

# Applying a Web-Service-Based Model to Dynamic Service-Deployment

Christos Chrysoulas<sup>1</sup>, Evangelos Haleplidis<sup>1</sup>, Robert Haas<sup>2</sup>,  
Spyros Denazis<sup>1,3</sup>, Odysseas Koufopavlou<sup>1</sup>

<sup>1</sup>University of Patras, ECE  
Department, Patras, Greece  
{cchrys, ehalep, odysseas}  
@ee.upatras.gr

<sup>2</sup>IBM Research,  
Zurich Research Laboratory,  
Rüschlikon, Switzerland  
rha@zurich.ibm.com

<sup>3</sup>Hitachi Research Lab,  
Sophia Antipolis, France  
Spyros.Denazis@hitachi-eu.com

## Abstract

*Owing to the increase in both heterogeneity and complexity in today's networking systems, the need arises for an architecture for network-based services that provides flexibility and efficiency in the definition, deployment and execution of the services and, at the same time, takes care of the adaptability and evolution of such services. In this paper we present an approach that applies a component model to GT4, a Web-service-based Grid environment, that enables the provision of parallel applications as QoS-aware (Grid) services, whose performance characteristics may be dynamically negotiated between a client application and service providers. Our component model allows context dependencies to be explicitly expressed and dynamically managed with respect to the hosting environment, computational resources, and dependencies on other components. Our work can be seen as a first step towards a component-based programming model for service-oriented infrastructures utilizing standard Web-service technologies.*

## 1. Introduction

In the recent years, Web-service technology has gained importance in the area of Grid Computing. The Open Grid Service Architecture [1] has motivated Grid architects to build environments based on a service-oriented architecture utilizing Web-service technology. The outcome of that effort was the evolution of the Globus Toolkit 4 [2] towards the Web Service Resource Framework [3].

Grids are mostly built following a service-oriented architecture using Web-service technology, which has not been designed to fit the idea of a component-based plug-and-play client programming framework. Services are typically discovered dynamically, using technologies such as the Monitoring and Discovery System (MDS) [4] in Globus Toolkit 4, rather than created. Moreover, they do not provide means to

describe dependencies, for example, in other services running outside the Grid. Web-service technology provides a versatile messaging facility but lacks an extensive component model applicable to service composition. In this paper we present a component-based architecture that addresses the above issues.

Our architecture is based not only on the Globus Toolkit 4 environment, a Grid architecture for the provision of parallel applications as Grid services over standard Web-service technology, but also makes extensive use of a component-based architecture to try to solve the problem of creating new services and the dependencies in other services and in components outside the architecture we present.

Our proposed Dynamic Service-Deployment Architecture is developing as part of the FlexiNET [5] IST research project, specifically as a sub-module of the FlexiNET Wireless Access Node (FWAN) module.

The remainder of the paper is organized as follows: Section 2 briefly explains the technologies used. Section 3 describes what FlexiNET is. Section 4 describes the architecture regarding the dynamic service-deployment. Section 5 presents a user-case scenario. A discussion on related work is given in Section 6. Conclusions and future work are presented in Section 7.

## 2. Technologies used

### 2.1. Web services

#### 2.1.1. Web-service technology

Interoperability across heterogeneous platforms, in particular the need to expose all or part of our application to other applications on different platforms or on different technologies, are the driving force for developing a Web-service architecture.

Web services standardize the messages that entities in a distributed system must exchange to perform various operations. At the lowest level, this standardization concerns the protocol used to transport messages (typically HTTP), message encoding

(SOAP), and interface description (WSDL). A client interacts with a Web service by sending a SOAP message; the client may subsequently receive response message(s) in reply. At higher levels, other specifications define conventions for securing message exchanges (e.g., WS-Security), for management (e.g., WSDM), and for higher-level functions such as discovery and choreography [6]. Figure 1 presents an overview of these different component technologies.

