TOWARDS UNIFIED SERVICE HOSTING

Josef Spillner, Iris Braun and Alexander Schill
Computer Networks, TU Dresden, Nöthnitzer Str. 46, Dresden, Germany

Keywords: SOA, Service provisioning, Heterogeneity.

Abstract: Service-oriented computing is increasingly assuming an important role in the research on distributed systems and software engineering. It has also reached application and service hosters by now who need to be able to host off-the-shelf services as well as custom services. Thus, they are faced with a variety of service descriptions, service package formats and runtime demands. Each service technology usually requires a separate execution platform which eventually leads to a high complexity for the administration and management of services. Similar to the unification of invocation of web services through protocol adapters, it is clearly needed to unify management aspects like monitoring and adaptation of services. We therefore propose an abstraction, a formalisation and a unification layer for hosting environments with the intent to pave the way for technology-agnostic, unified service hosting.

1 INTRODUCTION

Software applications can run either locally on a user's computer or remotely on a server. Historically, each *remote application* required its own protocol with binary or text-based data transmission. More recently, the use of *extensible protocol formats* like XML or JSON with a predefined structure and application-specific content has become popular especially in web-centric environments. In combination with a discoverable uniform interface, such applications are now known as *web services*.

Legacy software is often made accessible through the web for interaction with users, or as a web service for programmatic access. Such additions are known as *application service provisioning* (ASP) and lately as *software as a service* (SaaS). Despite this evolution of interface additions, the server-side installation procedure and consecutive service management has not become easier. To date, there are no uniform interfaces available for managing heterogeneous services, essentially preventing the consideration of services as distributable entities.

On the other hand, a growing number of service offerings is being created with an *internet of services* (Petrie, 2007) in mind. Easy installation, often just consisting of hot-deploying a self-containing *service archive*, and registration at a web service directory are among the characteristics of such services (Winkler, 2008). Due to the popularity of this development model, various different service execution techniques and programming languages as well as packaging formats for service deployment have been created and are widely used. Additionally, the runtime environments for such services range from operating-system level execution to highly sophisticated containers, execution engines and middleware infrastructures. In spite of the differences, they all provide a managing container for the deployed services. This assumption contrasts with recent research ideas for distributed, self-managed hosting (Harrison and Taylor, 2005) without containers, which is not widely used yet. We therefore refer to the individual middleware implementations of hosting environments for service execution and management as *containers*.

For service providers, this variety makes it difficult to offer the same level of hosting support and availability guarantees for any service. Even basic management tasks like retrieving a list of available services and service instances is not easily possible with today's systems. A unification of the containers into a *Unified Hosting Environment* (UHE) is thus needed. In this paper, we propose such a unification and will show how heterogeneous service hosting in the internet of services becomes possible without excluding the existing service development community.

Our contributions herein are threefold: First, we explain how to assess containers at an abstract level

Spillner J., Braun I. and Schill A. (2009).
TOWARDS UNIFIED SERVICE HOSTING.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 31-36
DOI: 10.5220/0002238700310036
Copyright © SciTePress

by focusing on the most important service hosting and management aspects. Second, we propose a formal notation in UML based on these abstractions. Third, we outline the concept of a UHE using this notation and well-known software design patterns.

2 RELATED WORK

Background information on the challenges of introducing new network-level and application-level services into existing infrastructures is given in (Villanueva and Touch, 2000). Both deployment and management, refined by us to cover monitoring and adaptation, has thus already been known to raise issues for about a decade.

A common issue with new technologies like SOA is the lack of widespread methodologies. The main contributing factors to this issue are heterogeneous stakeholders, programming languages, protocols, interfaces and implementation frameworks. The OASIS Framework for Web Service Implementation (FWSI) has been designed and extended to cover the requirements analysis, development and testing of services, but the subsequent phases of deployment and operation are not yet covered (Lee et al., 2006). The gap is shown in figure 1.

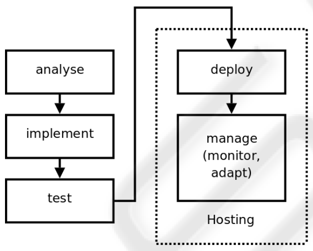


Figure 1: Typical service hosting lifecycle.

