Engineering an Autonomic Container for WSRF-based Web Services

Christoph Reich[*], Matthias Banholzer[*], Rajkumar Buyya[†] and Kris Bubendorfer[‡]
[*] Department of Computer Science
Hochschule Furtwangen University, Germany
reich@hs-furtwangen.de

[†] Department of Computer Science and Software Engineering
University of Melbourne, Australia
raj@csse.unimelb.edu.au

[‡] School of Mathematics, Statistics and Computer Science
Victoria University of Wellington, New Zealand
kris@mcs.vuw.ac.nz

Abstract

This paper presents an autonomic Web Service Resource Framework (WSRF) container that enables self-configuration using IBM's autonomic computing (AC) architecture and resolves Quality of Service (QoS) problems through service migration. The migration manager bases its decisions on an overall health status metric (H-metric). The H-metric characterises the health *of a service container. A unified AC sensor/effector interface, protocol, and metric summarization allows us to build up a hierarchical WSRF container structure to create a virtualized WSRF container.*

1. Introduction

Service-Oriented Architectures (SOA) feature loosely coupled services that support the requirements of and instantiate business processes and software systems. Service-Oriented Architectures are very often realised using various Web service standards and middleware. The Web Service Resource Framework (WSRF) [14] standard specified by OASIS [21] defines a set of standards that allow Web services to become stateful. We believe that in the future most service architectures will follow the WSRF specification, as it simplifies inter-operability and adds the use of properties. WSRF functionality with Web Service Distributed Management (WSDM) [20] allows managers to enumerate and view resources, even if they have no other knowledge of them.

The distributed nature of SOA systems makes it necessary to monitor and manage deployed services to meet Service Level Agreements. For scalability and ease of human management, we believe that it is essential to adopt autonomic principles for service containers. These autonomic containers are then autonomic in respect of self-configuration, self-optimizing, self-healing, and self-adapting [15],[16]. As a basis for implementing out AC service containers we adopt the widely referenced architecture MAPE (Monitor, Analyze, Plan, and Execute) [1] autonomic framework from IBM. The MAPE loop control is governed by policies stated through SLAs and by performance metrics.

Our autonomic service containers incorporate the AC features of maintaining the given SLA by self-managing dynamically the thread pool size (max, min spare, max spare), cache size, thread priority, and initiating migration to other containers when the performance cannot be resolved internally. If the container detects a performance metric violation (such as a response time that exceeds that permitted by the SLA), the container requests a service migration. In general migration addresses the following problems: performance optimization, cost optimization, and fault tolerance (high availability or shutdown for system maintenance).

In this work we have made the following *contributions:*

1. proposed and developed an autonomic container for hosting WSRF-based Web services,

2. introduced a migration protocol based on the H-metric and additional information to migrate a service to the optimal service container

3. designed and implemented innovative components us-

ing a JSR-77 [19] compliant (Geronimo/Tomcat) hosting environment and

4. carried out experiments demonstrating value of service migration.

The rest of the paper is organized as follows: section 2 describes related work, section 3 describes the architecture of the autonomic WSRF container, section 4 shows the migration protocol, section 5 presents our results, and finally section 6 concludes the paper.

2. Related Work

There have been a number of projects focusing on autonomic behaviour for managing web services, in particular Ecosystem [12] analyses and reconfigures a service-based system (with MAPE) to satisfy Service Level Agreements with minimal resource consumption. They conclude that migration is a heavy-weight exercise and should be avoided whenever possible and that migrating services to satisfy the minimal resource consumption can lead to unnecessary overhead. Like this approach, the principle is to migrate only when resource bottlenecks occur. The major difference is the amount of information the migration decision is using. The Ecosystem gathers lots of information centrally whereas this approach uses the summarised H-metric, which makes it more scalable. Hao [11] carries out migration of weblets, specialized Web services, that can be migrated, according to the round trip time, message size, data location and load of the weblet containers. This work uses standard stateless Web services or stateful WSRF Web services.

Other projects have attempted to address scalability issues for, such as [8], which partitions resources into individual, cluster and grid resources. Dowlatshahi et. al [7] have developed an architecture that uses a hierarchical tree structure for participating nodes distant from the Internet backbone, and uses a single peer-to-peer structure for service discovery at the root layer of the underlying tree structures. The key characteristics of their architecture are optimal search for both distant and close services, minimal overhead traffic, scalability, robustness, and easier QoS support. Mikic-Rakic et. al. [13] present an applied self-reconfiguration approach to support disconnected operations by allowing the system to monitor and automatically redeploy itself.

Berenbrink et. al [6] introduce a game-theoretic mechanism which they use to find suitable allocations. 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. This system is unlikely to scale well, as the resource discovery is centralised. The research of Zeid and Gurguis [22] 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 but without service migration.

