An overview of Service-Oriented/Aware Middleware

Martin Treiber and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology
[m.treiber|dustdar]@infosys.tuwien.ac.at

Abstract. In this paper we present an overview about existing Service-Oriented Middleware (SOM). We define Service-Oriented Middleware as middleware that follows the Service Oriented Architecture (SOA) paradigm. This implies that SOM must support the basic functions/features of SOA, i.e., the service brokerage, service discovery and service publishing. Furthermore, SOA middleware must provide for a loose coupling between the software components and support complex operations, for example the composition/orchestration of services. Using this definition of SOM, we compare existing middleware architectures according to their adequacy to our definition of SOM.

1 Introduction

Service orientation [1] has become a very important aspect of modern information systems. The SOA model with service brokerage, service provision and service discovery is a well accepted model for the design of distributed information systems. Since SOA is only an abstract paradigm for the creation of service oriented information systems, SOA is implemented on a range of different software architectures. For instance, is possible to create a SOA operating on a traditional client/server system where a server provides a set of services for clients.

The Web service paradigm [2] is the most prominent example for SOA. The W3C defines Web services as "[A Web service is a] software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-process able format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards".

Web services hide heterogeneity of different distributed systems with a set of wide accepted standards, such as UDDI [3] (discovery of services), WSDL [4] (description of service) and SOAP [5] (messaging, service invocation). Furthermore, Web services achieve loose coupling between different software components.

In contrast to SOA, middleware software architectures [6] have a long history. Modern middleware architecture has been developed since the 1980s when the first RPC based distributed systems were built. During the following years, the middleware software architecture has been extended form a mere remote procedure

calling supporting system to transaction processing and to message oriented middleware.

Middleware resides above the network operating system and below the application layer. Middleware provides the features to integrate distributed applications and services. Usually, middleware components provide value-added services such as naming and support for transactional processing.

It can be seen that middleware and SOA have some overlapping goals, both aim at

- integration of distributed software systems
- support of heterogeneity concerning implementation
- support of different (distributed) architectures

Nevertheless, SOA and middleware take different approaches, since

- Middleware operates on a more concrete level (e. g. message passing, message queuing, etc.)
- SOA provides "only" a conceptual model

As stated before, SOA and middleware nevertheless take approaches on different levels: SOA provides a high level understanding of services and therefore a coarse-grained approach for the integration of different information systems. Conventional middleware provide several features respectively components for the ability to integrate heterogeneous information systems. If we combine the approaches, we can benefit from several aspects:

- a coarse level of service provision (no object model details)
- reliable and flexible messaging between software components respectively services
- use of well adopted standards (WSDL, SOAP, etc.)
- extensibility

This allows us to classify SOM broadly as middleware architecture that is based on SOA concepts. To be more specific, middleware functionality is modeled after the SOA paradigm:

- middleware services are specified using standard (e. g. WSDL) service descriptions
- middleware services are loosely coupled
- middleware services communication uses standards such as SOAP for messaging

Thus, Service oriented middleware (SOM) is confined by two dimensions: the SOA dimension with its loosely coupled services (components), well defined standards and by the middleware dimension with its enterprise integration concepts of workflows, etc.

Using these two dimensions as confinement, we can think of Service-Oriented Middleware as "middleware that is implemented using SOA concepts and follows SOA standards".

The rest of the paper is organized as follows. Section 2 presents an overview of existing SO middleware systems. Section 3 compares these different approaches and Section 4 concludes the paper.

2 Web Service Oriented Middleware

This section provides an overview of existing middleware architectures and describes the architectures in greater detail.

2.1 Enterprise Service Bus

An enterprise service bus (ESB) describes a pattern of middleware that unifies and connects services, applications and resources within a business (see Figure 1). [7] defines "An Enterprise Service Bus (ESB) is a standard-based integration platform that combines messaging, web services, data transformation, and intelligent routing in a highly distributed, event-driven Service Oriented Architecture (SOA)".

