

Scalability and Interoperability in Service-Centric Architectures for the Web

* Ralf-Dieter Schimkat¹

† Gerd Nusser^{1,2}

‡ Dieter Bühler¹

¹ Wilhelm-Schickard Institute for Computer Science
Symbolic Computation Group
University of Tübingen
Sand 13, D-72076 Tübingen, Germany
<http://www-sr.informatik.uni-tuebingen.de>

² Institute for Applied Research
University of Applied Sciences Reutlingen
Alteburgstr. 150, D-72762 Reutlingen, Germany

{schimkat, nusser, buehler}@informatik.uni-tuebingen.de

Abstract

The Internet provides a computing infrastructure that is highly distributed by nature. In the future the number of actively and cooperatively interacting parties in the Internet, such as traditional computers, embedded devices or various service providers in the field of electronic commerce, will increase even more. In such an environment, traditional client/server architectures reach the limit of their capability. Dezentralized, distributed software architectures which actively support the object and code migration paradigm, are more suitable.

We present a sophisticated approach to a scalable and interoperable service architecture for the Web. We propose a loosely coupled network of service nodes which supports scalability, integration and management of heterogenous services in an uniform way. Our proposed service infrastructure provides a platform to easily locate, connect, integrate and use services of any type. It addresses issues of bandwidth, security, distribution and integration, which are particularly important within the Web, at a general framework level.

* Supported by *debis Systemhaus Industry*

† Supported by the *Ministerium für Wissenschaft, Forschung und Kunst* of the state of Baden-Württemberg, Germany

1. Introduction

The Web is just about to evolve towards a single, world-wide service system from which users, and services as well, can obtain information and use services made available by the Internet, and on which business can be transacted in an uniform way. This necessitates logically integrating services from multiple organizations (e.g. service providers), often across company and location boundaries, into what appears to be a single service infrastructure.

Offering and consuming services in the World Wide Web has additional requirements on the underlying system infrastructure than in a local area network environment. In the near future a huge amount of embedded systems, such as PDAs, are going to take part in the World Wide Web. Such devices are characterized by their temporarily availability to the service infrastructure. This will considerably affect the fluctuating number of active resources on the Web. The wide range of different types of services, providers, and consumers have to be managed in an uniform and adaptable fashion. Network bandwidth, scalability, interoperability, mobility and distribution have to be carefully taken into account by designing sophisticated service infrastructures for the upcoming Web requirements.

Mainly due to scalability and interoperability issues we propose a strictly decoupled architecture which is based on a network of service nodes. In this system each service node

can initiate requests to other services similar to a multi-tier client/server architecture which is based on the traditional request-response model. Furthermore a service request can dynamically be propagated to further service providers. The involved services can act as servers or clients depending on the respective execution context.

Each service node is encapsulated into an uniform object-oriented interface which guarantees stable interfaces over time, even for future services which might be integrated into the system. Generally, aspects such as communication protocols and distribution of services are completely hidden behind uniform framework [11, 7] interfaces and generic XML-based [23] descriptions. The XML wave is coming because it provides a simple, standard, self-describing way of storing and exchanging information and services on the Web.

The rest of the paper is structured as follows: Section 2 presents the main components of the proposed architecture which is thoroughly discussed in the succeeding section. Then related work in the field of distributed service architectures is given. The paper ends with a short conclusion.

2. Architecture

In this section we discuss the main components, namely a service, a service node, and a service execution engine of our proposed software architecture to particularly address issues of scalability and interoperability in service-centric architectures for the Web. The major service components are implemented in Java [8].