3. Architecture Overview

This section describes the autonomic WSRF service container and the means by which it can virtualise several containers to one virtual WSRF container (section 3.2).

3.1. Autonomic WSRF Service Container Architecture

The WSRF services are deployed in Axis2 [3] running in Tomcat [2] embedded within the Geronimo [4] (see Fig. 1) application server. JSR-77 [19] provided by JMX [18] 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 3.2 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 using GBeans,

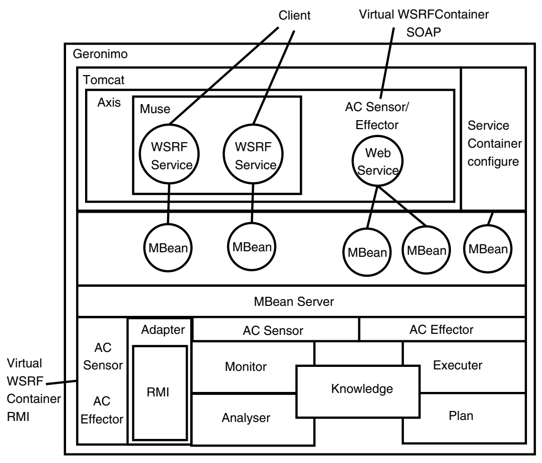

Figure 1. WSRF Service Container Architecture

the fundamental entity in Geronimo [10]. GBeans automatically generate MBeans, which are used by the management

Web service. Using GBeans provides access to Geronimo's Inversion-of-Control approach [9], wiring MBean connections at deploy time, having a central repository database, and the ability to develop custom applications running as GBeans inside the container.

3.2. Virtual WSRF Service Container Architecture

To virtualize several WSRF service containers a container with a dispatcher service is needed. The container that is virtualised is provided by configuration during the deployment process or later by a SOAP message to the management Web service. Fig. 2 shows the virtual WSRF service container architecture with the dispatcher service and the management Web service. The architecture is similar to the WSRF service container (see Section 3.1), except with a dispatcher service and different policies for the migration of WSRF services (see Section 4), and a virtualisation component to summarise the metrics of all containers.

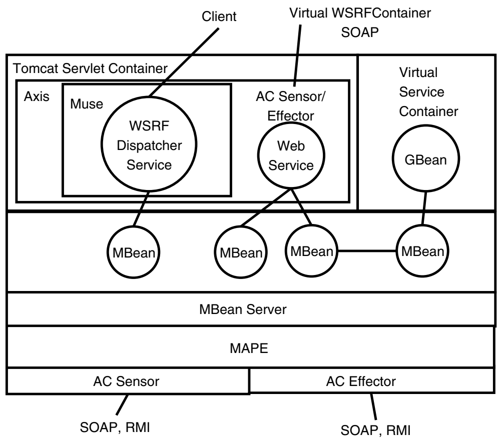

Figure 2. The Virtual WSRF Container

The virtualisation shown is a very general approach and can easily be extended to more complex structures as shown in Section 5 Figure 4.

4. Migration of WSRF Services

The approach of this framework is to use service migration to resolve SLA violations. Migration decisions are based on measured performance metrics and the any set policies. A WSRF service is migrated when a SLA violation is detected. The service moved may or may not be the service that experiences the SLA violation. This choice depends on the policies defined for the virtual WSRF service container.

4.1. WSRF Service Performance Metrics

A deployer who wants to deploy a service has to package the service and describe its SLA to the service container. The SLA parameters we have considered so far for such services are: CPU, where the a service has to have a specific MIPS or completion time; memory, where the service needs a certain amount of memory; response time, where a service depends on a certain response time.

In heterogeneous systems it is important that such values are comparable and thus to this end we have developed the H-metric, which is a single comparable measure of a container's health. For "CPU", "Memory", and "Response Time" special measurement instrumentation has been developed, to get the minimum metric parameters. The maximum metric parameters are determined dynamically from the other machines in the container network.

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

The health metric plays an important role during two distinct phases in the life-cycle of a WSRF container. Firstly during service deployment when services are assigned to containers and secondly during maintenance when violations are detected and services are migrated to other containers to preserve QoS. A H-metric is a simple approximation indicating the overall *health* status of the WSRF container, and is weighted to highlight the resource responsible for the SLA violation. Essentially, each monitored resource is normalised, then all of the resources are summed and renormalised. This allows the state of the responding machine to be summarised in a single comparable number, but permits the resource of interest to carry more weight when selecting a destination for migration. The H-metric is specific to each help request. Two simultaneous requests with different violating resources, will ideally result in two different H-metrics from the same container. Details on the H-metric and its computation can be found in Reich et. al [17].

