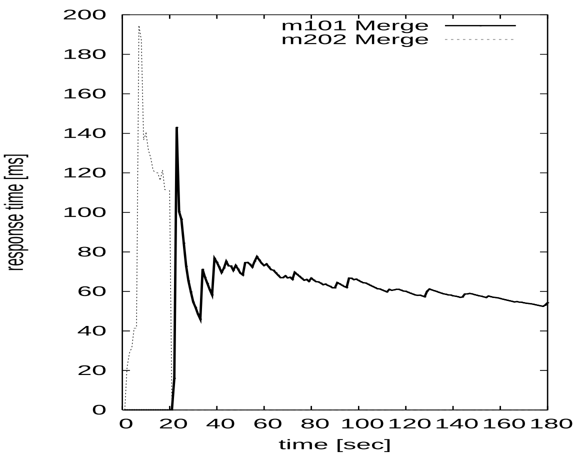

Figure 10. average response time of the merging service at m101 and m202

vice containers is enough to enable the migration manager to migrate services based on its migration policy. The AC sensor/effector interfaces are Web services, the unified management interface, and the single health status metric allows for the build up of hierarchical WSRF container structures, virtualizing it to one WSRF container. Our future work is to extend the framework so that the WSDM standard can be used at the virtual container, summarizing the meta information of all containers. Another goal is to integrate the grid security environment with virtual organizations. In addition, we will explore suitable strategies, driven by business goals and historical demand patterns, for initial deployment of WSRF-based Web services. If their are any changes in the environmental conditions, it will be dynamically handled at runtime as proposed in this paper.

References

[1] Apache axis2/java. Home-Page: http://ws.apache.org/axis2/.

[2] Apache geronimo. Home-Page: http://geronimo.apache.org/.

[3] Apache muse. Home-Page: http://ws.apache.org/muse/.

[4] Apache tomcat. Home-Page: http://tomcat.apache.org/.

[5] An architectural blueprint for autonomic computing. IBM, 2004. Available at http://www-3.ibm.com/autonomic/pdfs/ACwpFinal.pdf.

[6] P. Berenbrink, T. Friedetzky, L. A. Goldberg, P. Goldberg, Z. Hu, and R. Martin. Distributed selfish load balancing. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 354–363, New York, NY, USA, 2006. ACM Press.

[7] I. Foster. Globus toolkit version 4: Software for service-oriented systems. *IFIP International Conference on Network and Parallel Computing, Springer-Verlag*, LNCS 3779:pp 2–13, 2005.

[8] I. Foster, K. Czajkowski, D. Ferguson, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. Modeling and managing state in distributed systems: The role of ogsi and wsrf. In *Proceedings of the IEEE*, volume 93, pages 604 – 612, March 2005.

[9] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. V. Reich. The open grid services architecture, version 1.0. http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf, January 2005.

[10] M. Fowler. Inversion of control containers and the dependency injection pattern. *http://www.martinfowler.com/articles/injection.html*, January 2004.

[11] J. J. Hanson. Manage apache geronimo with jmx. August 2006.

[12] W. Hao, T. Gao, I.-L. Yen, Y. Chen, and R. Paul. An infrastructure for web services migration for real-time applications. In *SOSE '06: Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering (SOSE'06)*, pages 41–48, Washington, DC, USA, 2006. IEEE Computer Society.

[13] M. Humphrey, G. Wasson, K. Jackson, J. Boverhof, M. Rodriguez, J. Bester, J. Gawor, S. Lang, I. Foster, S. Meder, S. Pickles, and M. McKeown. State and events for web services: A comparison of five ws-resource framework and ws-notification implementations. In *4th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, Research Triangle Park, NC, July 2005.

[14] Sun's java management extensions (jmx) page. Home-Page: http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/.

[15] Jsr-77: J2ee management specification. http://jcp.org/en/jsr/detail?id=77.

[16] Y. Li, K. Sun, J. Qiu, and Y. Chen. Self-reconfiguration of service-based systems: A case study for service level agreements and resource optimization. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, pages 266–273, Washington, DC, USA, 2005. IEEE Computer Society.

[17] M. Mikic-Rakic and N. Medvidovic. Support for disconnected operation via architectural self-reconfiguration. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing (ICAC'04)*, pages 114–121, Washington, DC, USA, 2004. IEEE Computer Society.

[18] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. J. Krämer. Service-oriented computing: A research roadmap. In F. Cubera, B. J. Krämer, and M. P. Papazoglou, editors, *Service Oriented Computing (SOC)*, number 05462 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[19] M. Parashar and S. Hariri. Autonomic computing: An overview. In J.-P. B. et al., editor, *Unconventional Programming Paradigms*, volume 3566, pages 247–259, Mont Saint-Michel, France, 2005. Springer Verlag.

[20] Oasis web services distributed management (wsdm) tc. http://www.oasis-open.org/committees/wsdm.

[21] Oasis web services notification (wsn) tc. http://www.oasis-open.org/committees/wsn.

[22] Oasis. Home-Page: `http://www.oasis-open.org/`.

[23] Oasis web services resource framework (wsrf) tc. Web Page.

[24] A. Zeid and S. Gurguis. Towards autonomic web services. In *The 3rd ACS/IEEE International Conference on Computer Systems and Applications*, page 69, 2005.