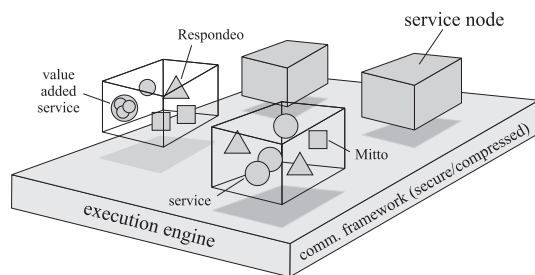


Figure 1. Architecture overview

2.1. Service, Service Nodes and Execution Engines

A service consists primarily of an XML [23] description of the service and a representation of the service itself, i.e. the object representing the service. The XML description includes meta information to identify a service, like for example the class, a symbolic name, and some service-specific properties. Furthermore XML is used to describe the interface and behaviour of a service. For example, a security

service offers the functionality to encrypt and decrypt information. Therefore, the XML part offers a human and machine readable description of the service and its peripheral services. Generally a service is a serialized object, which consists of an XML description and a serialized service object. A value-added service consists of a set of services as depicted in Figure 1.

In general, services can be distinguished between visible and invisible services. Visible services may have any kind of a graphical user interface or visible output. In contrast, invisible services offer their services without any direct user interaction.

One of the main characteristics of the proposed service architecture is that almost every active participant living in the system, is treated uniformly as a service. By establishing an abstract view on services, generic service interfaces can be introduced to the architecture. An interface strictly separates the specification of service functionalities (e.g. via XML) from their actual implementations. At this level several management facilities are provided which can be utilized by each participating service in an uniform and predictable manner.

Since a service or a service representation consists of an XML-based description and a serialized object, it is able to move and migrate among service nodes transparently. A service node contains a collection of services, as illustrated in Figure 1, and is responsible for the overall management. Each service node is attached to a lightweight execution engine which consists basically of a Java Virtual Machine (JVM). For example, any currently available browser (e.g. Netscape 4.x) or any embedded System with a small JVM (e.g. KVM [18], EmbeddedJava [17]).

There are three dedicated services called *Respondoo* [16], *Mitto* and *Jini* [6]. *Respondoo* provides special application and management services. *Mitto* serves as a general distribution service (see Section 3.5). *Respondoo* and *Mitto* are both lightweight such that they are small in code size and able to move among different execution engines. We have implemented a prototype for our proposed service architecture consisting out of the lightweight services *Respondoo* and *Mitto* where *Jini* serves as a general lookup service.

2.2. The Message Bus - MBus

Our service architecture includes a messaging component called *MBus* that serves as an uniformly accessible communication bus among service nodes. The architecture is characterized by the strict separation of service- and communication-related logic by wrapping services and service requests (actually any kind of service object) in standardized messages. Thus, the communication level of the proposed architecture has only to deal with messages regard-

less of their service specific content, i.e. the specific service logic. *MBus* which is introduced in [16] provides various kinds of communication channels which can be switched dynamically at run-time, e.g. the compression and the security channel. In the former one, all messages are compressed in order to reduce bandwidth. The latter one provides a secure data transfer on the message bus between services. For our prototype implementation we used the security service discussed in [9]. As far as the *MBus* is concerned, various communication channels can be aggregated and plugged together in order to build customized channels which communicate down a single network connection.

MBus is characterized by point-to-point communication (request/response) using Remote Method Invocation (RMI) or sockets as a general transport layer. The data delivery protocol can be synchronous or asynchronous depending on the service's profile. Each service request is wrapped in a message and transmitted to the destination service node.

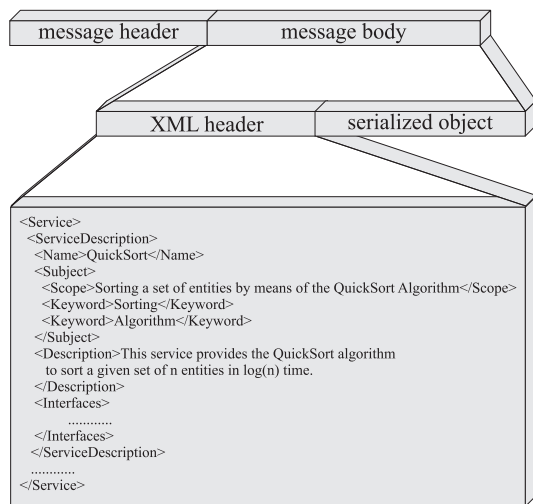


Figure 2. Extensible message layout

A message is composed of a header, a set of properties, and a body, as depicted in Figure 2. The header contains values used by services to describe the structure and the content of messages in a general manner. In addition, each message provides a set of optional and extensible properties which are used by services to specify dedicated service protocols by adding service-specific fields to a message. The body contains the data items (e.g. services) themselves, which can be of any type that is serializable in Java (e.g. XML-based service descriptions).

