February 2005

Service-Oriented Architecture Supporting Mobile Access to an ERP System

Anna Maria Jankowska
European University Viadrina, Frankfurt (Oder), Germany

Karl Kurbel
European University Viadrina, Frankfurt (Oder), Germany

In: Ferstl, Otto K, u.a. (Hg) 2005. *Wirtschaftsinformatik 2005: eEconomy, eGovernment, eSociety*; 7. Internationale Tagung Wirtschaftsinformatik 2005. Heidelberg: Physica-Verlag

ISBN: 3-7908-1574-8

Service-Oriented Architecture Supporting Mobile Access to an ERP System

Anna Maria Jankowska, Karl Kurbel
European University Viadrina, Frankfurt (Oder), Germany

Abstract: With the emergence of Web Services application vendors and organizations with heterogeneous software architectures have started to move towards Service-Oriented Architectures (SOAs). In a SOA, software functionalities are represented as discoverable services that are accessed through a network. SOA is a promising approach for Enterprise Application Integration problems. As computing becomes ubiquitous and users are supported by a wide range of mobile devices, enterprises have to think about integrating mobile clients into a SOA. We introduce an architecture that supports communication between mobile devices and Enterprise Resource Planning (ERP) systems equipped with a Web Services Façade. Theoretical foundations of Web Services and SOA and a prototypical implementation of mobile Web Services for an ERP system are discussed.

Keywords: Service-Oriented Architecture, Web Services Façade, mobile devices, ERP system

1 Introduction

In the past few years, the Internet and the concept of Web Services have brought forth new approaches to information systems (IS) design and new integration technologies in Enterprise Application Integration (EAI). Large companies such as SAP, Dell, General Motors, Nasdaq, and Tesco [Smol03] began to re-design their systems, implementing solutions that are loosely coupled and interoperable. Software and consulting firms followed this trend, proposing information systems burdened with many layers, interfaces and proxies. Although already known for a decade, Service-Oriented Architectures (SOAs) suddenly gained great popularity, due to the increased interest in Web Services. This concept has since been applied even for solutions that are not meant to be exposed as services for external use.

Key business applications in most enterprises include Enterprise Resource Planning (ERP), Supply Chain Management (SCM) and Customer Relationship Management (CRM). ERP systems are the IS backbone of many organizations. They are usually very complex, supporting all major business processes. Most organizations use software packages from vendors like SAP, PeopleSoft, Oracle, etc.

To integrate ERP systems with external applications, ERP vendors provide Application Programming Interfaces (APIs); include low-level programming languages in their software packages; and sometimes disclose the underlying data schema. However, using public APIs and metadata is not a satisfactory solution to the integration problem as high costs and severe maintenance problems may result [Puli03, p.48]. Industry standards such as the Common Object Request Broker Architecture (CORBA) [Gros01, pp. 449-469], Microsoft Distributed Component Object Model [Redm97], and XML provide interfaces on a higher level. Nevertheless, change management in EAI projects is a cumbersome process. Developers with specialized EAI package skills maintaining integrated applications often have a "job for life".

Although ERP is essentially an in-house application, more and more external users need access to such a system. In supply networks, for example, customers and suppliers may be granted access to ERP information. Sales representatives visiting customers and employees traveling may need to know the status of an order, an inventory level, or reconciliation of an invoice.

Web Services are a promising approach to addressing integration issues. Web Services are software components that support distributed computing using standard Internet protocols. They are self-contained, self-describing, and modular. They can be published, located and invoked across the Web. Since Web Services are loosely coupled they are suitable for organizations with a complex architecture comprising multiple information systems, multiple platforms, different object models, and different programming languages. The Web Services approach lends itself naturally to incorporation in the Service-Oriented Architecture paradigm. Information system architectures can be re-designed in this way, and new applications can be assembled from services with suitable functionalities provided by different software vendors [ChaJo03, p.14].

Mobile computing has improved the quality of doing business today [CaKo04]. Permanent access to information not only helps individuals but also allows firms to perform their tasks in a more efficient manner, reducing operational costs. Vendors of ERP systems such as SAP or infor AG have recognized the potential of mobile technology and extended their systems with mobile front-ends in order to meet new requirements [SAP04, Info04]. Mobile interfaces have resulted in significant savings and improvements in customer service, not only in large companies but also in small ones [Wass03]. Some firms were able to almost eliminate duplicate acquisition of data by sales representatives and office employees, and to decrease the error ratio significantly.

ERP vendors are nowadays moving to Service-Oriented Architectures built on Web Services [Smol03, KeRo03]. Therefore connecting mobile devices to such component-based information systems becomes a crucial issue. The need to adopt Web Services in the mobile domain was recognized by the Open Mobile Alliance (OMA). This organization, incorporating over 300 companies, created in 2003 a

Mobile Web Services Working Group that deals with standards for mobile access to Web Services interfaces [OMA04].

This paper describes architectural approaches that allow mobile devices to communicate with ERP systems that possess a Web Services Façade. It is organized as follows: In the next section, Web Services standards and application types of Web Services are presented. Section three provides an overview of the Service-Oriented Architecture and introduces the steps towards a complete SOA solution. The subsequent section focuses on mobile Web Services and their advantages. Section 5 presents a complex architecture which allows different mobile clients to access an ERP system built on Web Services. It also introduces the solution proposed by SAP for mobile access to its ERP system. In the final section some conclusions are drawn and issues for further research and development are discussed.

2 Web Services, Styles, and Applications

The W3C Web Services Architecture group defined a Web Service as a "software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artifacts. It supports direct interactions with other software applications using XML-based messages via Internet-based protocols" [W3C04b]. Web Services offer mechanisms for building interoperable, distributed, platform and language-independent applications. Although the concept of Web Services is often called "a new dressing for the old distributed-computing model", it gathered broad attention due to the wide acceptance of the underlying technologies.

The Web Services framework specifies communication methods between distributed components, enabling them to use each other's functionality via Internet. In terms of the TCP/IP reference model, the Web Services layer is located between the transport and application layers as shown in figure 1. The entire Web Services infrastructure is based on XML standards: Simple Object Access Protocol (SOAP) [W3C03], Web Services Description Language (WSDL) [W3C04c], and Universal Description, Discovery, and Integration (UDDI) [OASI03].