An approach to fill the gap is the Service Grid Infrastructure (SGI) (Böme and Saar, 2005). Its purpose is the deployment and execution of heterogeneous services under a number of requirements: dynamic deployment, stateful services and consideration of non-functional properties. By restricting the service implementation technology to OSGi and EJB, the approach fails to take advantage of its otherwise sound decoupled management and execution of stateful services in a grid context. On the management layer, it offers unified access to monitoring data, but no unified adaptation interface. Despite an extension to .NET services later on (Tröger et al., 2007), today's needs for dynamic service hosting range from small CGI scripts and BPEL files to complex, preconfigured virtual machines and corresponding SOA-aware management interfaces.

The Service Component Architecture (SCA) is an approach for unification of development and deployment of services. It wraps each service into a uniform component and defines dependencies between those (Krämer, 2008). However, there are several shortcomings with SCA regarding unified hosting. It requires the presence of the wrapper as part of the distributed service package. Furthermore, while a configuration interface is available, it does not cover dynamic reconfiguration at run-time. The omission of this capability leads to restrictions on adaptation mechanisms. Other dynamic aspects like switching connection protocols are supported, though. Therefore, we will not base our proposal on SCA but rather treat SCA as yet another concrete container in which SCA-compliant packages can be deployed.

Based on the evaluation of existing works, we formalise unified service hosting and propose a more complete and more suitable system which honours native service packages and hence doesn't require developers to learn additional technologies. Our proposal maintains a separation of concerns between access to services from clients and their management from hosting agents.

3 ABSTRACTION OF CONTAINERS

The abstract notation of containers shall be derived from a comparison of their commonalities. The variety of containers makes it hard to compare all of them. We selected typical representatives with a sufficiently high popularity in real-world deployments. This includes Java servlet containers, OSGi service platforms, conventional web servers for web applications, operating systems implementing the Linux Standard Base (LSB) service interface, BPEL engines and virtualised machines. Most of them expect services to be deployed in a format not understood by the other ones, and they also differ in their implementation languages, execution models and management interfaces. Table 1 shows a number of containers, implementations thereof and the accepted package formats for service deployment. For each package type, repositories are listed if they exist and are commonly used.

The capabilities of service packages differ widely. Automatic and semi-automatic functional testing, for example, requires a formal syntactical description of the service interface. In the case of BPEL archives, the corresponding WSDL files are generally included,

Table 1: Overview on service containers.

Container Overview				
	Container	Implementations	Deployment	Repository
1.	OSGi	Knopflerfish, Equinox	bundle, PAR	OBR
2.	Servlet	Jetty, Tomcat	servlet	none
3.	Web server	Apache, thttpd	LSB package	Debian
4.	OS	Linux		
5.	BPEL engine	ActiveBPEL, ODE	BPEL archive	none
6.	SCA runtime	Tuscany, Fabric3	SCA composite	none
7.	VM	Eucalyptus Cloud	Disk image	none

1) OSGi Bundle Repository, http://www.osgi.org/Repository/HomePage
2) Debian Package Search, http://www.debian.org/distrib/packages

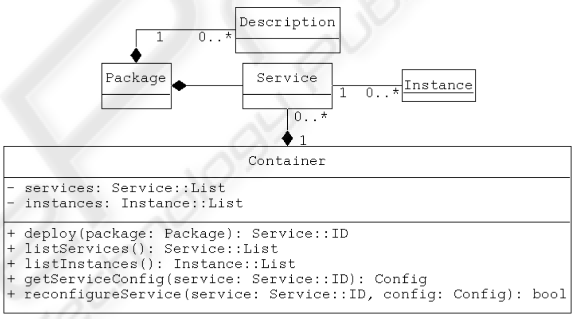
whereas LSB packages don't contain such descriptions in most cases. A study conducted by us found that among more than 24,000 packages of the Debian GNU/Linux distribution, 7,121 are registered as being applications and 267 as being network services. These numbers are likely higher in reality since the process of categorising the packages is ongoing. Yet, despite many of the service packages offering RPC, XML-RPC, SOAP or REST interfaces, only a total of two packages (`Sympa` and `phpWiki`) ship with WSDL descriptions.

