PRACE NAUKOWE AKADEMII EKONOMICZNEJ WE WROCŁAWIU
Nr 1081 — 2005
Nowoczesne technologie informacyjne w zarządzaniu

Anna Maria Jankowska, Bożena Jankowska

EVOLUTION OF ARCHITECTURAL SOLUTIONS FOR ENTERPRISE RESOURCE PLANNING SYSTEMS

1. Introduction

In the last three decades, software architectures underwent a radical shift from relatively simple, monolithical systems to compound structures, consisting preferably of reusable software components. This development is also reflected in the architectural solutions for Enterprise Resource Planning (ERP) systems that are the IS backbone of many organizations. Important trends and technologies such as Web Services, mobile and ubiquitous computing, sophisticated design patterns or the popularity of Internet had a remarkable impact on the current ERP systems. One of the most significant issues which is nowadays driving the design of new ERP architectures is Enterprise Application Integration (EAI). 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.

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 [Pulier 2003, p. 48]. Industry standards such as the Common Object Request Broker Architecture (CORBA) [Grosso 2001, pp. 449-469], Microsoft Distributed Component Object Model [Redmond 1997], and XML [W3C, Internet 2005h] provide interfaces on a higher level. Nevertheless, change management in EAI projects is a cumbersome process.

Web Services, which are becoming a basis of new architectures, are a promising approach for 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 [Charlesworth, Jones 2003, p.14].

This paper describes the most important steps in the evolution of architectural approaches for ERP systems. It is organized as follows: In the next section, the concept of software architecture, multi-tier systems and mobile ERP systems is introduced. Component-based software, Service-Oriented Architectures and Web Services are outlined in section three. Section four demonstrates on the example of Infor: COM system an architectural solutions for an ERP system based on the SOA concept. In the final section some conclusions are drawn.

2. From Modules to Multi-Tier Architectures

In the 70s, the main discussion's topics were software structures, their optimization and the decomposition of systems into modules [Parnas 1972]. As the personal computers (PCs) became more powerful and powerful terminal computers were not longer needed, the term software architecture became a new topic of debates. Although there is no common agreement on the definition of software architecture, most authors describe it as a kind of a plan or a blueprint which specifies the types of software elements, their organization and relationships to each other as well as the principles governing its design and evolution [Dickman 2002].

Current application systems have usually three major layers: presentation layer, application's logic layer, and services layer [Britton 2000, pp. 91-106]. The presentation layer provides the user interface and is responsible for the interaction between the user and the device. The application or business logic layer contains the business rules that drive the given enterprise. The services layer provides general services needed by the other layers such as database services, file services or communication services. The functionalities of these three layers can be assigned to logical entities called tiers.

During the last twenty years system architectures have evolved from client/server to distributed architectures. In the client/server model (called also two-tier architecture) processing management was split between the client and the

server while the presentation layer was located on the client and the server was responsible for database management.

In three tier architectures an additional tier (called a middle tier) was introduced between the client interface and the database management server. The middle tier should perform queuing, application execution, database staging and remove the application logic from the client. In this way systems should be more scalable, support and installation costs decreased and performance could be improved.

The term multi-tier architecture usually refers to the fact that more servers than just one application server are included. An immediate example is a Web server as part of an application that is accessed over the Internet. Generally speaking an application server can request services from many other servers, i.e. the services themselves use other services to respond properly to the client's original request. Three-tier and multi-tier architectures are a typical solution for many ERP systems (e.g. the open-source ERP system Compiere[1] or Infor).