SOAP defines a common syntax for data exchange assuring syntactic interoperability. Any Web application, independently of the underlying programming language, can send a SOAP message with the service name and input parameters via Internet and will in return obtain another SOAP message with the results of this remote call.

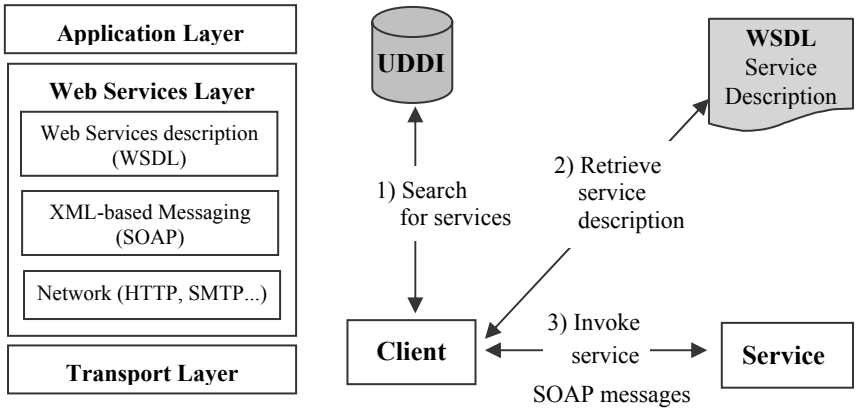

Figure 1: Web Services protocol stack and framework

Web Services can be implemented as either an RPC-style Web Service or a document-style Web Service [ChaJe02, pp. 28-34]. RPC-style Web Services expose the server-side functionality as a remote object typically accessed via a local proxy object on the client side. In a document-style Web Services the service uses the entire body of a SOAP message and parses it as a standard XML document. Table 1 compares the two styles.

	RPC-style Web Service	**Document-style Web Service**
Message body	Treated as a collection of parameters	Processed as a document
Processing model	Parameters mapped to native data structures	XML parsed according to an external XML Schema
Invocation mechanism	Method called on local proxy object (stub), routed directly to service interface	Document published to server for processing

Table 1: Comparison of Web Services styles

WSDL is used for the description of Web Service interfaces. It specifies input and output parameters, the structure of functions and the service's protocol binding. UDDI serves as a mechanism to discover and locate available Web Services. UDDI registries are databases containing contact, business and technical information about registered Web Services. Any organization may look in a public registry [Micr04a; IBM04] using a SOAP call and will obtain a list of services that meet the given criteria. For internal purposes, some companies create their own UDDI registries accessible only for architects and developers of that company.

Web Services are used for the following reasons [cf. Asto03]: To enable real-time interoperability of applications, to aggregate capabilities of particular applications into business services exposed for internal/external usage, to assure architectural agility, and to provide better self-service for employees, partners and customers.

The driving force for redesigning information systems towards SOA is Enterprise Application Integration (EAI). EAI means unrestricted sharing of data and processes among connected applications or data sources in the company [Lint99, p. 3]. Based on Web Services data from various systems can be consolidated into a single view. Aggregating functionality within a business service means that other applications can use that service without understanding its internal complexity.

In Enterprise Resource Planning multiple front-end systems (mobile applications, Web-based programs, traditional GUIs, etc.) should have the possibility to interact with the ERP system. Consider, for example, the entering of customer orders. This can be done manually, or via separate applications for all possible communication channels through which orders can arrive (mobile devices, Web interfaces, e-mail, etc.). Instead of such a collection of disparate solutions, a common Web Service for order processing might be built. Each channel would then communicate with the Web Service and simply pass order details to it for further processing.

Web-Service enabled solutions may increase the agility of organizations - they can react faster to technological or business changes. For example, if a company purchases another one, it can quickly integrate IT solutions of the acquired firm with its own IT landscape even if the underlying systems differ significantly.

Furthermore, corporate identity may be improved by wrapping the functionality of all applications in Web Services and exposing them to a common front-end application. Companies can enhance their service quality or improve their opportunities to create revenues by providing access to their information systems for clients or suppliers. For example, concepts from Supply Chain Management like Vendor Managed Inventory (VMI) and Collaborative Planning, Forecasting and Replenishment (CPFR) can be supported by Web-Service interfaces, giving partners in the supply chain information about forecasts of delivery dates, enable real-time invoicing, or allow suppliers to re-schedule deliveries.

3 Service-Oriented Architecture

Service-Oriented Architecture is a concept that focuses on configuring entities (services, registries, contracts and proxies) in a way that maximizes loose coupling of components and their reuse [cf. McGo+03, pp. 35-38]. In SOA software functionality is represented by discoverable services on the network. A service is defined as behavior that is provided by a component and can be used by other components, whereby the interaction is based on the interface contract. SOA consists

of the following six entities: service consumer, service provider, service registry, service contract, service proxy, and service lease.

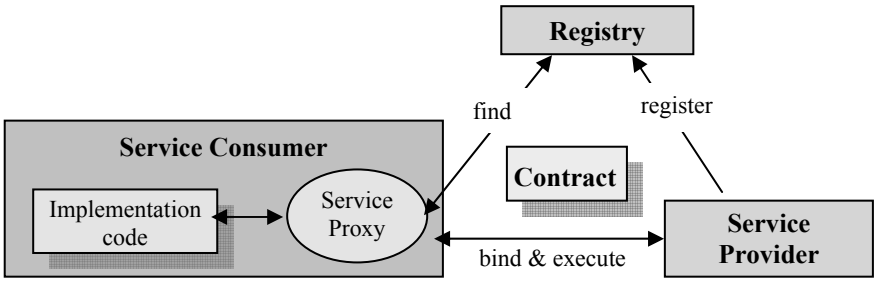

Figure 2: Main components of SOA, "find-bind-execute" paradigm [McGo+03, pp. 37, 39]

The *service consumer* is some kind of software module (e.g. application, another service) that works according to the "find, bind and execute" paradigm shown in figure 2. The consumer initiates the process of locating a service in the registry, binding to the service, and executing its function. The *service provider* is a network-addressable component that publishes its contract in the registry and responds to customers' requests. A *service registry* is a directory that stores contracts with providers and displays them to customers. A *service contract* is a specification describing the interactions between a service provider and a consumer. It discloses the format of requests and responses and may also specify pre- and post-conditions for service execution or quality of service (QoS) levels. *Service lease* restricts the time for which a contract is valid. *Service proxy* is an additional entity that helps the consumer execute a service by calling a proxy function instead of accessing the service directly.