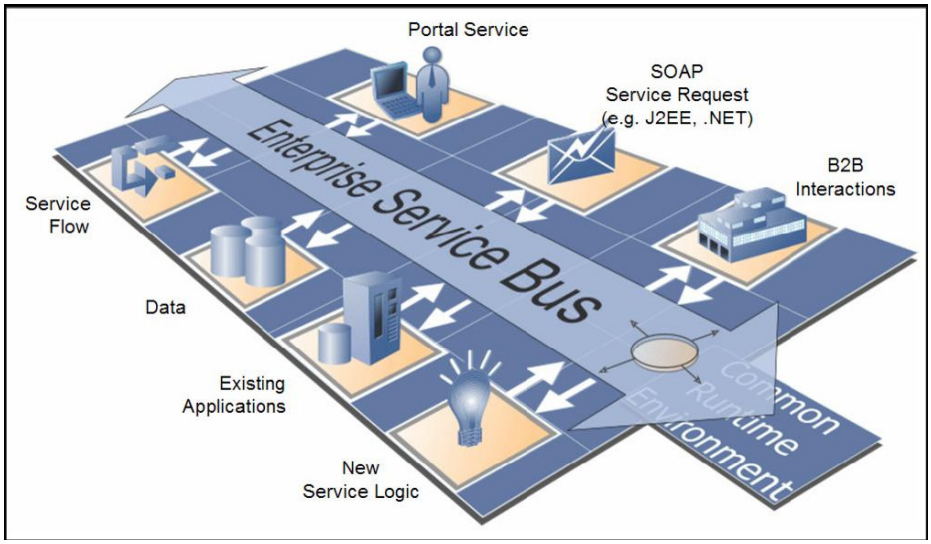

Fig. 1. Enterprise Service Bus

Several examples of the enterprise service bus patterns exist today; an example is Mule [23]. Mule is an open source framework that supports the ESB paradigm. Figure 2 shows the overall architecture of Mule.

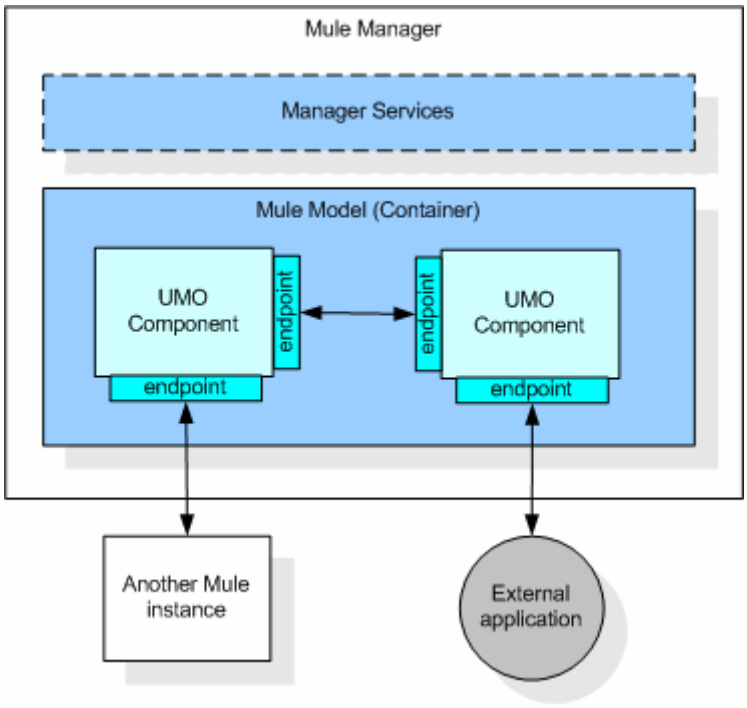

Fig. 2. Mule architecture

Every Mule instance provides a so-called Mule-Manager that controls the message flow between Mule components respectively other Mule instances. The messaging follows a channel based approach, where every message is placed as UMO (Universal Message Object) in a channel and delivered to the corresponding channel endpoints.

2.2 Service Component Architecture

IBM's Service Component Architecture (SCA) [8] uses the SOA model of service description. In contrast to Web services itself, SCA emphasis the assembly of existing applications in order to create respectively to compose new services. In that way, SCA complements Web services, because SCA supports the assembly of services and provides construction model for services.

SCA encapsulates (Web) services into container that can be customized using preferences and connectors. This results in a plug-able architecture, in which different software container can be plugged together to form services. SCA organizes services into modules, which is the main abstraction of SCA. These modules can be assembled to a complete system as depicted in Figure 3.