By using messages as the exclusive communication paradigm between services each call to remote functionalities or services is dispatched to the appropriate method call of the destination service node. Since the communication interfaces between services do not change over time regardless of the number and type of services which are or might be integrated in the service system, no additional commu-

nication-related code has to be provided, such as compiled static interfaces like stubs and skeletons. Instead, dedicated service protocols wrapped in messages are used as generic interfaces to initiate remote requests and to transmit the corresponding response back from the destination service node.

3. Discussion

In this section the proposed architecture is discussed by means of particular important issues in the World Wide Web context such as scalability, bandwidth, interoperability, mobility and distribution.

3.1. Scalability

The central issue of scalability in the Web is the system's ability to support both growing numbers of services and of various service types. Below we will focus on some important design considerations which support scalability in different contexts.

System Service Scalability The overall number of services connected concurrently and acting as active participants in our service architecture is not predictable and varies from time to time. We therefore propose a loosely coupled network of service nodes with the standardized *MBus* in between. This approach has the following major benefits:

Since each service node manages a collection of services, service requests are mediated through a service node. Thus the initiator and recipient of service requests are completely decoupled from each other. A service node can control the overall workload in a manageable manner which is similar to the role of the middle-tier within multi-tier client-server architectures.

The *MBus* provides an uniformly accessible communication channel which is completely independent from any specific service representation. Therefore, the interaction of multiple services through dedicated service protocols which are wrapped in general messages, makes no assumptions about the communication-related code of services apart from the common *MBus* interface. This scalable approach enables the access to any collection of service representations at each service node.

Service Type Scalability The scalability issues related to the potentially huge number of different service types, such as printers, robotics, and databases are crucial to our service infrastructure. The point to note here is that the service node provides the right place to encapsulate different types of services and to give them an uniformly accessible and manageable interface. Thus, our infrastructure solves the

service type-related scalability issues at a higher level of abstraction:

Each service is encapsulated within general service interfaces which hide the different kinds of service types and their implementations respectively. Each service contains an XML-based description of itself.

For each service, a pool can be configured which is managed by the service node. A pool basically synchronizes the access to and manages the pooling of service objects, i.e. connection pooling for fast and efficient handling of database connections.

Various configuration policies can be applied even at run-time to each pool of service objects in an uniform manner since the architecture strictly separates between the management of services and their implementations.

In general, solving scalability issues at a higher level of abstraction leads to an exhaustive reuse of the provided middleware services and speeds up the overall integration process tremendously.

3.2. Bandwidth

The issue of limited bandwidth concerns network and Web applications in particular because the reliability and capacity of the underlying network is not sufficient in the common case. Thus, one goal is to keep the size of the transmitted data as small as possible by providing adequate data encoding schemes. When applying sophisticated data compression schemes, the availability of local resources for the encoding and decoding of the data has to be kept in mind. In general, since there is a tradeoff between managing bandwidth and the availability of local resources, a flexible strategy has to be provided which can be changed at run-time. For example, when the volume of the data is large and local decoding resources are available, apply adequate encoding mechanisms. Especially on the Web, the additional time for the decoding of the transmitted data has only little effect on the overall communication time. When the volume of the data is small, it is transmitted without any additional encoding.

Generally, scalability affects the limited bandwidth problem directly due to the increased volume of data, since an increasing number of services demands an increasing throughput of service calls. Our infrastructure provides a solution to these problems by encapsulating the entire communication process and data transfer within the uniformly accessible *MBus* which offers a compressing, a secure and a default communication channel for transmitting messages (e.g. services). The strategy for choosing the appropriate communication channel is exchangeable at run-time.