In SOA services should be loosely coupled, self-contained, modular, discoverable, dynamically bound, composable, and location-transparent. Dynamic discovery, binding and location transparency refer to the ability of a consumer to locate and execute a service without a-priori knowledge about the service. Modularity means that services can be aggregated into an application with a limited number of well-known dependencies. A service is called self-contained if its functionality is limited to a distinct problem domain function. Services are interoperable - they support different platforms and languages. In addition, they should have coarse-grained and network-addressable interfaces. Coarse-grained interfaces address functions of many different APIs that enclose detail-oriented methods into a small number of business-oriented messages [see McGo+03, pp. 50-59]. Systems based on SOA are self-healing – they can recover from errors without human interventions.

All these features are aiming at business agility – the ability of an enterprise to respond quickly and efficiently to change. Further benefits from implementing SOA encompass faster time-to-market, reduced costs and risks, introduction of a process-centric architecture and leverage of previous investments.

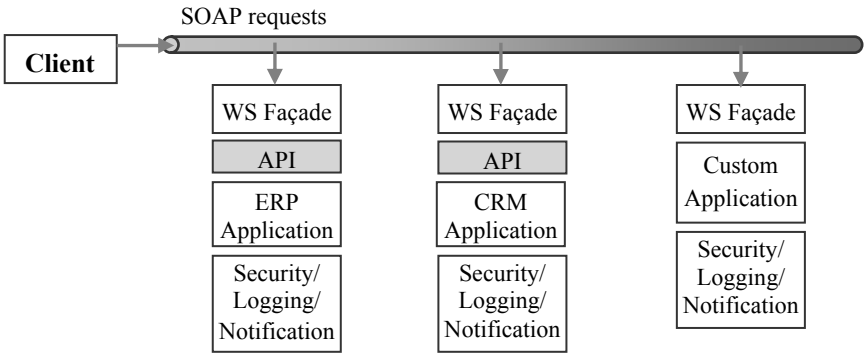

Figure 3: Web Services Façades

The transition from traditional, monolithic architectures towards SOA is a highly complex process, consisting of several stages [ThWo02, pp. 5-9]. In the first stage existing applications are wrapped with a Web Services Façade (see figure 3). The functionalities of applications can be then accessed through a common communication protocol. This approach lacks central management of services (monitoring, usage statistics, and security). Services such as logging or notifications are implemented separately for each application.

In the next stage common services from the lower layer of SOA are also implemented as Web Services and can be shared by many applications. Service management including issues such as authentication, monitoring or service registries is performed centrally. Business processes, however, still remain in different applications and can be changed by trained developers.

The last stage is characterized by sharing all business-neutral services (e.g. logging) by all applications. Processes and services are managed centrally.

Due to the quick adoption cycles and incompleteness of standards, most organizations moving towards SOA are currently still in the first stage.

4 Mobile Web Services

Mobile devices are available in two variants – as fat or as thin clients. Thin clients are mainly responsible for the presentation whereas the business logic remains on the application server. Fat clients are at least partially in charge of the business logic. Wireless appliances with mobile browsers are examples of thin clients - they can connect to different applications and provide a suitable user interface without knowing about the application logic. Wireless devices can be equipped with different browsers that support various media formats such as WML,

XHTML or HTML [W3C99; W3C03b]. Personal Digital Assistants (PDAs), Palmtops or J2ME-enabled phones are examples of fat clients. They can perform tasks and manage data without the help of a server.

Both types of clients have advantages and drawbacks. In thin clients, user interfaces are bound to the specific markup elements, data caching is not possible, and in periods of disconnectivity the browsers are useless. Generating presentation code for a multitude of devices on the server side is computationally expensive and increases the complexity on the server. In the case of fat clients, user interfaces are similar to those provided for desktop computers, with a functionality that goes beyond the scope of browsers. Program updates, however, have to be performed on the client side and special software has to be installed on the clients.

The most important technology for fat mobile clients is the Java 2 Platform, Micro Edition (J2ME) [cf. Topl02; WhHe02]. It was designed to accommodate a variety of embedded and hand-held devices. It is composed of a configuration and a profile. The *configuration* consists of a virtual machine (VM), core libraries, classes, and APIs. It defines a minimum set of Java VM features and class libraries available on a particular category of devices. The *profile* defines a minimum set of APIs for a particular family of devices. Profiles are implemented on a particular configuration, and applications are written for a particular profile (cf. table 2).

Profile	Foundation profile	Personal profile	RMI profile	PDA profile	MID Profile (MIDP)
Configuration	Connected Device Configuration (CDC)			Connected Limited Device Configuration (CLDC)	
Virtual machine	CVM			KVM	
Device memory	10 MB ◄——► 1 MB			512 Kb ◄——► 160 Kb	

Table 2: The J2ME stack

The Mobile Information Device Profile (MIDP) provides a set of Java APIs specific to a particular category of devices (cell phones, PDAs, etc.). In J2ME the data can be cached on the client with the help of the MIDP Record Management Store (RMS) API and sent to a database when the connection is established.

Mobile Web Services are conventional Web Services with functionality that is used by mobile devices. They can deliver pertinent, timely information to mobile users and provide a reliable infrastructure that works across platforms and architectures. Mobile Web Services can improve integration of employees, partners, or suppliers regardless of locations and access devices. Furthermore, they enable asynchronous connectivity for applications in wireless environments.

Mobile access to an Enterprise Resource Planning system offers various advantages in terms of business agility. For example, when an inventory hits the reorder level, the Web Services Façade of the ERP system can communicate this to the

inventory manager equipped with a mobile device and present options for resolving the event. If the ERP Web Services are integrated with the suppliers' Web Services, the manager could immediately order products, entering order data into the ERP system through his/her mobile phone.