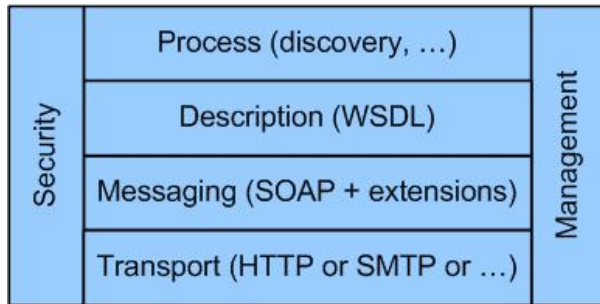


Figure 1. An abstract view of the various specifications that define the Web-service architecture

### 2.1.2. Web-service implementation

From the client's perspective, a Web service is simply a network-accessible entity that processes SOAP messages. To simplify service implementation, it is common for a Web-service implementation to distinguish the following:

- The hosting environment, i.e., the logic used to receive a SOAP message and to identify and invoke the appropriate code to handle the message, and potentially also to provide related administration functions.
- The Web service implementation, i.e., the code that handles the message.

The SOAP message is typically transported via HTTP, thus an "HTTP engine" or "Web server" is required and a "SOAP engine" are needed in order to process the SOAP messages. Figure 2 illustrates these various components.

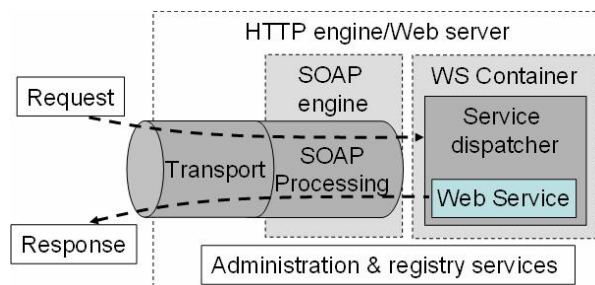


Figure 2. High-level picture of functional components commonly encountered in Web-service implementations, showing the path taken by requests and responses

## 2.2. Globus toolkit 4 (GT4)

GT4 makes use of open-source frameworks such as Tomcat [7] and Axis [8] to provide high-performance applications over standard Web service technology (XML, SOAP, WSDL).

XML [9] is used extensively within Web services as a standard, flexible and extensible data format.

SOAP [10] provides a standard, extensible, composable framework for packaging and exchanging XML messages between a service provider and a service requestor. SOAP is independent of the underlying transport protocol, but is most commonly carried on HTTP.

WSDL [11] is an XML document for describing Web services. Standardized binding conventions define how to use WSDL in conjunction with SOAP and other messaging substrates. WSDL interfaces can be compiled to generate proxy code that constructs messages and manages communications on behalf of the client application. The proxy automatically maps the XML message structure into native language objects that can be directly manipulated by the application. The proxy frees the developer from having to understand and manipulate XML.

The Monitoring and Discovery System 4 (MDS4) component of the GT4 is a key part of our architecture. The MDS streamlines the tasks of monitoring and discovering services in a distributed system.

The monitoring and discovery applications both require the ability to collect information from multiple, distributed, information sources. To this need, MDS4 provides the so-called aggregator services that collect recent state information from registered information sources as well as browser-based interfaces, command lines tools, and Web-service interfaces that allow users to query and access the information collected.

MDS4 makes heavy use of XML and Web-service interfaces to simplify the tasks of registering information sources and locating and accessing information of interest. In particular, all information collected by aggregator services is maintained as XML, and can be queried via XPath [12].

## 3. FlexiNET architecture

As stated in Section 1, the DSD module is developed for the FlexiNET Project. The primary aim of the project is to define and implement a scalable and modular network architecture incorporating adequate network elements (FlexiNET Node Instances) offering roaming connection control, switching/routing control, and advanced services management/access functions to the network access points that currently only support

connectivity between user terminals and network core infrastructures [13], [14], [15].

The FlexiNET network architecture consists mainly of node instances, communication buses and data repositories, as can be seen in Figure 3.