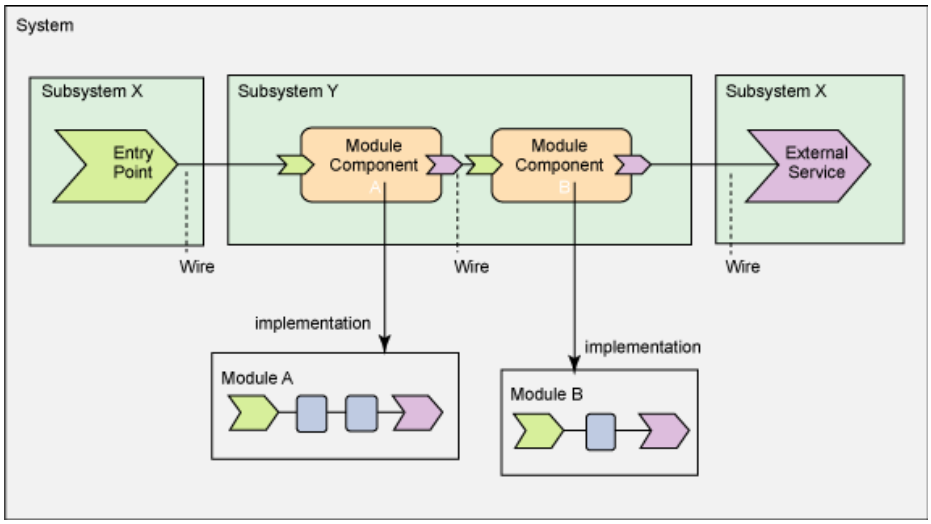

Fig. 3. Service Component Architecture overview

Using this abstract component approach, SCA allows for different implementations of components, for example, it is possible to implement a component in Java and other components in C++, etc.

2.3 Just in time middleware

The Just in time middleware architecture (JiM) [9] tries to refine the paradigm of middleware altogether: instead of providing a static monolithic middleware implementation, JiM aims at the generation of tailored middleware at design time. Thereby, JiM uses the implicit available knowledge of the application domain and the applications itself in order to create a tailored middleware application. JiM introduces four stages for the creation of middleware:

- **Explicit Acquisition.** Selection of required middleware features at design time.
- **Functional Inference.** Automated reasoning about middleware features to discover dependencies among requirements and possible extension of selected middleware features.
- **Verification.** Automated constraint check of selected middleware features and check for violations of feature constraints.
- **Synthesis.** Automated creation of the actual middleware software.

The outcome of these four stages is either a build configuration or an executable middleware instance. Zhang et. al. introduce a prototype called Abacus, a CORBA based middleware implementation. Abacus integrates Arachne, a tool that models the various Just-in-time customization steps. The tool is composed of three components: an aspect aware IDL compiler, a Java source parser and an inference engine.

2.4 Web Service middleware

The work in [11] introduces an architecture that supports specialization based on runtime access to service metadata. The overall architecture is composed of middleware components for service provider and web service requesters. Figure 4 illustrates the overall architecture of the proposed middleware.

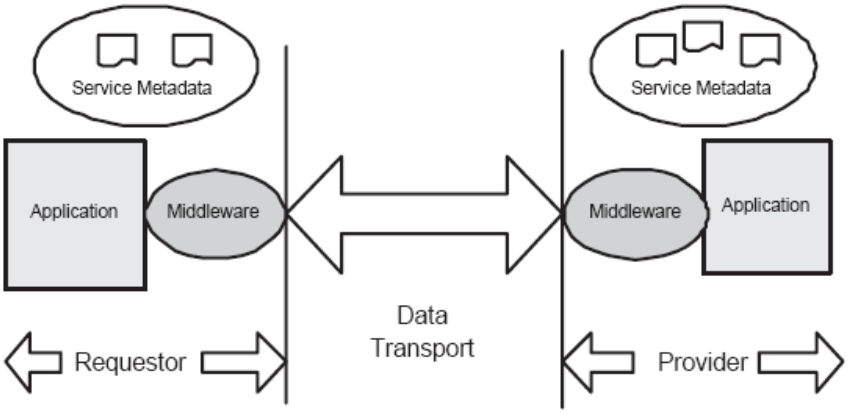

Fig. 4. Web service middleware

WSM provides a model that uses several metadata elements which describe different aspects of the service provider. The service requestor makes use of these metadata elements in order to select appropriate software components that provide a tailored middleware solution based on application requirements.