With the help of mobile Web Services it is possible to make complete ERP functionality available for users with different devices and different software. The same task can be performed by a user with a Pocket PC and Microsoft .Net Compact Framework [Micr04], a mobile browser using ASP.Net with mobile controls [Espo03, pp. 507-550], or a J2ME-enabled device with the kSOAP [Enhy04] library. With SOA gaining popularity and enterprises adapting this approach, access to Web Services from mobile devices is becoming a necessity.

5 An Architecture for a Mobile ERP System Based on the SOA Paradigm

5.1 Technological Considerations for Mobile Web Services

Before developing a mobile Web Service, various constraints of wireless devices have to be considered carefully. If a mobile appliance is supposed to be a SOAP client with the capability of using SOAP, it has to be quite powerful in terms of memory and processing speed. Despite the rapid progress in mobile technology, wireless devices still have to cope with severe constraints. Typically, mobile phones are capable of running between 1 and 10 million instructions per second, which is about 1/10 of the speed of a PDA and about 1/100 of the speed of a desktop computer. Furthermore, wireless phones have only between 128 and 512 KB of memory, whereas the memory of a PDA ranges from 2 to 64 MB.

A significant loss of performance has to be expected when parsing SOAP on the mobile client's side. Neither is it feasible to build or parse large XML documents. In addition, the overhead from using SOAP/XML messages during the transport (as compared with the raw data) is estimated to range from 400% to 600%, depending on the amount of information [Tian+04, p. 1098]. If a mobile client is resource-constrained in this sense, it should better communicate with the Web Service through a middle-tier and simply display the results.

In addition to these points, most mobile devices do not offer support for Web Services. J2ME with its MIDP supports the HTTP protocol but not Web Services. Although in the Java Specification Request 172 (JSR172) - J2ME Web Services Specification - [Sun04] Sun addressed the use of XML, SOAP and Web Services on mobile appliances, implementation of standardized support as part of J2ME technology is still in progress. Results are expected around the end of 2004.

kSOAP [Enhy04] is a third-party library from Enhydra that can be used with J2ME for SOAP processing. kSOAP is built on top of kXML which uses less memory for XML processing than traditional XML parsers. kXML is considered suitable for Java applications on mobile devices. kXML-RPC [Endy04a] is also based on kXML, providing support for the XML-RPC protocol for J2ME-enabled devices. kXML-RPC offers J2ME developers more flexibility when selecting a data-exchange mechanism. XML-RPC based communication together with the kSOAP library is in many cases more convenient than document-style Web Services. Compared with traditional libraries for XML processing Enhydra's libraries are really small. While Xerces.jar (a library for processing and transforming XML) has over 1 MB, kSOAP and kXML in a .jar file need only around 42 KB.

For devices equipped with browsers, using Web technology (e.g. Java Server Pages, Servlets, ASP.Net) to deliver the appropriate end-format (WML, XHTML, HTML) seems to be the best choice. Since the content has to be provided in various markup languages depending on browser characteristics, the server should detect device features and dynamically create the appropriate format. Intensive consideration should be given to layout, pagination, and navigation issues.

5.2 A Web Services Façade for an ERP System

A typical ERP system as shown in figure 4 consists of application modules like sales, production, accounting, materials management, etc. and has interfaces to other systems like SCM, CRM, E-commerce applications, and legacy systems. ERP modules communicate with external software packages through remote procedure calls (RPC) [Gros01].

In order to demonstrate the benefits of the SOA concept, we re-implemented parts of a real-world ERP system as a Web Service and provided access from wireless devices. The system used is *infor:COM* [Info04]. Some functions of infor's *Sales* module were re-implemented.

An architecture as shown in figure 5 was developed, based on a three-tier model with a Web Services Façade [Broe03, pp. 189-258]. The application is divided into business objects, business services, and business workflow tiers. Business objects are products, customers, etc. Business services (e.g. a sales service) implement methods to access or manipulate the objects (e.g. retrieving quotes). Business methods are instantiated by business workflows. Such workflows can import business services and business services can import business objects. In J2EE all above mentioned components map to Enterprise JavaBeans (EJB): Business workflows are represented by stateful Session Beans, business services are mapped to stateless Session Beans and business objects to Entity Beans.

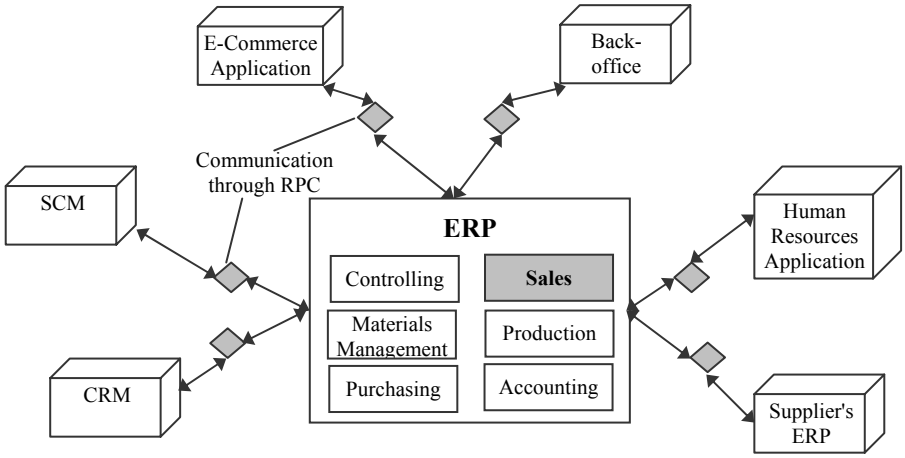

Figure 4: ERP system and possible extensions

Session Beans perform work on behalf of clients. They are generally short-lived and are responsible for quick actions, such as submitting or retrieving an order. A stateful Session Bean retains the state on behalf of a client and can span multiple method requests. A stateless Session Bean does not maintain state across method invocations. Entity Beans represent business data. They are generally long-lived and map to an underlying storage, such as an RDBMS system.