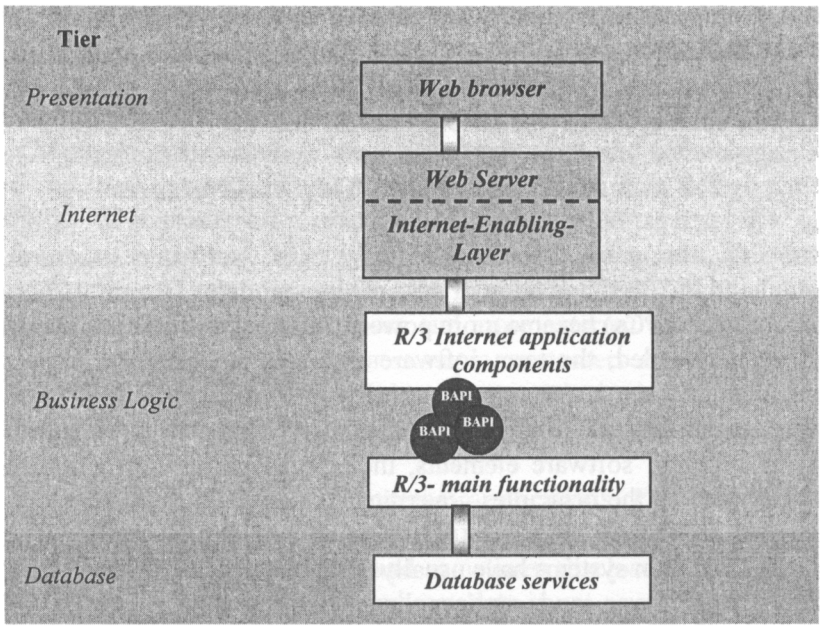

Fig. 1. SAP R/3multi-tier architecture

Source: [SAP 1997].

In 1997 SAP introduced a multi-tier architecture in its ERP system [SAP 1997]. In addition to the three classical tiers (data, business logic and presentation) a layer for Internet access (called Internet layer or Internet-enabling layer) was

[1] Cf. http://www.compiere.org.

provided. Figure 1 presents the architecture. The business logic tier was split in conventional R/3 components and R/3-Internet application components.

Mobile computing has improved the quality of doing business today [Caldwell, Koch 2004]. 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 multi-tier ERP systems with mobile front-ends in order to meet new requirements [SAP 2005; Infor 2005]. Mobile interfaces have resulted in significant savings and improvements in customer service, not only in large companies but also in small ones [Wassink 2003]. Some firms were able to almost eliminate duplicate acquisition of data by sales representatives and office employees, and to decrease the error ratio significantly.

3. Web Services and Service-Oriented Architectures

In 1993, the distributed/collaborative enterprise architecture emerged. It was based on shared, reusable business models (not just objects) on an enterprise-wide scale. Web Services based on standardized technologies and creating applications from distributed software components can be seen as a descendent of this architecture.

Web Services. 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" [W3C 2004b]. 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 entire Web Services infrastructure is based on XML standards: Simple Object Access Protocol (SOAP) [W3C, Internet 2005i], Web Services Description Language (WSDL) [W3C Internet 2005k], and Universal Description, Discovery, and Integration (UDDI) [OASIS 2003].

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.

Web Services can be implemented as either an RPC-style Web Service or a document-style Web Service [Chappell, Jewell 2002, 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.

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 [Microsoft 2004, IBM 2004] 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.

Service-Oriented Architecture (SOA). 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. [McGovern et al 2004, 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.

The service consumer is some kind of software module (e.g. application, another service) that works according to the "find, bind and execute" paradigm. 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. Additionally, 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 [McGovern et al 2003, 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. WS Fa ade

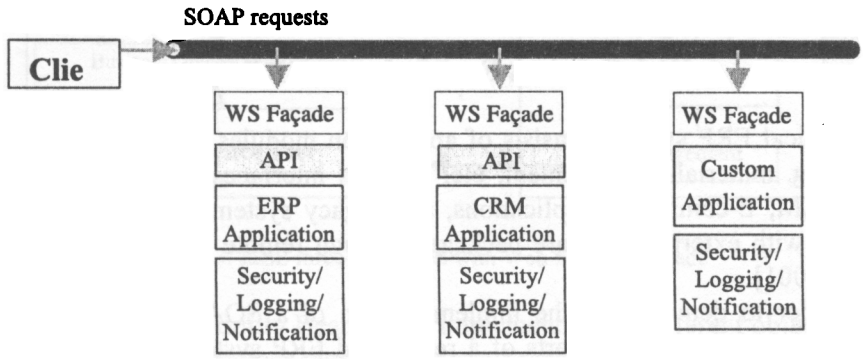

Fig. 2. Web Services Façades

The transition from traditional, monolithic architectures towards SOA is a highly complex process, consisting of several stages [ThoughtWorks 2002, pp. 5-9]. In the first stage existing applications are wrapped with a Web Services Façade (see Figure 2). 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.