A prototype implementation of WSM is based on the Web Service Invocation Framework (WSIF) [20] and the Web Services Gateway [21]. The authors propose the use of so called message interceptors that provide well defined task and receive messages that flow from and to the application. The message interceptors can be composed into linear chains, depending on the requirements of the provider (Figure 5).

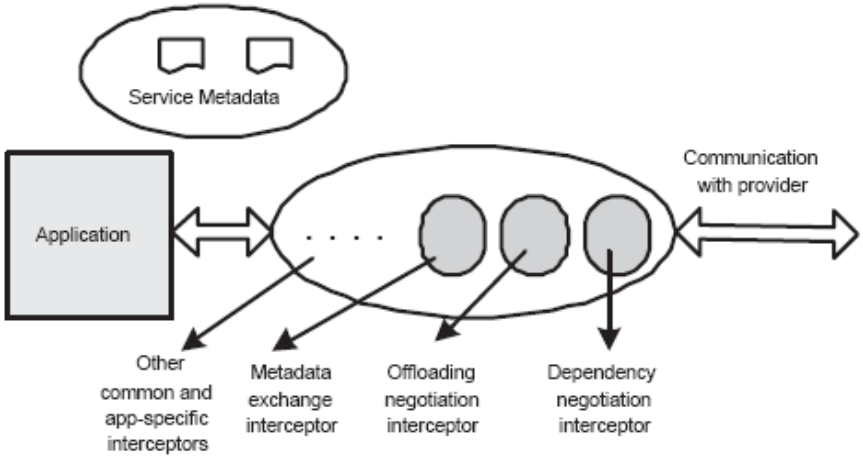

Fig. 5. Middleware architecture of provider

The service requestor uses WSIF to execute the corresponding services. The service provider uses the Web services gateway to route the messages to concrete services instances. Furthermore, interceptors are available for the interception of messages and provide therefore extensions for the available middleware.

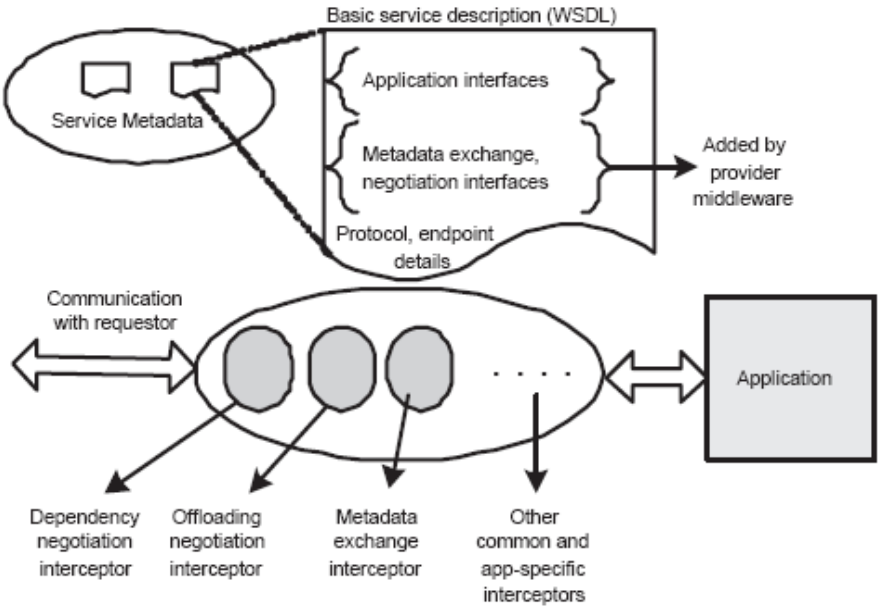

Fig. 6. Service requestor architecture

2.5 Smart Middleware

The approach taken in [12] introduces a P2P based service oriented middleware system. The middleware components are composed of different agents that perform activities on behalf of their peers respectively, their users. Peers are distinguished as fat peers or as thin peers. In contrast to thin peers, fat peers can handle XML based protocols, for example SOAP.