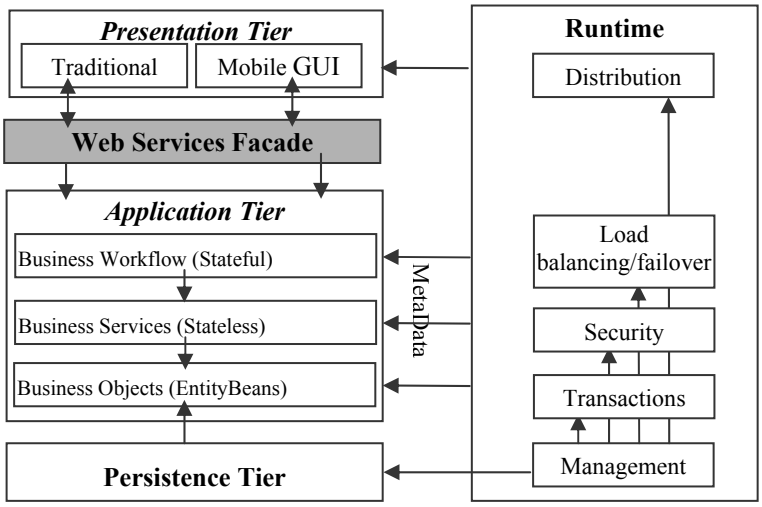

Figure 5: J2EE three-tier model with Web Services Façade

Some examples of Entity Beans include stocks, orders and customers. Typically, Session Beans call Entity Beans to achieve their desired actions [Roma99, pp. 71-204]. In our framework Entity Beans were implemented for different objects such as quotes, sales, stock, invoices, etc. In stateless Session Beans we keep methods for manipulating the objects of Entity Beans, such as retrieving quotes or sales by particular criteria, or inserting orders. In stateful Session Beans we created new methods which consist of the methods previously implemented in stateless Session Beans, necessary to perform a particular task. The beans are exposed as Web Services, using data types that can be supported [see Engl02, pp. 85-107]. This means, for example, that instead of using Java Vectors the methods in Session Beans return arrays of objects from Entity Beans.Web Services were partly generated automatically from EJBs using the JDeveloper 10g IDE [Orac04]. Web Services are exposed both as RPC-style Web Services and as document-style Web Services. Each type of Web Service therefore possesses two WSDL files - one for document style (including the XML schema document) and one for RPC style.

5.3 Mobile Access to Web Services Façade

Our architecture supports thin and fat client applications. It is entirely based on Java technology (Java 2 Platform, Enterprise Edition and Mobile Edition; Java Web Service Developers Pack - JWSDP [Sun04a]). Java was chosen because it satisfies cross-platform requirements; it is an open standard and provides support for Web Services. The ERP system's functionality (Sales module) is exposed in form of a Web Services Façade. The Web Services are implemented as RPC-style services and document-style services. Both types can be accessed through mobile and wireless devices, directly from a mobile device or indirectly, with the help of middleware we developed. The main components of the architecture are described in the following sections. The overall framework is presented in figure 6.

5.3.1 Mobile Web Services Access Engine for RPC-Style and Document-Style Web Services

In our prototypical architecture, mobile access to Web Services is provided through an additional middle layer called *Mobile Web Services Access Engine (MWSAE)*. This engine has the tasks of communicating with the Web Services and delivering device-independent content to the user. It determines the type of browser and the most important device characteristics, and tailors the content to the features of the device.

As a part of the MWSAE, we implemented a Device Context Retriever (DCR) responsible for obtaining the characteristics of the device. The device context is taken from CC/PP profiles (Composite Capabilities/Preference Profiles) [W3C00] or, if CC/PP is not available, from HTTP standard Accept headers [W3C02]. HTTP headers include information about the supported media types (MIME

types), character sets, content encoding, and languages. CC/PP, developed by W3C, is intended for modeling context related to device specifications and user preferences. It is based on the XML serialized Resource Description Framework (RDF) [W3C04a]. A CC/PP profile generally consists of a number of components (e.g. browser) which can have various attributes (e.g. supported media types, screen resolution, etc.). For constructing and processing device profiles in DCR an open-source library, DELI (DElivery Context LIbrary for CC/PP and UAProf), was used [But02].

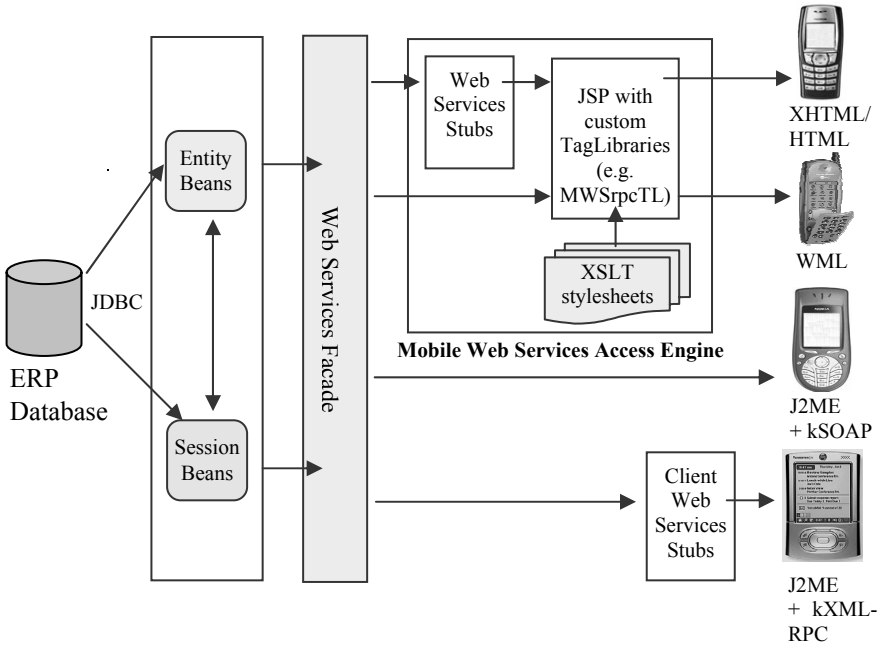

Figure 6: Architecture for mobile ERP system based on SOA concept

DCR was also implemented as a tag library (named Device Context Retriever Tag Library). Tag libraries are reusable modules that can build and access programming language objects and influence the output stream [Sun04b]. They usually encapsulate frequent tasks and can be used across applications, increasing speed and quality of development. Tag libraries have access to all objects available to JavaServer Pages [Good00], they can communicate with each other and can be nested, allowing for complex interactions within a page.