4.3. WSRF Service Migration Management

Initially a WSRF service container tries to manage everything by itself. 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 4.2)

- the resource that violated the SLA, e.g. “out of Memory” and its characteristic H-metric function parameters (x_{10}, x_{90}, etc.) which allow modeling of non-linear characteristics.

4.4. Migration Protocol

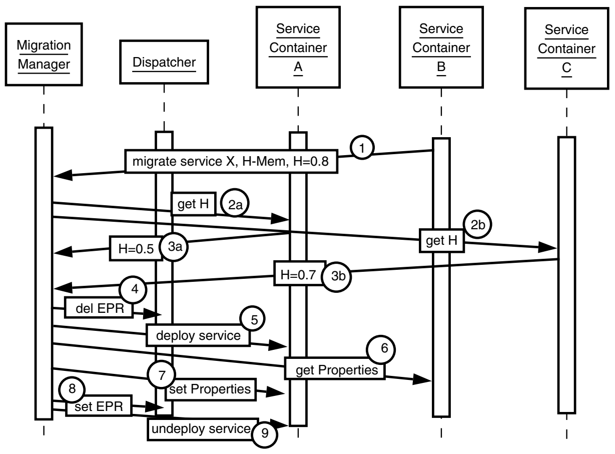

Figure 3. Migration Sequence Diagram

Figure 3 is an example of 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 migrating manager that it has run out of memory and that service X is the migration candidate.
2. The migrating manager asks all other service containers (A and C) for the H-metric giving them the information that the problem was *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 migration 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.

5. Experimental Evaluation

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 migrating 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.

5.1. Experimental 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 class of machines was: 276984. We used version 1.1.1 of the J2EE application server Geronimo [4] with Tomcat [2]. To realise the WSRF services we installed Axis2 [3] and used Muse 2.0 [5] to generate the subs, skeletons, and java interfaces.

Figure 4 shows the configuration setup of the machines and measured H-metric values. 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: $p_{min} = 6ms$, $p_{max} = 160ms$.
- migrating-Service: Response Time for all services: $p_{min} = 6ms$, $p_{max} = 160ms$.
- migrating-Service SLA: The migrating 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: $x_{10} = 0.1$, $h_{10} = 0.1$, $x_{90} = 0.9$, $h_{90} = 0.9$

5.2. Performance Results

Figure 5 shows the CPU average load and Figure 6 the health status metric of all 4 machines.

Figure 7 shows the average response time of the migration candidate service at machine `m202` and machine