More differences can be found in the runtime features of the containers. Some allow for a dynamic external reconfiguration or multiple instantiation, while others don't, as can be seen in table 2.

There is clearly a need to unify the access to such containers so that service hosters can cope with the heterogeneity imposed by the preferences of service developers.

4 A FORMAL NOTATION FOR CONTAINERS

Considering the variety of properties even of abstract views on containers, a symbolic expression will not be able to capture all aspects. On the other hand, extensible and machine-readable notations are hard to understand by humans and are unable to selectively focus on important details. Therefore, we use an UML notation to express a common model for service containers. An excerpt is shown in figure 2. The model refers to a single container which can deploy any number of packages. Each package contains n services which are each described by m formal service descriptions and l endpoints. For each service srv_i each invocation yields an instance $inst_{ilj}$. By extension, the invocation properties can be guaranteed with contracts sla_{ik} for each client for any service. We will however leave discussion of handling contracts out of this paper for brevity.



```
- services: Service::List
- instances: Instance::List
+ deploy(package: Package): Service::ID
+ listServices(): Service::List
+ listInstances(): Instance::List
+ getServiceConfig(service: Service::ID): Config
+ reconfigureService(service: Service::ID, config: Config): bool
```

Figure 2: Formal notation of service containers.

Today's package formats support this model only to a certain degree. For example, the description and endpoint of a servlet-based service cannot be inferred from the servlet metadata automatically. Instead, heuristics need to be applied to find the correct information inside the package. OSGi-based and LSB services already provide rich metadata, including the licence which is important for the consideration of service migration. Still, the packaging formats could be improved to cover the needs of dynamically deployable services.

5 A UNIFIED HOSTING ENVIRONMENT

Based on the abstract notation of containers, a collection of a number of them unified behind a delegating proxy yields a more powerful environment capable of handling more kinds of services.

For the deployment, a package containing the service is routed to the respective container. For exam-

Table 2: Dynamic aspects in service containers.

Container Dynamics				
	Container	Configuration	Reconfiguration	Instantiation
1.	OSGi	manifest file	messages	singleton
2.	Servlet	web.xml	servlet reload	per call
3.	Web server	/etc config	server reload	per call
4.	OS	/etc config	SIGHUP	per call
5.	BPEL engine	none	none	per call
6.	SCA runtime	properties	none	various
7.	VM	/etc config	SIGHUP	singleton

ple, a BPEL file is written to the BPEL server's hot-deployment directory, whereas a web application is stored on the disk followed by a notification of the web server. If the deployment at the container fails or no suitable container has been found, the UHE deployment fails.

Similarly, the reconfiguration either uses an appropriate mechanism of the respective container, or fails if either the mechanism fails or no container can be found. While the heterogeneous nature of containers and service concepts might pose a challenge to such combinations, preliminary findings suggest that at least among the considered implementations there are no hard discrepancies.

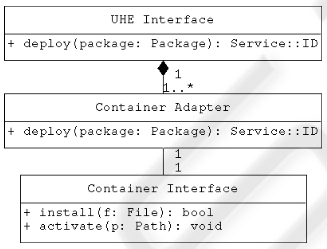

UHE Interface
+ deploy(package: Package): Service::ID
1
1..*
Container Adapter
+ deploy(package: Package): Service::ID
1
1
Container Interface
+ install(f: File): bool
+ activate(p: Path): void

Figure 3: Design pattern notation of container abstraction.

The architecture of the environment can thus be assumed to follow the pattern in figure 3, leading to figure 4. In terms of software design patterns, UHE is defined as a *proxy* connected to a number of *adapters*, each of which wraps a specific container. Any container method can be invoked on the unified environment, and depending on the service type is redirected to the container which handles the service. While the server administrator or corresponding tools communicate with the services through UHE, clients still communicate with each container directly, thus the introduced overhead is being minimised.