The engine provides support for RPC-style and document-style Web Services. It uses the SOAP with Attachments API for Java (SAAJ) and the Java API for XML-based RPC (JAX-RPC) [Sun04b]. SAAJ allows creating, sending and ob-

taining XML messages over the Internet, whereas JAX-RPC shields developers from the complexity of such a communication.

JAX-RPC is a concept similar to Remote Method Invocation (RMI). Using JAX-RPC, a client can access a Web Service as if the Web Service was a local object mapped into the client's address space. A remote method invocation is performed via the SOAP protocol. A developer can specify the remote procedures by defining remote methods in a Java interface, from which a WSDL file is generated. A stub class (a service proxy masking the marshalling and remote procedure calls) is then generated from a WSDL file. In JWSDP stubs are created by xrpcc [Sun04b].

Instead of sending SOAP to a Web Service, a client can invoke a method on a stub. The JAX-RPC runtime converts this invocation into a SOAP message and transmits it over HTTP. The JAX-RPC runtime on the server side translates the message into a method invocation. This method is invoked on the tie object which delegates it to the service implementation. Figure 7 illustrates the invocation of a Web Service.

Dynamically generated stubs exist for all Web Services. JavaServer Pages contained in the *Mobile Web Services Access Engine* then create an instance of the stub class and invoke the business methods through this proxy by specifying methods, parameters and the service endpoint. This technique was used for retrieving information that is limited in scope, such as quotes numbers.

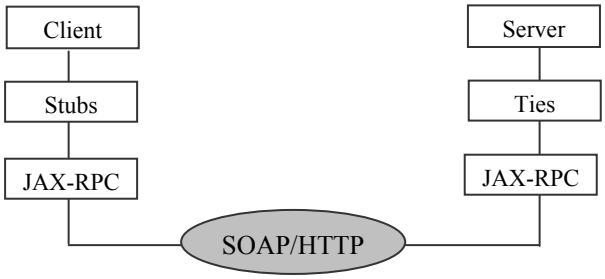

Figure 7: XML-RPC with JAX-RPC

In addition we developed a *Mobile Web Services RPC Tag Library (MWSrpcTL)* with the previously described functionality. The *MWSrpcTL* tag library allows binding to a Web Service as well as sending and receiving SOAP messages by invoking methods with parameters on the Web Service and receiving output. It also offers the possibility to map SOAP/XML to Java types. Instead of mixing Java code with JSP pages, the developer can simply put appropriate tags for interacting with the desired Web Service.

In order to deliver the information retrieved from a Web Service in the correct format, a Content Adaptation Tag Library (CATL) was used (for more details see [JaDa03]). This library converts certain elements, such as tables, forms, images,

etc., to device-specific elements. It also takes care of pagination and produces appropriate output based on information obtained from the DCRTL. However, it is still possible to apply a different technique (e.g. XSLT) for transforming the content to the accurate end format.

SAAJ is used for document-style Web Services. In SAAJ all information is transported as SOAP messages over a connection. This type of connection is called a *point-to-point* connection (from one endpoint to another endpoint) and the messages are called request-response messages. Endpoints are represented by a `java.net.URL` objects. Messages are sent over a `SOAPConnection` object with the method `call`, which sends a request and waits until it receives a response (cf. [Sun04b] for more details).

The Mobile Web Services Access Engine provides two classes for document-style Web Services. The first class is responsible for creating SOAP messages (with and without attachments); the second one has the challenging task of sending those messages and obtaining the results. Received SOAP messages are transformed into the appropriate format by transformation objects with XSLT stylesheets [W3C03c] according to device characteristics delivered by the Device Context Retriever component.

SOAP documents are parsed and modified according to the respective stylesheet. In our approach the Java Transformation API for XML (TrAX) [Apac04a] was chosen to invoke XSL stylesheets. TrAX is capable of compiling stylesheets and holding them in memory, thus improving the performance significantly. Usually TrAX takes an XML (SOAP) file and a stylesheet as input to create an output representing the result of the transformation. For better performance our stylesheets are compiled when they are used for the first time. Then they can be re-used in multiple transformations. For each output type a number of reusable template stylesheets was prepared.

5.3.2 Direct Access to Web Services from Mobile Devices

The second way of accessing Web Services is to invoke them directly from a sufficiently powerful mobile device. For appliances supporting J2ME technology, we developed a set of reusable classes that facilitate the consumption of Web Services. Those classes can be instantiated in an MIDP application (so-called midlet), and particular methods can be invoked.

For document-style Web Services our implementation is based on kSOAP. The class we developed is responsible for generating SOAP messages, sending them, receiving input and processing the SOAP messages obtained. Simple midlets can be generated automatically based on the functionality provided by the Web Service. They may subsequently be modified by a developer as needed.

For RPC-style Web Services the kXML-RPC library was used. Appropriate stubs are provided on the client side for all services. The stubs can be re-generated automatically if the description of a Web Service in a WSDL files changes.

Features of devices or services	Appropriate Protocol
Small memory, reduced bandwidth	XML-RPC
Speed, efficiency, easy maintenance	XML-RPC
Interoperability with other parties beyond control	SOAP
Exchange of robust data structure, flexible messaging architecture	SOAP

Table 3: Characteristics of devices/services vs. Web Services protocols [cf. GaGo02, p.46]

Midlets invoke particular methods from the stub and the communication process is similar to the one presented in figure 7. Instead of JAX-RPC, kXML-RPC is applied now. Stubs and midlets can be generated automatically from the WSDL description and changed by developers according to specific clients' needs. The generation is based on the Axis WSDL2Java tool [Apac04] with some enhancements regarding the generation of stubs for mobile clients.

Since our architecture supports both types of services, table 3 provides some guidelines as to the benefits of RPC-style and document-style Web Services, when they are invoked directly from mobile appliances.