The driving force for redesigning information systems according to the SOA paradigm is Enterprise Application Integration (EAI). EAI is defined as the unrestricted sharing of data and business processes among any connected

applications or data sources in the enterprise [Linthicum 1999, p. 3]. Based on Web Services data from various systems can be consolidated into a single view. Aggregating application functionalities within a business service means that other applications can use that service without understanding its internal complexity.

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.

4. Web Services Façade for an ERP System

A typical ERP system 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) [Grosso 2001].

In order to demonstrate the implementation of a SOA concept for an ERP system, we re-implemented parts of a real-world ERP system as a Web Services Façade. The system used is infor:COM [Infor 2005].

An architecture as presented in Figure 3 was developed. It is based on a three -tier model with a Web Services Façade [Broemmer 2003, 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 of the 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 correspond to Entity Beans.

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. Some examples of Entity Beans include stocks, orders and customers. Typically, Session Beans call Entity Beans to achieve their desired actions [Roman 1999, pp. 71-204].

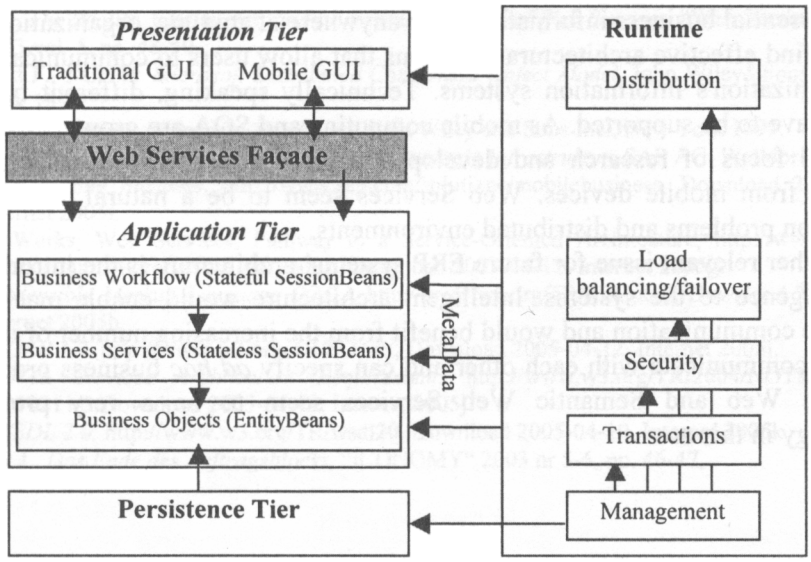

Fig. 3. J2EE three-tier model with Web Services Façade

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 [Englander 2002, 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 [Oracle 2005]. 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. Conclusions and Future Trends

SOA is a promising approach for enterprises with a complex architecture encompassing multiple systems residing on multiple platforms. It can help to solve the common EAI problems. Therefore vendors of software packages such as ERP are redesigning their systems according to this paradigm.

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. 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.

Another relevant issue for future ERP systems' architecture is the introduction of intelligence to the systems. Intelligent architecture would enable machine-to -machine communication and would benefit from the increasing number of devices that can communicate with each other and can specify *ad hoc* business processes. Semantic Web and Semantic Web Services seem to be a very promising technology in this area.

References

Bass L. et al., *Software Architecture in Practice*, Addison-Wesley, London 2003.

Britton Ch., *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems*, Addison-Wesley, Boston 2000.

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

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, Download 2005-04-17, Internet 2005.

Chappell D., Jewell T., *Java Web Services*, O'Reilly & Associates, Sebastopol 2002.

Charlesworth I., Jones T., *The EAI and Web Services Report*, "eAI Journal" 2003 nr 3, pp. 12-18.

