COMPONENTS, MIDDLEWARE AND WEB SERVICES

Dimka Karastoyanova
Technische Universität Darmstadt
Wilhelminenstrasse 7, 64283 Darmstadt, Germany
dimka@gkec.tu-darmstadt.de

Alejandro Buchmann
Technische Universität Darmstadt
Wilhelminenstrasse 7, 64283 Darmstadt, Germany
buchmann@informatik.tu-darmstadt.de

ABSTRACT

Web services are a logical evolution of software components and middleware. Based on a comparison of Web services with middleware and components we come up with the statement that Web services are components and middleware in one. However, for the Web services to live up to their potential as an integrating technology for mission critical Web-based applications, it is necessary to provide viable coordination and transaction capabilities. We introduce the notion of hyperware for the framework that will play the same role for Web services the middleware plays for components.

KEYWORDS

Web services, middleware, components, hyperware

1. INTRODUCTION

The natural evolution of computing technologies has brought us object-oriented programming, component-based programming, middleware, and now Web services (WSs). For the obvious reason that software components and middleware are the technologies predominantly used in the computing world nowadays, in this paper we make a comparison between WSs and components, and WSs and middleware. These comparisons motivate our statement that Web services are both components and middleware. We think such a discussion clarifies the position of WSs among integration technologies. We suggest that WSs are going to become a hyper-technology – hyperware as we name it, based on a broad consensus.

A Web service is a *unit of functionality* exposing an XML interface, describing a WS in terms of the messages it receives and generates. It is a *self-describing*, *self-contained* entity that can be *registered* with a registry and located by the potential users. WSs can be combined in any pattern to provide a more complex, functional entity, which can in turn be exposed as a Web service. Access to and *communication* between WSs is done in terms of *messages* through *ports*. The messages are based on a single protocol over the Web; thus Web services utilize the characteristics of the *Internet*, namely simplicity and ubiquity. Web services have dual nature. On the one hand WSs are single *units of functionality*, and on the other hand they are represented by *a set of protocols* (SOAP, WSDL, UDDI) that are supported by a surprisingly great number of vendors and organisations to make any piece of functionality/application work together and interoperate independent of the hardware and software platform, with any other piece of application over the Internet.

2. WEB SERVICES VS. COMPONENTS

This section deals with the differences and similarities between Web services and components. One can recognize many similarities between Web services and components, but the differences might be useful for recognizing the potential behind the Web services technology.

A *component* is an executable unit of functionality. One can buy or download it, deploy it, and it works. It is a software black box, i.e. a *binary unit of deployment* [Szyperski, 1998]. Being a unit of functionality, Web services do not differ from the software components. However, certain tangibility characterizes the software components (the user downloads it and can "touch" it). Generally, there are several other features the software components exhibit.

Components are s*elf-describing* units. A component has an interface specifying explicitly what it does, encapsulating it, and separating it from other components. The idea behind the WSDL (Web Services Description Language) description of a Web service is basically the same.

A component is also s*elf-contained*. It is a packaged functionality that can be deployed and used as a real physical good. For communication a component connects to other components using *ports*. The Web services are also described in terms of ports, where messages arrive at or are sent from.

Both components and Web services are *registered* at a repository, to enable discoverability. The registration mechanism for Web services is based on the UDDI (Universal Description Discovery and Integration) specification, which is rather different from the one used for components. The UDDI registry itself contains only references to the location of the services and their schemata and not the interfaces themselves.

In order to communicate with each other and the environment components utilize a framework, making integration possible. The framework also provides additional services (lifecycle management, caching, transactions support, security, builder services, component flow coordination etc.) [Stiemerling, 2002]. Some of these services are either not yet developed for Web services or are considered irrelevant.