3.3. Interoperability and Flexibility

One of the main benefits of the proposed system is the mediation between almost arbitrary hard- and software entities. For instance, it is possible to integrate hardware access into the service system by encapsulating legacy systems or dedicated hardware gateways (like the one presented in [4]) into a service object. In a CORBA architecture this would be done via rigid, binary IDL interfaces which have to be defined at compile time. Describing the interfaces and the semantics of our services in XML and not in a binary format is a more flexible and dynamic approach. It allows fine grained lookup and assessment of services at runtime. Since no IDL compiler is needed this description is completely independent of the target programming language and the descriptive power of the IDL.

Furthermore services offered by small embedded systems which offer no CORBA support by themselves can easily be integrated into the system in a similar way like this is achieved by Jini. But unlike Jini our service description is done in XML and thereby offers a more powerful, dynamic and extensible description framework.

Since a flexible interoperability of a wide variety of services providing access to a wide variety of resources is crucial to our system, we decided to accept the overhead of having to decode the service description on the consumer side which has only to be performed at the first service contact.

3.4. Mobility

The most popular paradigm for structuring distributed applications is the traditional client-server model primarily used by CORBA [13] environments. It is based on the remote method or object invocation mechanism [2] which basically makes the interaction of components (e.g. services) independent of their locations (location transparency). In [22] it is discussed that there are cases, especially in large-scale computer networks, where interaction among components residing on remote hosts is rather different in terms of latency and access to services. Hiding such differences might end up in unexpected performance problems. Therefore the location of components and services in wide-area networks as the Web has to actively taken into consideration as opposed to service systems as CORBA which regards the location of services and objects just as an implementation detail.

According to Carzaniga et al. [5] mobility is the “capability to reconfigure dynamically, at run-time, the binding between software components (e.g. services) of the application and their physical location within a computer network” such as the Web. Our proposed service architecture supports the paradigm of mobile code in order to provide a sophisti-

cated distributed infrastructure which is capable of moving code and of migrating services among service nodes.

The following benefits for our service architecture derive from the paradigm of mobility (mobile code):

There are cases where the number of remote communication calls and interactions among services is reduced remarkably.

In order to support a maximum number of services for the Web, our proposed infrastructure does not predetermine and restrict the location of services. Thus each service can move dynamically to other locations while adapting to the current state and context of a service node. In addition such a mobile service can decide by its own what strategy for migrating to another location is the best suited. *Mitto* and *Respondeo* are designed as mobile services as well.

A service can be replicated at run-time by serializing it. The service keeps its current state and can make itself persistent to permanent storage. The replication of services while keeping their current state and the support of mobile code make the service architecture capable of placing a valid and active copy of services closer to the location of a service consumer. Therefore by establishing a kind of service caching, the overhead introduced by remote communication is reduced significantly.

3.5. Distribution

Distribution of services and of service requests is a major challenge in designing scalable, distributed service infrastructures. As far as services are concerned, they need a flexible way to migrate and move among service nodes. A major concern is the decoupling of service providers and consumers in order to support scalable service interactions.

In our proposed service architecture, distribution is accomplished by a lightweight, message-oriented middleware component called *Mitto*. *Mitto*'s primary communication paradigm is based on the publish/subscribe mechanism as opposed to the traditional request/response protocol. *Mitto* is lightweight by the means of that it is small in code size and only provides a subset of the functionalities specified in the Java Message Service [10]. Therefore it is a suitable service component for the Web since it can be moved among service nodes similar to common services. *Mitto* provides an object-oriented notification infrastructure for supporting recipients in the interpretation of the received message (event) semantics. For example complete code fragments can be included in the event notification as opposed to CORBA. This makes any additional handshake with the sender of the event needless.

Mitto does not provide any subscription language, since it needs to be scalable in terms of services and service types. In practice, every client can have its own dedicated filter which is dynamically uploaded to *Mitto*. Additionally any

type of message and event, i.e. XML-messages, can be transmitted as long as it is serializable in Java. Finally *Mitto* makes a draft on the flexible communication facilities provided by *MBus*. In general, *Mitto* serves just as a lightweight execution environment for distributing events.