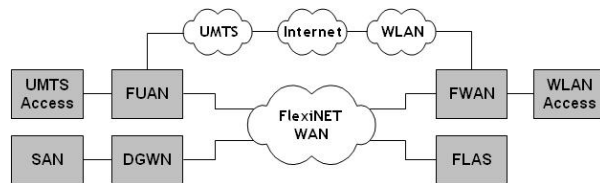


Figure 3. FlexiNET Architecture

The FlexiNET UMTS Access Node (FUAN) provides to the FlexiNET interfaces, functions such as switching/routing control, access to applications data & service logic, etc. The FUAN complements existing access nodes (RNC, BSC) of UMTS networks.

The FlexiNET WLAN Access Node (FWAN) acts as both a services access-gateway (user authentication, service authorization, service discovery, etc.), and connection gateway between WLAN infrastructures and the FlexiNET WAN. FWAN has to achieve user and service-roaming capabilities between different providers and service programmability using dynamic service deployment. Service programmability functions will be provided using the Hitachi distributed-router architecture [16], which will provide dynamic service deployment utilising the ForCES protocol [14].

The FlexiNET Data Gateway Node (DGWN) acts as the gateway between a generic Storage Area Network (SAN) infrastructure and the FlexiNET network. It is used by other FlexiNET instances for accessing subscriber (profile, location, etc) and applications data needed for service execution. It provides a Generic Data Interface that other FlexiNET elements may use to access the stored data in the SANs.

The Generic Applications Interface Bus is used for the implementation of dynamic application-related functions and the communication of information flows pertaining to the execution of application and service logic, including a framework allowing service registration, discovery and binding.

The FlexiNET Applications Server (FLAS) is the physical entity, which hosts the logic of the applications that the FlexiNET network architecture provides. These services are called from other entities remotely and executed locally. Using the DGWN they are in position to retrieve specific information needed for services execution.

The DSD module is part of the FWAN. The FWAN architecture can be seen in Figure 4 and is based on Hitachi's distributed router. Hitachi's distributed router

consists of two functional blocks, the basic and the extensible function block. The basic function block processes the general packet-forwarding functions which are common to all services, e.g. it classifies received packets, handles routing table retrieval, packet switching, and node management. The extensible function block processes the packet-forwarding functions for specific protocols and services, such as packet filtering for firewalls, layer-7 processing for content-switching, address translation, encryption, etc.

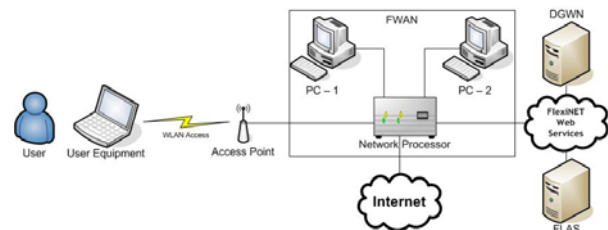


Figure 4. FWAN architecture

As a basic functional block, the FWAN has a network processor, and as an extended functional block, two PCs. A user will access the FWAN through an access point with either a laptop or a mobile phone. The FWAN is responsible for authenticating native and roaming users through the FLAS using an AAA proxy.

The Dynamic Service Deployment module (DSD) must be deployed on the FWAN before boot-up. The bootstrap process is responsible for booting up the FWAN with the AAA proxy module. To accomplish this task, it reads from a static configuration file which is stored in a local copy of the bootstrap process, followed by a series of commands which will be sent to the DSD Module in order to create the FWAN's node fundamental functionalities. The bootstrap process will mainly trigger the installation of the AAA proxy through the DSD module.

The AAA proxy module forwards the authentication packets to the FLAS Server, encapsulates the EAP packets [17] into XML messages that are passed over Web services and vice versa, to authenticate and authorize the user. The AAA proxy service is deployed in the FWAN at boot-up time. It is stored in a local directory and deployed by the DSD module. The code will be requested from the DGWN through Web services.