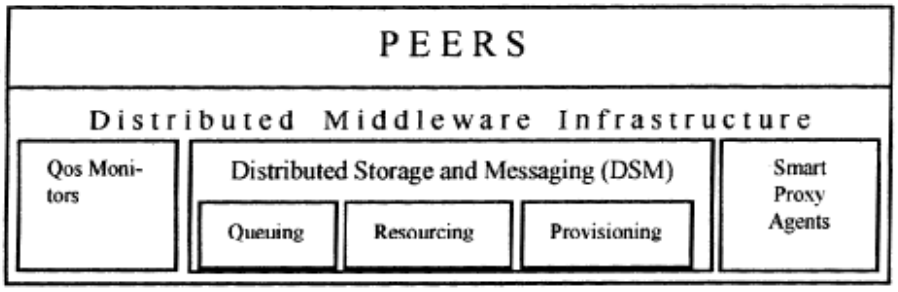

Fig. 7. Smart middleware architecture

The architecture of Smart middleware (Figure 7) is organized into three components:

- Data Storage and Message System (DSM). The DSM (see Figure 8) manages data from the different peers and delivers messages to the different peers. DSM uses message queues to support asynchronous communication patterns between peers.

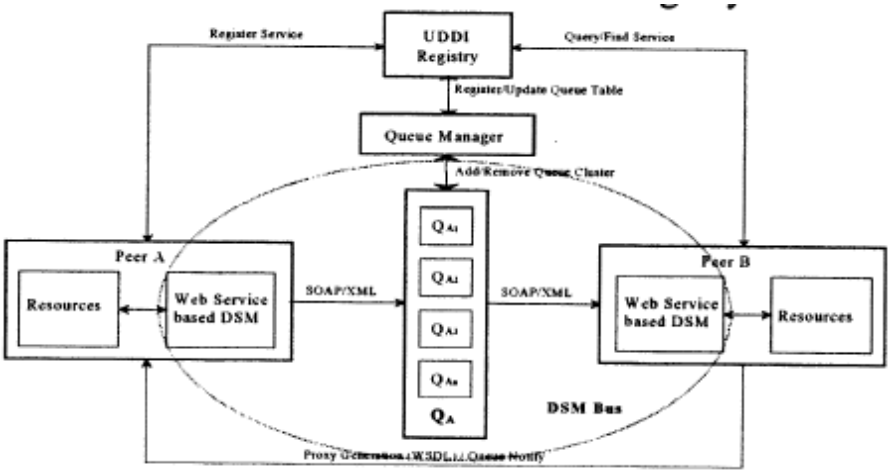

Fig. 8. DSM architecture

- QoS Monitors. QoS monitors monitor the usage of local resources at peers and provide QoS profiles at runtime.
- Proxy Agents (PA). A PA is responsible for the coordination of activities and data sharing between peers. PAs use policies (specified in Rei [13]) and information about available resources (described in OWL [14]) to coordinate between peers. Pas furthermore interpret the data supplied by QoS monitors and pool resources across multiple peers.

2.6 Grids

Grid architectures [10, 15] provide a well defined set of basic interfaces that offer the core functionality of grid based applications. In contrast to Web service architectures, grid based architectures provide an extended support for the life-cycle of services. Grids maintain state information and provide interfaces to access this type of information. Furthermore, grids define every available resource as service, for example storage devices, computers, sensors, etc.

2.6.1 OGSA

The Open Grid Service Architecture (OGSA) [19] marks the convergence between grid architectures and Web services. OGSA adopts Web service technologies (SOAP, WSDL) in order to create Web service compliant Grid services. OGSA services provide several WSDL port types:

- **GridService.** Defines the provided grid service specifies the interfaces a grid compliant service must implements (e.g. lifetime management, etc.).
- **Factory.** Creates grid service instances.
- **HandleResolver.** Resolves grid handles provided by service factories.
- **NotifcationSource.** Provides the means for registering of messages.
- **NotificationSink.** Delivers notification messages.