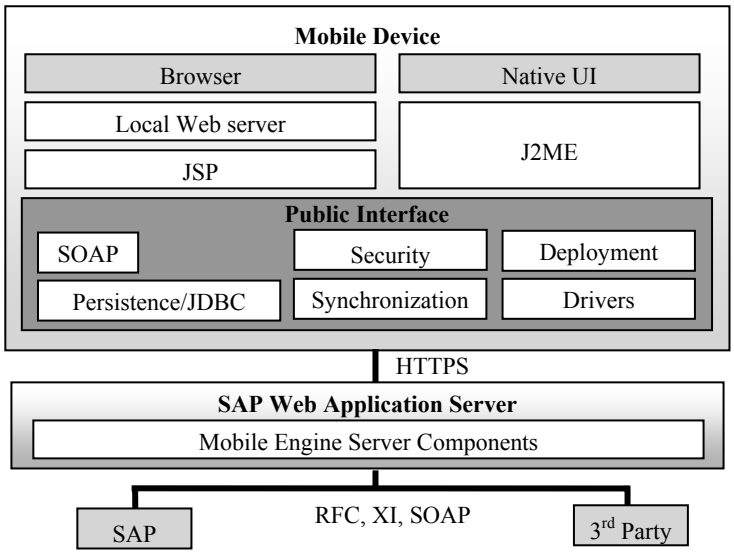

Figure 8: Software architecture of the SAP Mobile Infrastructure [cf. SAP04a]

5.4 Related Work

SAP, the largest provider of Enterprise Resource Planning systems, developed a solution which offers mobile access to its ERP application based on Web Services. The so-called Mobile Infrastructure (MI) is embedded in the SAP NetWeaver application and integration platform and includes SAP Mobile Infrastructure Client (Mobile Engine) and SAP Mobile Infrastructure Server (cf. [SAP04a; SAP04]. The SAP MI client (cf. figure 8) consists of its own Web server, database layer and business logic. Developers can build applications with JavaServer Pages for standard browsers or J2ME for PDAs, laptops and smart phones. Since the SAP MI client has to be installed locally on a mobile device, it can be applied only on powerful devices, preferably PDAs, supporting at least the Java Runtime Environment (JRE) 1.1.8. It cannot be used on Palmtops without such support nor on resource-scarce wireless devices. The SAP Mobile Infrastructure offers access to Web Services directly from the mobile client. However, it does not provide the possibility to communicate with Web Services through middleware for wireless devices which are not equipped with Java. Our approach covers both cases.

6 Conclusions and Further Work

Analysts and industry experts forecast that the market for wireless applications will continuously grow over the next few years. With increasing user needs to access essential business information from anywhere at anytime, organizations will have to find effective architectural solutions that allow users to communicate with the organization's information systems. Technically speaking, different types of clients have to be supported.

SOA is a promising approach for enterprises with a complex architecture encompassing multiple systems residing on multiple platforms. SOA can help to solve the EAI problem. Therefore vendors of software packages such as ERP are redesigning their systems. According to Gartner Group, SOA-based systems will be the prevailing software-engineering practice by 2008 [Nati03]. As mobile computing and SOA are growing together, a special focus of research and development has to be set on accessing Web Services from mobile devices. Web Services seem to be a natural solution for integration problems and distributed environments.

However, mobile devices still lack support for core Web Services technologies. This paper proposed four approaches for the communication between Web Services Façades on top of an ERP system and a variety of mobile devices supporting J2ME or equipped with wireless browsers. For the latter type of devices, a supplementary middle layer was introduced. J2ME-enabled devices with additional lightweight libraries are capable of communicating directly with a Web Service. Yet this is rather a solution for the future since kSOAP and kXML-RPC are not

standard libraries integrated in J2ME. It may be expected that this situation will change with further standardization and the advent of a new version of J2ME technology, providing support for SOAP and Web Services.

Our architecture currently allows content delivery in some basic formats such as HTML, XHTML, or WML. Future developments will concentrate on adding support for new markup languages like VoiceXML [W3C03a]. We also plan to explore issues of interoperability of existing Web Services and mobile clients built with the Microsoft .NET framework or Windows CE technology.

A short-term goal is to develop a dedicated Integrated Development Environment (IDE) that will facilitate the creation of mobile Web Services for further ERP modules. Some IDEs for Web Services, even for mobile Web Services (JDeveloper10g, Eclipse 2.1. with appropriate plug-ins, etc.), already exist. However, the support they offer is very limited for a task like mobilizing an ERP system (e.g. lack of support for document-style Web Services, support only for own servers with their deployment peculiarities, etc.). The IDE is supposed to substantially automate the creation of different components by providing suitable wizards and by re-using modules which were developed before.

References

[Apac04] Apache Software Foundation: Axis 1.2. http://ws.apache.org/axis/, 2004. Viewed 04-07-01.

[Apac04a] Apache Software Foundation: Java Transformation API for XML (TrAX). http://xml.apache.org/xalan-j/trax.html, 2004. Viewed 04-07-01.

[Asto03] Astor, A.: Patterns in Web Services Projects, in: Web Services Journal, 5/2003, pp. 42-44.

[Broe03] Broemmer, D.: J2EE Best Practices: Java Design Patterns, Automation, and Performance, Wiley Publishing, Inc., Indianapolis, 2003.

[Butl02] Butler, M.H.: DELI: A Delivery Context Library for CC/PP and UAProf, External Technical Report HPL-2001-260. http://www-uk.hpl.hp.com/people/marbut/DeliUserGuideWEB.htm, 2002. Viewed 04-06-23.

[CaKo04] Caldwell, D.; Koch, J.: Mobile Computing and its Impact on the Changing Nature of Work and Organizations. http://sts.scu.edu/research/MobileComputing.pdf, 2004. Viewed 04-06-27.

[ChaJe02] Chappell, D.; Jewell, T.: Java Web Services, O'Reilly, Sebastopol, 2002.

[ChaJo03] Charlesworth, I.; Jones, T.: The EAI and Web Services Report, in: *e*AI Journal, 3/2003, pp. 12-18.

[Engl02] Englander, R.: Java and SOAP, O'Reilly, Sebastopol, 2002.

[Enhy04] Enhydra: kSOAP 2.0. http://ksoap.objectweb.org/, 2004. Viewed 04-06-23.

[Enhy04a] Enhydra: kXML-RPC. http://kxmlrpc.objectweb.org/, 2004. Viewed 04-06-23.

[Espo03] Esposito, D.: Programming Microsoft ASP.NET, MS Press, Redmond, 2003.