On boot-up the DSD module is requested by the boot-up process to deploy the AAA proxy module. The DSD module retrieves the AAA proxy service code through the DGWN and deploys it on any of the two PCs based on specific algorithms. In addition, based upon the user profiles, the DSD module will deploy a quality of service module (QoS) that is responsible for providing QoS to specific users. The required configuration of the network processor will be handled

by the ForCEG module which receives Web Service requests from the AAA Proxy and the QoS Module and translates them into ForCES protocol messages [18].

## 4. DSD architecture

### 4.1. DSD definition

Dynamic Service Deployment refers to a sequence of steps that must be taken in order to deploy a service on demand. The necessary steps regarding the service deployment are service code retrieval, code installing destination according to matchmaking algorithms, and service deployment. The matchmaking algorithms provide the most efficient use of system resources by examining the available resources of the FWAN and comparing them with the resources required by the service to be deployed.

### 4.2. Proposed DSD architecture

Figure 5 depicts the current, proposed DSD architecture. As can be deduced from the Figure, the DSD is the sum of the following sub-components:

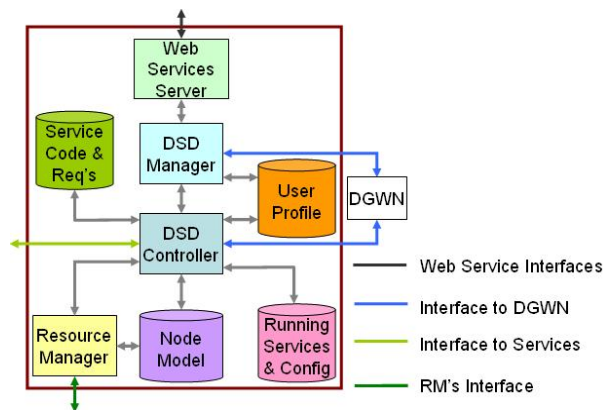


Figure 5. DSD Architecture

#### 4.2.1. Web-service server

The Web-services server sub-component hosts the interfaces with the AAA proxy and the bootstrap process. At this stage only these two processes will interact with the DSD Module. This server is responsible for exchanging messages between the DSD module, the AAA proxy module, and the bootstrap process. The Web-service server sub-component has the functionalities needed to register a Web service in a UDDI directory. This component also is capable of finding other Web-service interfaces.

#### 4.2.2. DSD manager

The DSD manager sub-component has two functions, depending on whether a user profile is required:

- If the AAA proxy communicates with the DGWN, the DSD manager must download the user profile, in order to find, which services must be deployed, and provides the request to the DSD controller.
- In the case of bootstrap process, the DSD manager passes the bootstrap services required for deployment to the DSD controller.

The DSD manager is responsible for checking whether a user has terminated the connection, and for undoing the user's personal configuration.

#### 4.2.3. DSD controller

The DSD controller sub-component has the following duties: it receives the service request from the DSD Manager, communicates with the DGWN in order to download the service code and the service requirements, retrieves the available resources from the node model, performs the matchmaking algorithm in order to find the most suitable resources, and finally deploys the service. The DSD controller is responsible for the services in three dimensions: download, deploy, and configure.

#### 4.2.4. Resource manager

The resource manager sub-component discovers and monitors the resources. With the help of the resource manager Interface, it collects information from all the components of the node model, and also from the DSD controller. The entire information collected is available to the remainder of the sub-components through the WebMDS interface the resource manager provides. Only the necessary information is passed to the node model.

#### 4.2.5. Node model

The node model is responsible for keeping all information about FWAN. It provides a complete view of the FWAN, and contains information on available and used physical resources, as well as data on running services.

#### 4.2.6. User profile

The user profile is the data storage where the user profile downloaded is stored. It is responsible for keeping the user profile.

#### 4.2.7. Service code and requirements

The service code and requirements data storage is responsible for storing the downloaded code and the

requirements (in terms of physical resources) that describe a service.

#### 4.2.8. Running Services and configuration

The running services and configuration data storage is responsible for storing data about running services and their current configuration.