The Open Service Grid Architecture (OSGA) provides a basic service (the grid service) that is extended to actual services that access different resources (databases, CPU cycles, printers, etc.). The OGSA service model adds service management facilities (creation, monitoring, deletion, etc.) to the (SOA) service model. This enables OSGA to obtain runtime information about the service and provide additional information about the Web service. Furthermore, in contrast to the basic SOA service model, OSGA compliant services are stateful, i.e., OSGA is capable of managing state data during the execution of services. The overall architecture of OGSA is depicted in Figure 9. An implementation of OSGA is the Globus toolkit Version 4 [18] which implements the specifications of the WSRF [16].

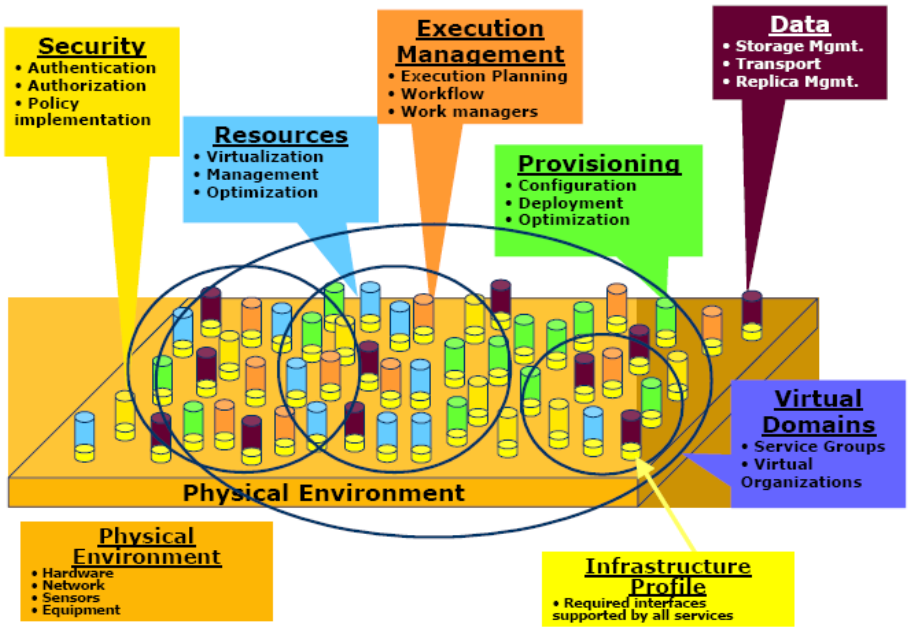

Fig. 9. OGSA architecture. Cylinders represent different grid services.

2.7 Message oriented Middleware

Tai et al. [20] present in their work the ratio for using Message oriented middleware (MOM) for communication between Web services. Their work illustrates the different approaches for the implementation of reliable messaging between Web services such as (i) Message Queuing Middleware (e.g. IBM Websphere MQ) or (ii) messaging standards (e.g. JMS).

Therefore, MOM covers only the message aspect of service oriented middleware, which is nevertheless the on key aspect of service oriented middleware. In SOM, MOM plays the role of message brokering and allows for a reliable and flexible message exchange patterns between Web service provider and Web service requesters.

3 Comparison

The comparison of middleware in the light of service orientation bears some difficulties. First of all, not all of the presented middleware solutions adopt the same (service oriented) technologies. Technologies range from reliable message exchange using message queues to BPEL execution engines. Furthermore, the scope and the

functionality of the middleware architectures differ. We decided to limit our comparison to the following attributes:

- **Support for Web services.** Web service support includes the use of standards such as SOAP, WSDL, UDDI, etc.
- **Support for decoupling of services.** Implicit support of service oriented software systems for decoupling of software components respectively services.
- **Support for dynamic adaptation**. The support for dynamic reconfiguration (tailoring) of available middleware components to different requirements of service providers during runtime.
- **Support of context data.** The support for meta information about the execution context of Web services.
- **Toolkit Support.** Provision of APIs and management tools for the middleware components.

Table 1 provides a comparison of the different approaches.