In general, middleware services for software components enable both synchronous and asynchronous interaction patterns. Components' *communication* is event-based, or implements method calls – RPC, RMI. Some models involve also the messaging mechanism. No (vendor-specific) component model implements all available communication mechanisms, but development in this direction continues. Web services on the other hand, use only the messaging mechanism. This implies the possibility of both synchronous and asynchronous communication, but only synchronous calls to Web services are implemented so far. The communication among Web services is based on the SOAP (Simple Object Access Protocol) standard, known as lightweight RPC protocol that encapsulates remote procedure calls and their parameters. Additionally, SOAP is used for document exchange between services. The possibility for document exchange is a significant difference that needs further consideration for achieving more efficient data exchange patterns.

There are component-based models that are both platform- and language-independent (JavaBeans); others are only language-independent (CORBA, DCOM) [Stiemerling, 2002]. Interoperability across organisations is fostered by each of these models provided the same vendor implementation is used. To make component-based applications from different vendors interoperate is however a difficult and tedious task. In comparison, Web services do not depend on any vendor-specific protocols, but rather on a single set of broadly accepted protocols. Moreover, the Web service functionality resides and runs at the provider's host, and users do not need to download them for local use. Web services are both language- and platform-independent.

3. WEB SERVICES VS. MIDDLEWARE

In the previous section we considered Web services in comparison with software components. Still, we must compare Web services and middleware.

It is well known that the process of implementing information systems spanning different enterprises comes across the heterogeneity and distribution problems. The usual way to solve these problems is by supporting standard programming *interfaces* to assure portability, and standard *protocols* to make programs interoperate [Bernstein, 1996]. The industry has already addressed these issues by developing so-called *middleware services*. These are distributed system services having standard programming interface and protocols. As their name implies, middleware services reside between applications on the one hand and the operating system together with the networking software on the other (see Figure 1).

In principle, what middleware services do is to replace the non-distributed functions of the operating system with distributed functions that use the network.

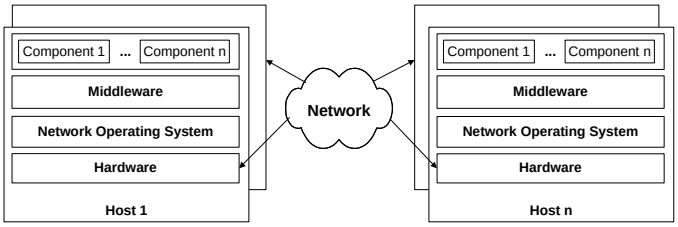

Figure 1. Middleware in distributed system construction [Emerich, 2000]

We see certain similarity between Web services and middleware. It is in the fact that Web services are considered a combination of standard protocols (HTTP, SOAP, WSDL and UDDI) and are accessible through a single, unified interface. This implies that Web services are not only the implementation itself, but rather the protocols and interfaces the Web service technology defines. Moreover, Web services were introduced with the intention to resolve heterogeneity problems in distributed systems, just like the middleware [Emerich, 2000]. These problems range from transformation between different encoding types, mapping object models and type systems, to tackling different binding and inheritance approaches. Web services approach these problems by mapping all kinds of data types to the XML Schema. Data exchanged among systems is encoded according to the SOAP encoding rules, whereas all format transformations take place behind the WSDL interface. The use of a single data format hides the heterogeneous features the environment exhibits.

A significant difference is however that Web services aim at preventing incompatibilities between middleware of different vendors. This is an advantage substantiated by the vendor- and platform-independent nature of Web services. The consequences of taking this advantage are crucial for application integration in distributed environments.

Distributed systems utilize middleware services for communication, coordination, control, transactional support, presentation management, information management, computation, and system management. Additionally, middleware ensures reliability, scalability, and performance to enterprise system. It also provides high-level primitives and thus hides the lower-level primitives of network operating systems and makes distributed systems design much easier and faster. In this respect, middleware services successfully tackle these problems; the Web services on the contrary, have a long way to go, in order to reach the same success. The basic principles in their future development should be to maintain the consensus among vendors, so that incompatibilities between implementations are avoided, and to leverage extensively the existing concepts and technologies, or at least the knowledge we gain out of them.