A possible extension here is to introduce a recursive-hierarchical distributed UHE by adding a UHE adapter which accepts all package formats and

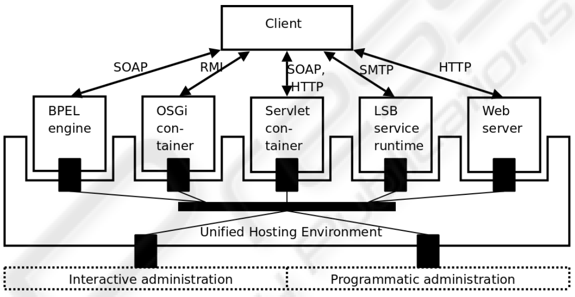


Figure 4: Hosting environment architecture.

redirects requests to other UHE installations, thereby creating a *composite* environment. Due to the many implications like globally unique identifiers, we will not discuss this topic in this paper.

In the following subsections, we will explain the implications of the unification on a number of aspects in hosting environments.

5.1 Deployment

Service packages are assumed to be either self-contained, as can often be seen with servlets and associated libraries in WAR packages, or to rely on a dependency resolution and configuration mechanism as is the case with LSB package managers or OSGi bundle loaders (Cosmo et al., 2008). We thus define the `self-contained` property of the package and the `dependency-resolving` property of the respective container. In both cases, additional restrictions have to be assumed about the existence and versions of the installed system software, including language-level virtual machines, container behaviour and system call interfaces. This assumption can be weakened if the implementation supports a recognition of the required environment configuration and setup of a matching environment using a virtualised operating systems or other techniques. This distinction shall be expressed by the `virtualisable` property of the container, and

Table 3: Properties of service packages.

Service package properties			
	Package type	self-contained	self-described
1.	OSGi bundle	possibly	yes
2.	Axis WAR	possibly	no
3.	Webapp/DEB	no	yes
4.	System/RPM	no	yes
5.	ODE BPEL archive	no	yes
6.	SCA composite	possibly	no
7.	VM image	yes	no

the corresponding `self-described` property of the package.

Therefore, the following property matches are considered a requirement depending on the desired universality of the hosting environment: If any package is not `self-contained`, then the container must be `dependency-resolving`, otherwise the service will not run. If a package is not `self-described`, then the container must meet all its implicit requirements, otherwise the service will not run either.

The table 3 shows the varying degree of self-containedness and self-description of service packages, corresponding to the containers in the previous tables. It is interesting to see that no service package format mandates self-containedness, yet a couple of them usually ship with dependency libraries while others do not.

5.2 Monitoring and Adaptation

A number of software components on a typical hosting system can be considered stakeholders in a unified environment. Adaptation mechanisms, for example, implement abstract adaptation strategies for a controlled change of either the system, the services or contract offers and running contracts, within well-defined limits. Likewise, monitoring agents need ways to capture system-level, container-level and service-level indicators about running service instances, avoiding to directly provide supporting implementations for the growing variety of containers and preferring to use the available query interfaces. Such requirements reach beyond a unification on the messaging level, as is provided by enterprise service buses. UHE is a suitable layer to unify monitoring and adaptation aspects. Adaptation mechanisms on the middleware level may include load balancing (Lodi et al., 2007). In such cases, having a unified interface to deploy packages across machines is beneficial as well.

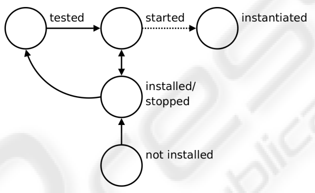


Figure 5: Service state transitions.

5.3 Testing

Another requirement from hosting companies is that each service must first run in a quarantined environment, also known as sandbox, to find out potential issues. None of the containers we have evaluated supports a sandbox. Therefore, we consider it mandatory and architecturally sound to provide such a feature within the UHE, as shown in figure 5.

Migrating an installed and configured service into the production container will take a small amount of work compared to the overall configuration efforts (Sethi et al., 2008).

5.4 Interfaces

A hosting environment should be accessible both by humans and by software. In both cases, the interfaces should not depend on the actual variety and number of containers installed. Instead, the interface should remain stable whenever new technology demands the addition or removal of containers, thus guaranteeing the same level of usability.