### 5. User-case scenario deployment of the AAA proxy component

The sequence of events that takes place for the deployment of the AAA proxy can be seen in Figure 6

- The DSD manager receives a "Deploy AAA proxy" command through the Web service interface.
- The DSD manager understands that this command has been issued by the bootstrap process and communicates with the DGWN, via a Web service, to download the code and the requirements for deploying of the AAA proxy.
- The DSD controller locates the module where the AAA proxy should reside.
- The DSD controller deploys the AAA proxy.
- The DSD controller sends an acknowledgement to the DSD manager which in turn sends the acknowledgement to the bootstrap process saying that the AAA Proxy has been deployed.

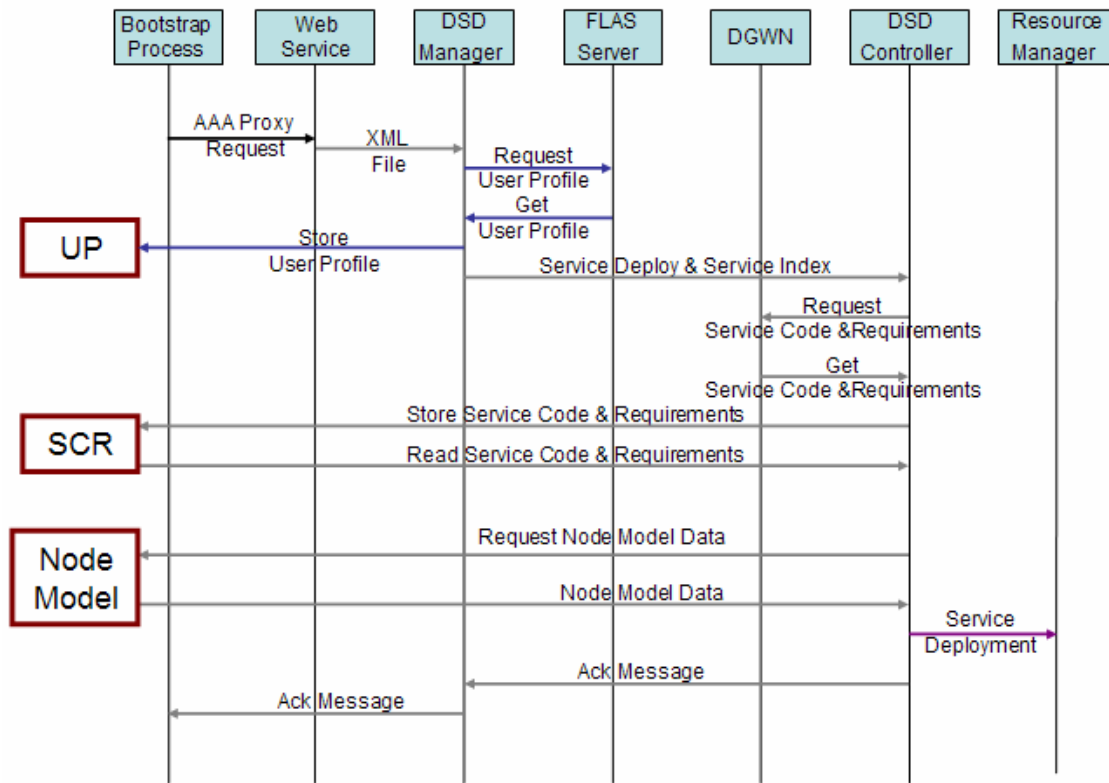


Figure. 6. Message exchange during the deployment of the AAA proxy

### 6. Discussion of related work

Distributed component models, such as Cobra [19] or DCOM [20], are widely used, mainly in the context of commercial applications. The Common Component Architecture developed within the CCA Forum [21] defines a component model for high-performance computing based on interface definitions. XCAT3 [22] is a distributed framework that accesses Grid services, e.g. OGSi [1], based on CCA mechanisms. It uses

XSOAP [23] for communication and can use GRAM [2] for remote component instantiation. The Vienna Grid Environment (VGE) [24] is a Web service oriented environment for supporting High Performance Computing applications. The VGE has been realized based on state-of-the-art Grid and Web service technologies, Java and XML. Globus Toolkit 4 [2] is an environment that mostly deals with the discovery of services and resources in a distributed environment rather than the deployment of the services themselves.