Attribute	Bus	SCA	Grid	WSM	SMW	JiM	MoM
Web Service standard support	No	No	Yes	Yes	Yes	No	Yes
Decoupling of services	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Dynamic adaptation	No	No	Yes	Yes	No	No	No
Context information	No	No	No	No	No	Yes	No
Toolkit support	Yes	No	Yes	Yes	Yes	Yes	Yes

Table 1. Comparison of Web service oriented middleware. The abbreviations are as follows: Bus=Mule, SCA=Service Component Architecture, Grid=OGSA and WSRF, WSM=Web Service Middleware, SMW=Smart Middleware, JiM=Just in time Middleware, MoM=Message oriented Middleware

4 Conclusion and outlook

The selection of “pure” service oriented or service aware middleware proved to be difficult. Current middleware approaches focus more on “traditional” aspects of middleware, like for example the support for generation of skeletons, stubs, toolkits, APIs, etc. The paradigm of service orientation as imposed by Web service standards is only partially supported by current middleware implementations, closest to being SOM are grid based solutions that follow Web service standards and provide mechanisms for the management of services and support for dynamic change in the selection of services.

References

[1] Wikipedia. Service-oriented architecture. http://en.wikipedia.org/wiki/Service-oriented_architecture. Accessed February 3rd 2006

[2] W3C. Web Services Architecture. http://www.w3.org/TR/ws-arch/. 2004

[3] OASIS TC. UDDI Version 3.0.2. http://uddi.org/pubs/uddi-v3.0.2-20041019.htm. 2004

[4] W3C. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl. 2001

[5] W3C. SOAP Version 1.2 Part 0: Primer. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/. 2003

[6] Gustavo Alonso, Fabio Casati, Harum Kuno, Vijay Machiraju. Web Services. Springer 2004

[7] David A. Chappell. Enterprise Service Bus. OReilly Assiociates, June 2004.

[8] BEA, IBM, Interface21, IONA, Oracle, SAP, Siebel, Sybase. Service Component Architecture. November 2005

[9] Charles Zhang, Dapeng Gao and Hans Arno Jacobsen. Towards Just-in-time Middleware Architectures. Proceedings of the 4th international conference on Aspect-oriented software development AOSD '05. 2005

[10] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, J. Von ReichThe Open Grid Services Architecture, Version 1.0. 2005.

[11] Nirmal K Mukhi. Ravi Konuru. Francisco Curbera. Cooperative Middleware Specialization for Service Oriented Architectures. WWW2004, May 17-22 2004

[12] Piyush Maheshwari, Salil S. Kanhere and Nandan Parameswaran. Service-Oriented Middleware for Peer-to-Peer Computing. 3rd IEEE Intemational Conference on Industrial Informatics (INDIN). 2005

[13] W3C. OWL Web Ontology Language Guide: http://www.w3.org/TR/owl-guide. 2004

[14] L. Kaga. Rei: A policy language for the Me-Centric project. http://www.hpl.hp.com/techreports/2002/HPL-2002-270.pdf. 2002

[15] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid - Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15(3), 2001.

[16] OASIS. WSRF Primer. http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-01.pdf Accessed February 20th 2006

[17] Global Grid Forum. Open Grid Service Architecture, Version 1.0. http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf

[18] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. IFIP International Conference on Network and Parallel Computing. 2005. LNCS 3779, pp. 2-13. Springer-Verlag 2005

[19] I. Foster, C. Kesselman, J. Nick, S. Tuecke . The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration., Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002

[20] S. Tai, T. Miklasen, I. Rouvellou. Using Message-oriented Middleware for Reliable Web Service Messaging. WES 2003. LNCS 3095, pp. 89-104. Springer-Verlag. 2004

[21] M. J. Duftler, N. K. Mukhi, A. Slominski, S. Weerawarana. Web Services Invocation Framework (WSIF). OOPSLA 2001 Workshop on Object-Oriented Web Services, October 2001.

[22] N. K. Mukhi, R. Khalaf, P. Fremantle. Multiprotocol Web Services for Enterprises and the Grid. Proceedings of the EuroWeb 2002 Conference on the Web and the Grid: From e-science to e-business, Oxford, UK, December 2002.

[23] Mule. Universal Messaging Objects. http://mule.codehaus.org/ Accessed February 20th 2006