4. Related and Future Work

Jini Aschemann et al. [1] propose a framework for integrating legacy devices into a Jini-enabled [6] management environment. They argue that their framework is applicable to a large number of different devices and communication protocols.

Similar to our proposed service architecture there are standardized management interfaces for Jini-enabled devices. However, they do not provide a highly interoperable approach based on XML to the composition of services and service descriptions to easily build value-added service groups. In addition there is no strict separation of application- and communication-related logic of services supported. Thus each service type has its own communication-specific logic as opposed to our architecture which clearly address scalability and bandwidth issues of the Web at a complete distinct layer (e.g. the service bus). Furthermore Aschemann et al. do not discuss sophisticated distribution mechanisms which are crucial in a Web-based computing environment.

CORBA As far as interoperability and scalability of services in the Web are concerned, we argue that CORBA [13] does not provide the appropriate platform since it does not take the specific requirements of a massive Web-based service computing into account:

CORBA lacks a true mobile code paradigm since it does not support a true call-by-value mechanism for complex, user-defined objects as discussed in [21]. Conceptually it is extremely difficult because CORBA systems allow different implementation languages on different sites, making different object representations unusable [3].

It is a heavyweight infrastructure which can not be easily integrated into a computing environment such as the future computing network built out of various different types of embedded devices. The execution engines of CORBA are not capable of migrating.

CORBA restricts the level of interoperability to the object level by specifying interfaces according to the Interface Definition Language (IDL). XML is better suited for a general concept of self-describing services than IDL.

Virtual Java Devices The Virtual Java Devices, introduced in [12], transparently interface to real embedded devices by wrapping them into CORBA making them accessible via the Web through Java applets for instance.

Conceptually CORBA is used to connect to and access different kinds of real devices in an uniform and platform-independent way. However, an interoperable view of the Java Virtual Device as a more general system service within a highly distributed service environment is not provided.

Globe Globe [20, 19] is an object-based framework for developing wide-area distributed applications resulting in scalable and flexible Web documents. It focuses on objects, their interfaces and replication mechanisms within a distributed environment like the Internet. Objects in Globe can be defined with the Globe Interface Definition Language which is closely related to the CORBA IDL.

In contrast to Globe, we focus on the concept of a service rather than on an object. We also propose to shift the level of interoperability from the object level (by defining IDL interfaces) to the service level (by specifying XML definitions) which offers a greater potential for future interoperable service architectures. In addition we incorporate services and services nodes directly in current Web architectures as we provide mobile, Java-based framework and system services which can dynamically migrate and move through the Web. In contrast, Globe is integrated into the current Web infrastructure (e.g. browser, http-protocol) by providing a kind of gateway technology which makes Globe objects respectively accessible in the Web.

Future Work Our future work will extend our prototype implementation by integrating various different services. Additionally work is going on in order to integrate mobile and stationary agents, which are discussed in [15], as common services into our system. Furthermore, research work is undertaken to provide a lightweight graphical user interface (GUI) engine which can serve as an adaptable front-end for services. Therefore we adopt the ideas which are presented in [14].

5. Conclusion

We proposed a service architecture for the Web which is mainly characterized by its ability to face the challenges introduced by the increasing demands of the future Web. We argued that XML-based service descriptions accompanied by a uniform accessible and manageable communication bus fulfils the future Web requirements of scalability and interoperability among services in a generic and dynamic manner. The prototype implementation is a first step towards the realization of this system and forms a basis for our future work.