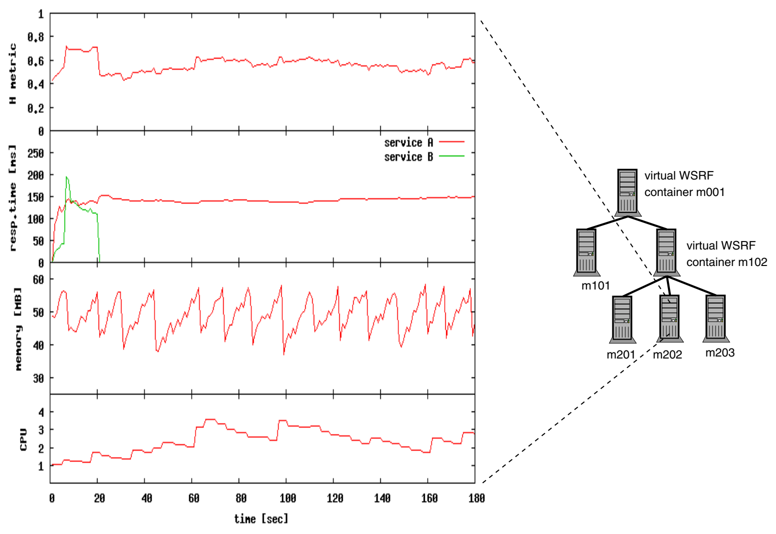

Figure 4. Cascading Service Container Hierarchically

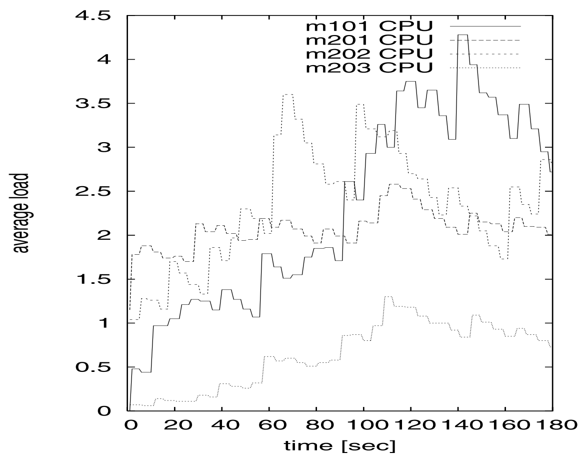

Figure 5. CPU average load metric

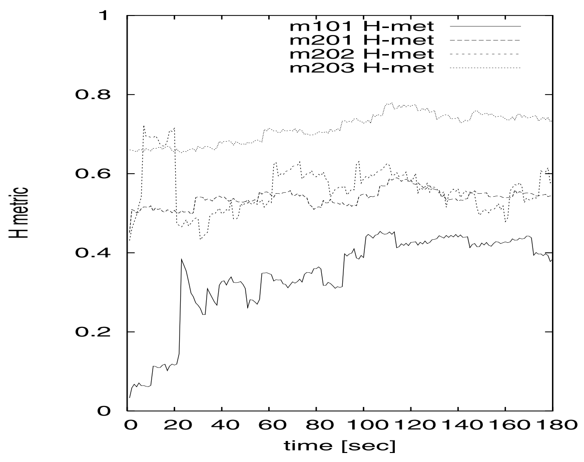

Figure 6. health status metric

m101. It can be seen that after about 5s the average response time is approaching 90% causing a SLA violation. Migration is initiated. As the H-metrics of the machines m201 and m201 are both higher than that of the machine m202, it is not sensible to migrate the service it to one of these machines. Therefore the virtual WSRF container m102 tries to move the service elsewhere and asks the virtual container m001. The result of this migration can be seen in Figure 7 at 20s, when the move of the service is finished.

6. Conclusion and Future Work

Runtime reconfiguration of a set of WSRF containers by migrating services is needed to solve resource bottlenecks that a single WSRF container can not by itself resolve. We introduce a single overall health status metric (H-metric) for the service containers, which we show is enough to enable the migration manager to make sensible migration decisions. In the container implementation the AC sensor/effector interfaces are Web services. The unified management interface, and the single health status metric enables the construction of hierarchical virtualised WSRF container structures.

In our future work we plan to extend the framework so

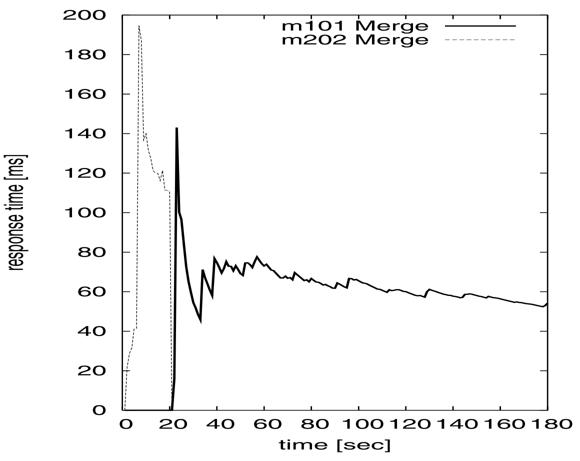

Figure 7. average response time of the migrating service at m101 and m202

that the WSDM standard can be used at the virtual container, permitting the summarisation of the meta information from all containers. We also plan to extend the autonomic management of the containers by permitting them to arrange their own hierarchies and to introduce differentiated SLAs. We would also like to explore suitable strategies, driven by business goals and historical demand patterns, for initial deployment of WSRF-based Web services.

7. Acknowledgements

This work is partially supported by Australian Research Council (ARC) Discovery Project and DEST International Science Linkage Program grants.

References

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

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

[3] Apache. Axis2/java. Home-Page: `http://ws.apache.org/axis2/`.

[4] Apache. Geronimo. Home-Page: `http://geronimo.apache.org/`.

[5] Apache. Muse. Home-Page: `http://ws.apache.org/muse/`.

[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] M. Dowlatshahi, G. MacLarty, and M. Fry. A scalable and efficient architecture for service discovery. In *The 11th IEEE International Conference on Networks, 2003. ICON2003.*, pages 51 – 56, September 2003.

[8] M. El-Darieby and D. Krishnamurthy. A scalable wide-area grid resource management framework. In *ICNS '06. International conference on Networking and Services, 2006.*, pages 76 – 86, Silicon Valley, USA, July 2006. IEEE Computer Society Press.

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

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

[11] 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.

[12] 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.

[13] 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.

[14] OASIS. Web services resource framework (wsrf) tc. Web Page.

[15] 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, 2006.

[16] 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.

[17] C. Reich, K. Bubendorfer, M. Banholzer, and R. Buyya. A SLA-Oriented WSRF Container Architecture. Technical Report GRIDS-TR-2007-9, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, May 28 2007.

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

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

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

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

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