4. THE HYPERWARE

Web services are nothing revolutionary. They are the normal continuation of the development in computing technologies, and are practically based on component-based programming. Quite stunning however is the support for this technology, based on the acceptance of XML by all organizations. There are still missing components within the whole protocol stack, especially standard protocols for coordination, for transactional support and management of business processes.

So, are Web services middleware or software components? The middleware services are generic across applications and industries, they run on multiple platforms and are distributed, and support standard interfaces and protocols. The Web services also comply with this definition, excluding the fact that they are not yet accepted everywhere for critical implementations, reflecting the lack of maturity of the technology. The fact is Web services resemble middleware and are intended to perform the same functionalities all the sorts of specialized middleware do. Web services are also very similar to software components. They are separate implementation units providing characteristics that make integration of applications/systems belonging to different organizations easier. They are mostly based on software components, utilizing their interfaces to provide a single XML interface. Web services exhibit a certain shift in the way services are described and accessed. They complement the normal components to a more flexible technology facilitating integration of services over the Internet.

Given the above considerations we postulate that Web services are components (pieces of application, interface) and middleware (services, communication, security, discovery, compositions) in one.

Exhibiting characteristics of components, as well as of middleware, Web services have the potential to become the technology to solve the integration problem. Just like middleware resides between applications on the one hand and the operating system together with the networking software on the other, the same way Web services are layered between the applications running on one platform and the applications running on another platform, even across enterprises. In principle, what the middleware services do is to replace the non-distributed functions of the OS with distributed functions that use the network. Similarly, Web services are capable of replacing the distributed middleware services functionalities that typically use a LAN network with functionalities that use the Internet. After all the missing blocks are developed with respect to the Web services suite of protocols it might become evident that they are not simply services but an x-ware, say *hyperware* or extensible hyper-ware. This vision is expressed in Figure 2, but is not yet a reality.

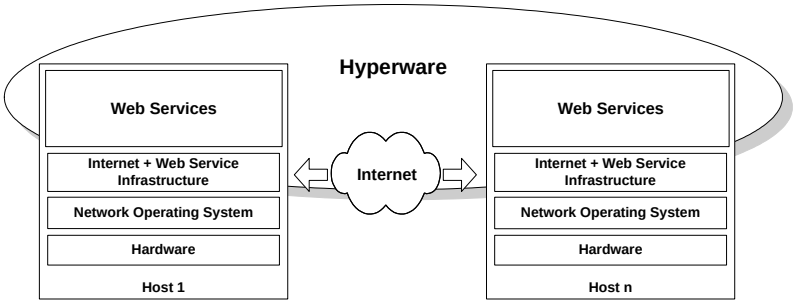

Figure 2. The hyperware

It would be beneficial for everyone to put more effort in developing the Web services technology reflecting the hyperware vision. Our contention is that this vision might provide a framework which would help reuse and leverage existing technologies, and foster interoperability among different computing models implementations. It would direct the development of the Web services paradigm in a unified manner, targeting an overall solution for integration. The hyperware idea might prove to be a starting point and a good reason for all vendors and standardising organizations for the process of assuring even greater consensus on a single flexible technology.

5. CONCLUSION

The main proposition of this paper is that Web services are components and middleware in one. Web services are considered of a dual nature being both protocols and physical services. For this purpose, a comparison of the basic Web services concepts with the concepts of the software components has been made, as well as with the basics of middleware. Our conviction is that at some future point in time Web services will grow into, or at least will be the basis of a new integration framework, called hyperware. This framework should support the WSs with services such as transactions, coordination and security.

REFERENCES

Bernstein, P., 1996. Middleware: A Model of Distributed System Services. *Communications of ACM.* Vol. 39, No. 2, pp 86-98.

Emerich, W., 2000. Software Engineering and Middleware: A Roadmap, *Proceedings of the conference on The future of Software engineering*, Limerick, Ireland, pp. 117-129.

Stiemerling, O., 2002. Web Services als Basis fuer evolvierbare Softwaresysteme, *Wirtschaftsinformatik 44*(2002) Vol. 5

Szyperski C., 1998. Component Software: Beyond Object-Oriented Programming. Addison Wesley