[GaGo02] Gabhart, K., Gordon, J.: Wireless Web Services with J2ME, Web Services Journal, Vol. 2/2, 2002, pp. 44-48.

[Good00] Goodwill, J.: Pure Java Server Pages, Sams Publishing, Indianapolis, 2000.

[Gros01] Grosso, W.: Java RMI, O'Reilly & Associates, Sebastopol, 2001.

[IBM04] IBM UDDI Business Registry. https://uddi.ibm.com/ubr/registry.html. Viewed 04-06-29.

[Info04] Infor:COM. http://www.infor.de. Viewed 04-06-24.

[JaDa03] Jankowska, A.M.; Dabkowski, A.: Content Adaptation Tag Library - An Approach for User Interface Adaptation for Different Devices, in: Proceedings of the Net.ObjectDays 2003 Conference, Erfurt, Germany, pp. 78-92.

[KeRo03] Kezmah, B.; Rozman, I.: Web Services in ERP Solutions: A Managerial Perspective., in: Proceedings of the International Symposium Metainformatics 2002, Esbjerg, Denmark, 2003, pp. 177-179.

[Lint99] Linthicum, D.: Enterprise Application Integration, Addison Wesley, Boston, 1999.

[McGo+03] McGovern, J.; Tyagi, S.; Stevens, M.; Matthew, S.: Java Web Services Architecture, Morgan Kaufmann Publishers, San Francisco, 2003.

[Micr04] Microsoft: .NET Compact Framework. http://msdn.microsoft.com/mobility/prodtechinfo/devtools/netcf/, 2004. Viewed 04-06-25.

[Micr04a] Microsoft UDDI Business Registry. http://uddi.microsoft.com/default.aspx. Viewed 04-06-29.

[Nati03] Natis, Y.: Service-Oriented Architecture Scenario, Gartner Research. http://www4.gartner.com/resources/114300/114358/114358.pdf, 2003. Viewed 04-06-28.

[OASI03] OASIS: UDDI Version 3. http://www.uddi.org/specification.html, 2003. Viewed 04-06-29.

[OMA04] Open Mobile Alliance Mobile Web Services Working Group. http://www.openmobilealliance.org/tech/wg_committees/mws.html. Viewed 04-06-29.

[Orac04] Oracle Corp.: JDeveloper 10g, http://otn.oracle.com/products/jdev/index.html. Viewed 04-06-20.

[Puli03] Pulier, E.: The Reality, Challenges, and Enormous Potential of Web Services, in: Web Services Journal, 5/2003, pp. 48-50.

[Redm97] Redmond, F.: DCOM: Microsoft Distributed Component Object Model, John Wiley, New York, 1997.

[Roma99] Roman, E.: Mastering Enterprise JavaBeans, John Wiley, New York, 1999.

[SAP04] SAP AG: http://www.sap.com/solutions/mobilebusiness. Viewed 04-06-24.

[SAP04a] SAP: SAP Mobile Engine: An Open Platform for Enterprise Mobility. http://www.sap.co.kr/solutions/mobilebusiness/pdf/50057072s.pdf. Viewed 04-09-28.

[Smol03] Smolnicki, J.: How XML and Web Services Will Change Your Business, CIO, 11/2003, http://www.pwcglobal.com/Extweb/ncinthenews.nsf/docid/2C9CB295270D752DCA256DD5006D122D. Viewed 04-06-24.

[Sun04] Sun Microsystems: Java Specification Request 172: J2ME Web Services Specification, http://jcp.org/en/jsr/detail?id=172, 2004. Viewed 04-06-21.

[Sun04a] Sun Microsystems: Java Web Services Developer Pack, http://java.sun.com/webservices/jwsdp/index.jsp, 2004. Viewed 04-06-23.

[Sun04b] Sun Microsystems: The Java Web Services Tutorial, 2004, http://java.sun.com/webservices/docs/1.3/tutorial/doc/index.html, 2004. Viewed 04-06-20.

[ThWo02] ThoughtWorks: Web Services: Pathway to a Service-Oriented Architecture. http://www.thoughtworks.com/us/library/SOA.pdf, 2002. Viewed 04-06-29.

[Tian+04] Tian, M. et al.: Performance Considerations for Mobile Web Services, in: Computer Communications Journal, Volume 27, Issue 11, 2004, pp. 1097-1105.

[Topl02] Topley, K.: J2ME in a Nutshell, O'Reilly & Associates, Sebastopol, 2002.

[W3C00] W3C: Composite Capabilities/Preference Profiles: Terminology and Abbreviations, Working Draft. http://www.w3.org/TR/2000/WD-CCPP-ta-20000721/, 2000. Viewed 04-06-23.

[W3C02] W3C: Delivery Context Overview for Device Independence. http://www.w3c.org/2001/di/public/dco, 2002. Viewed 04-06-17.

[W3C03] W3C: SOAP 1.2. http://www.w3.org/TR/SOAP/, 2003. Viewed 04-06-29.

[W3C03a] W3C: Voice Extensible Markup Language (VoiceXML) Version 2.0. http://www.w3.org/TR/voicexml20/, 2003. Viewed 04-06-20.

[W3C03b] W3C: XHTML 2.0 The Extensible HyperText Markup Language Specification. http://www.w3.org/TR/xhtml2/, 2003, Viewed 04-06-25.

[W3C03c] W3C: XSL Transformations (XSLT) Version 2.0. http://www.w3.org/TR/xslt20, 2003. Viewed 04-06-23.

[W3C04a] W3C: RDF Primer. http://www.w3.org/TR/rdf-primer, 2004. Viewed 04-06-27.

[W3C04b] W3C: Web Services Architecture Requirements. http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211/, 2004. Viewed 04-06-29.

[W3C04c] W3C: WSDL 2.0. http://www.w3.org/TR/wsdl20, 2004. Viewed 04-06-24.

[W3C99] W3C: HTML 4.01 Specification. http://www.w3.org/TR/html4/, 1999. Viewed 04-06-26.

[Wass03] Wassink, J.: Das Ende des Auftragsblocks, iCONOMY, 5-6/2003, pp. 46-47.

[WhHe02] White, J.; Hemphill, D.: Java 2 Micro Edition, Manning Publ., Greenwich, 2002.