References

- [1] G. Aschemann, S. Domnitcheva, and P. Hasselmeyer. A framework for the integration of legacy devices into a Jini management federation. In *10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'99)*, number 1700 in Lecture Notes in Computer Science. Springer-Verlag, 1999.
- [2] A. Birell and B. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1), Feb. 1984.
- [3] M. Boger. Migrating Objects in Electronic Commerce Applications. In W. Lamersdorf and M. Merz, editors, *International IFIP/GI Working Conference (TREC'98)*, number 1402 in Trends in Distributed System for Electronic Commerce. Springer-Verlag, 1998.
- [4] D. Bühler, G. Nusser, G. Gruhler, and W. Küchlin. A Java client/server system for accessing arbitrary CANopen fieldbus devices via the Internet. *South African Computer Journal*, (24):239–243, Nov. 1999.
- [5] A. Carzaniga, G. Picco, and G. Vigna. Designing distributed applications with mobile code paradigms. In *Proceedings of the 19th International Conference on Software Engineering*, pages 22–32, May 1997.
- [6] W. Edwards. *Core Jini*. The SUN Microsystems Press Java Series. Prentice-Hall, Inc., 1999.
- [7] M. Fayad and D. Schmidt. Object-Oriented Application Frameworks. *Communications of the ACM*, 40(10), Oct. 1997.
- [8] J. Gosling and K. Arnold. *The Java Programming Language*. Addison-Wesley, Reading, Massachusetts, 1996.
- [9] G. Gruhler, G. Nusser, D. Bühler, and W. Küchlin. Tele-service of CAN systems via internet. In *Proc. of the International CAN Conference (ICC)*, Torino (Italy), Nov. 1999. CAN in Automation (CiA).
- [10] M. Hapner, R. Burrige, and R. Sharma. *Java Message Service*. SUN Microsystems, JavaSoft, Nov. 1999.
- [11] R. Johnson and B. Foote. Designing Reusable Classes. *Object-Oriented Programming*, 1(2):22–35, 1988.
- [12] T. Lumpp, G. Gruhler, and W. Küchlin. Virtual Java devices – Integration of fieldbus based systems in the Internet. In *Proc. of the 24th IEEE International Conference on Industrial Electronics, Control and Instrumentation (IECON '98)*, pages 176–181. IEEE Society Press, Sept. 1998.
- [13] Object Management Group (OMG). *The Common Object Request Broker: Architecture and Specification*, June 1999. Revision 2.3, available at <http://www.omg.org/docs/formal/98-12-01.ps>.
- [14] R. Schimkat and W. Küchlin and R. Krautter. An object-oriented framework for rapid client-side integration of information management systems. *South African Computer Journal*, (24):244–248, Nov. 1999.
- [15] R. Schimkat, W. Blochinger, C. Sinz, M. Friedrich, and W. Küchlin. A service-based agent framework for distributed symbolic computation. In *Proceedings of the 8th International Conference on High Performance Computing and Networking Europe (HPCN)*, number 1823 in Lecture Notes in Computer Science, pages 644–656, Amsterdam, Netherlands, May 2000. Springer-Verlag.

- [16] R. Schimkat, S. Müller, W. Küchlin, and R. Krautter. A Lightweight, Message-Oriented Application Server for the WWW. In *Proceedings of the 15th ACM Symposium on Applied Computing (SAC)*, pages 934–941, Como, Italy, Mar. 2000. Association for Computing Machinery.
- [17] Sun Microsystems, <http://www.sun.com/software/embeddedjava/>. *EmbeddedJava Technology*.
- [18] Sun Microsystems, <http://java.sun.com/products/kvm/wp/>. *The K virtual machine (KVM) - A white paper*.
- [19] M. van Steen, P. Homburg, and A. Tanenbaum. Globe: A wide-area distributed system. *IEEE Concurrency*, pages 70–78, Mar. 1999.
- [20] M. van Steen, A. Tanenbaum, I. Kuz, and H. Sips. A scalable middleware solution for advanced wide-area web services. *Distributed Systems Engineering*, 6(1):34–42, Mar. 1999.
- [21] S. Vinoski. New features for CORBA 3.0. *Communications of the ACM*, 41(10), Oct. 1998.
- [22] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A note on distributed computing. Technical Report TR-94-29, SUN Microsystems Laboratories, Nov 1994.
- [23] World Wide Web Consortium (W3C), <http://www.w3.org/TR/REC-xml>. *Extensible Markup Language (XML) 1.0*, 1998.