Dickman A., *Two-Tier Versus Three-Tier Applications*, "Informationweek" 2002 nr 553, pp. 74-80.

Englander R., *Java and SOAP*, O'Reilly & Associates, Sebastopol 2002.

Grosso W., *Java RMI*, O'Reilly & Associates, Sebastopol 2001.

IBM, *UDDI Business Registry*, https://uddi.ibm.com/ubr/registry.html., Download 2005-04-15, Internet 2005a.

Infor:COM. http://www.infor.de. Download 2005-04-14, Internet 2005b.

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

Linthicum D., *Enterprise Application Integration*, Addison Wesley, Boston 1999.

McGovern J. et al., *Java Web Services Architecture*, Morgan Kaufmann Publishers, San Francisco, 2003.

Microsoft, *UDDI Business Registry*, http://uddi.microsoft.com/default.aspx, Download 2005-04-14, Internet 2005c.

OASIS, *UDDI Version 3*, http://www.uddi.org/specification.html, 2003, Download 2005-04-12, Internet 2005d.

Oracle Corp., *JDeveloper 10g*, http://otn.oracle.com/products/jdev/index.html, Download 2005-04-02, Internet 2005e.

Parnas D., *On the Criteria to be Used in Decomposing Systems into Modules*, "Communications of the ACM", 1972, vol. 15 nr 12, pp. 1053-1058.

Pulier E., *The Reality, Challenges, and Enormous Potential of Web Services*, "Web Services Journal" 2003 vol. 5, pp. 48-50.
Redmond F., *DCOM: Microsoft Distributed Component Object Model*, John Wiley&Sons Inc., New York 1997.
Roman E., *Mastering Enterprise JavaBeans*, John Wiley and Sons Inc., New York 1999.
SAP AG, *Funktionen im Detail, System R/3, Technologie Infrastruktur*, SAP AG, Walldorf 1997.
SAP AG: *Mobile Business*, http://www.sap.com/solutions/mobilebusiness, Download 2005-04-12, Internet 2005f.
ThoughtWorks, Web Services, Pathway to a Service-Oriented Architecture, http://www.thoughtworks.com/us/library/SOA.pdf, 2002, Download 2005-04-12, Internet 2005g.
W3C, *Extensible Markup Language (XML)*, http://www.w3.org/TR/REC-xml, Download 2005-04-13, Internet 2005h.
W3C, *SOAP 1.2*. http://www.w3.org/TR/SOAP/, Download 2005-04-12, Internet 2005i.
W3C, *Web Services Architecture Requirement*, http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211/, Download 2005-04-12, Internet 2005j.
W3C, *WSDL 2.0*. http://www.w3.org/TR/wsdl20, Download 2005-04-10, Internet 2005k.
Wassink J., *Das Ende des Auftragsblocks*, "iCONOMY" 2003 nr 5-6, pp. 46-47.

EWOLUCJA ARCHITEKTUR SYSTEMÓW KLASY ERP

Streszczenie

Rozwój architektur softwarowych z „monolitów" do systemów bazujących na komponentach, jak rownież nowe technologie i trendy, takie jak serwisy sieciowe, urządzenia mobilne, koncepcja architektury zorientowanej na serwisy czy wreszcie ogromna popularność Internetu nie pozostały bez wpływu na architektury systemów klasy ERP. Niniejsza publikacja pokazuje zmiany, jakim podlegały te systemy na skutek pojawiania się nowych koncepcji w zakresie architektury, zwiekszającej się kompleksowości programów, zmieniających się warunków rynkowych oraz rosnących możliwości sprzętu.

Dipl.-Kffr., Dipl. ESC Anna Maria Jankowska is a teaching and research assistant at the Chair of Business Informatics at the European University Viadrina, Frankfurt/Oder
e-mail: jankowska@uni-ffo.de

Dipl-Kffr. Bożena Jankowska is a research assistant at the Chair of Business Informatics, Banking and Finance at the European University Viadrina, Frankfurt/Oder
e-mail: euv-6204@uni-ffo.de