Our model addresses the above issue regarding the dynamic deployment of new services.

## 7. Conclusion and future work

We presented an architecture that adds a dynamic perspective to Web-service-based Grid infrastructure. Our component-based model addresses the issue of dynamic deployment of new services in a distributed environment and the way these address each other in that environment. We expect that this work is not only relevant to the Grid community but also to the Web-services and the network communities as we not only addressed concerns related to Grid computing but also discussed architectural issues regarding Web-service configuration and deployment. Our implementation of the model is still at the prototype stage, and requires further refinement and analysis. As future work we plan to provide a more sophisticated model for service deployment and selection based on QoS properties.

## 8. Acknowledgment

This work is supported by the European Union's FlexiNET Project under contract FP6-IST-1-507646.

## 9. References

- [1] Foster, I., Kesselman, C., Nick, J., Tuecke, S. "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", Globus Project, 2002. Available at <http://www.globus.org/research/papers/ogsa.pdf>.
- [2] The Globus Alliance. <http://www.globus.org>.
- [3] The Web Service Resource Framework. <http://www.globus.org/wsrf/>.
- [4] The Globus Alliance. <http://www.globus.org/toolkit/docs/development/3.9.4/info/wsmds.html>.
- [5] FP6-IST1 507646 FlexiNET Technical Annex.
- [6] A Globus Primer. Available at [http://www.globus.org/toolkit/docs/4.0/key/GT4\\_Primer\\_0.6.pdf](http://www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf).
- [7] Apache Tomcat. <http://jakarta.apache.org/tomcat/>.
- [8] Apache Axis. <http://ws.apache.org/axis/>.
- [9] <http://www.w3.org/XML/>.
- [10] <http://www.w3.org/TR/soap12-part1/>.
- [11] <http://dev.w3.org/cvsweb/~checkout~/2002/ws/desc/wsd120/wsd120-primer.html>.
- [12] <http://www.w3.org/TR/xpath>.
- [13] FP6-IST1 507646 FlexiNET D21 "Requirement, Scenarios and Initial FlexiNET Architecture".
- [14] FP6-IST1 507646 FlexiNET D22 "Final FlexiNET Network Architecture and Specifications".
- [15] Aladros, R., , Kavadias, C., Tombros, S., Denazis, S., Kostopoulos, G., Soler, J., Haas, R., Dessiniotis, C., Winter, E., "FlexiNET: Flexible Network Architecture for Enhanced Access Network Services and Applications", IST Mobile & Wireless Communications Summit 2005, Dresden.
- [16] Hirata, T., and Mimura, I., "Flexible Service Creation Node Architecture and its Implementation", IEEE Computer Communications Workshop, 2003.
- [17] RFC 3748: Extensible Authentication Protocol (EAP), June 2004
- [18] Evangelos Haleplidis, Robert Haas, Spyros Denazis, Odysseas Koufopavlou, "A Web Service- and ForCES-based Programmable Router Architecture", IWAN2005, France.
- [19] CORBA Component Model, v3.0, OMG. <http://www.omg.org/technology/documents/formal/components.htm>.
- [20] COM Component Object Model Technologies, Microsoft, <http://www.microsoft.com/com/default.msp>.
- [21] The CCA Forum. <http://cca-forum.org/>.
- [22] Krishnan, S., and Gannon, D., "XCAT3: A Framework for CCA Components as OGSA Services" Proceedings of the Ninth International Workshop on High-Level Parallel Programming Models and Supportive Environments, April 2004, pp. 90-97.
- [23] XSOAP toolkit. <http://www.extreme.indiana.edu/xgws/xsoap>.
- [24] Benkner, S., Brandic, I., Engelbrecht, G., Schmidt, R., "VGE - A Service-Oriented Environment for On-Demand Supercomputing", Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (Grid 2004), Pittsburgh, PA, USA, November 2004.