For administrators, an interactive interface for basic management tasks like uploading services, listing installed services and running instances and modifying the service states is proposed. It could be implemented as a web interface or as a desktop application integrated into a general system management framework. In addition, we propose having a suitable web service interface to the UHE so that its service management capabilities can be offered to any remote agent. Among the potential users, a service registry could benefit from a formal way of interacting with the containers which collectively can be considered the service repository.

6 DISCUSSION AND CONCLUSIONS

We have explored commonalities and disparities between various containers acting as hosting environments, and based on a comparison presented an abstract view on their structures and abilities. Using a formal notation, we were able to define the notion of a Unified Hosting Environment which acts as a proxy and adapter to all concrete containers. UHE allows for deployment and management of heterogeneous services without any additional service development effort and run-time execution overhead. The concept of UHE is complementary to recent progress towards development methodologies for distributable services. It is supported by specialised implementation techniques with mobile code (Liu and Lewis, 2005) and the tradable service archive format SAR.

Conceptual extension points including contract handling and distributed operation have been identified and will be analysed regarding their unification potential in the future. By now, we have already implemented the deployment part of UHE which will help service providers to keep up with the high variety of development methods and packaging formats in service offerings. The easier service hosting becomes, the faster the vision of an *internet of services* can be turned into a real infrastructure.

ACKNOWLEDGEMENTS

The project was funded by means of the German Federal Ministry of Economy and Technology under the promotional reference “01MQ07012”. The authors take the responsibility for the contents.

REFERENCES

Böme, H. and Saar, A. (2005). Integration of heterogenous services in the Adaptive Services Grid. In *GI-Edition - Lecture Notes in Informatics (LNI), P-69: NODe 2005/GSEM 2005*. 2nd International Conference on Grid Service Engineering and Management, Erfurt, Germany.

Cosmo, R. D., Trezentos, P., and Zacchiroli, S. (2008). Package upgrades in FOSS distributions: details and challenges. First ACM Workshop on Hot Topics in Software Upgrades (HotSWUp). Nashville, Tennessee, USA.

Harrison, A. and Taylor, I. (2005). Dynamic web service deployment using WSPeer. In *Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies*, pages 11–16. Baton Rouge, Louisiana, USA.

Krämer, B. J. (2008). Component meets service: what does the mongrel look like? In *Innovations Syst Softw Eng*, pages 385–394. Springer.

Lee, S. P., Chan, L. P., and Lee, E. W. (2006). Web Services Implementation Methodology for SOA Application. In *Proceedings of the 4th IEEE International Conference on Industrial Informatics (INDIN)*. Singapore.

Liu, P. and Lewis, M. J. (2005). Mobile Code Enabled Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pages 167–174. Orlando, Florida, USA.

Lodi, G., Panzieri, F., Rossi, D., , and Turrini, E. (2007). SLA-driven clustering of QoS-aware application servers. *IEEE Transactions on Software Engineering*, 33(3).

Petrie, C. (2007). The world wide wizard of open source services. Fourth International Workshop on Semantic Web for Services and Processes (SWSP). Salt Lake City, Utah, USA.

Sethi, M., Kannan, K., Sachindran, N., and Gupta, M. (2008). Rapid deployment of SOA solutions via automated image replication and reconfiguration. IEEE International Conference on Services Computing. Honolulu, Hawaii, USA.

Tröger, P., Meyer, H., Melzer, I., and Flehmig, M. (2007). Dynamic Provisioning and Monitoring of Stateful Services. In *Proceedings of the 3rd International Conference on Web Information Systems and Technologies - WEBIST*. Barcelona, Spain.

Villanueva, O. A. and Touch, J. (2000). Web Service Deployment and Management Using the X-bone. In *Spanish Symposium on Distributed Computing (SEID)*. Ourense, Spain.

Winkler, M. (2008). Service description in business value networks. Doctoral Symposium of the 4th International Conference Interoperability for Enterprise Software and Applications (I-ESA). Berlin, Germany.