December 2006

A Web Services Façade for an Open Source ERP System

Karl Kurbel
European University Viadrina

Denny Schreber
European University Viadrina

Brandon Ulrich
B2 International- Ltd.

A Web Services Façade for an Open Source ERP System

Karl Kurbel
European University Viadrina, POB 1786,
15207 Frankfurt (Oder), Germany
wi-sek@uni-ffo.de

Denny Schreber
European University Viadrina, POB 1786,
15207 Frankfurt (Oder), Germany
schreber@uni-ffo.de

Brandon Ulrich
B2 International, Ltd., Tompa u. 21. 6.em 4.
1094 Budapest, Hungary
bulrich@b2international.com

ABSTRACT

Web services are becoming a popular means of realizing service-oriented architectures. In enterprise resource planning (ERP), web services can enhance the flexibility of ERP systems by making them accessible from other programs over the Internet. This paper presents an approach on providing a web service interface to an open source ERP system (Compiere). After a short introduction, the implementation of a web services façade for Compiere is discussed. To facilitate access to Compiere's business logic and pertinent data from external systems, this implementation provides interfaces in the form of web services to enable the visibility in supply networks that is needed by supply chain management (SCM) applications. The interface exposes a simplified façade that reduces Compiere's complexity and decreases network traffic by aggregating numerous fine-grained method calls. Potentials and problems of this approach are discussed in the paper.

Keywords

web service façade, open source enterprise systems, enterprise resource planning, Compiere

INTRODUCTION AND BACKGROUND OF THE WORK

The work reported in this paper is embedded in a project in the field of supply chain event management (SCEM). In this project, intelligent agents at the partners' sites are supposed to monitor the supply chains and take actions if unforeseen events occur. To be able to do so, agents need information from other partners which is typically available in their ERP systems. Like other software around an ERP system, software agents need an interface through which they can access ERP functionality, provided that the partner allows such access. Web services have emerged as a potential solution to this problem and have been quickly embraced by both researchers and practitioners alike. Since web services are also a flexible and effective means to create interfaces for existing ERP systems, we developed a web service façade for the Compiere ERP system.

Another phenomenon besides web services has emerged in the enterprise market, sometimes even called a disruptive technology: open source software. While the initial hype suggesting that open source software would supplant proprietary software has faded, open source software has continued to gain ground in business environments. Yet enterprises have to adapt to the characteristics of open source software. Some opportunities and drawbacks experienced in the work with the Compiere open source ERP system are reflected in this paper.

The remainder of this paper is organized as follows: In the first section, we present a brief overview of Compiere and reasons that it is an alternative to proprietary solutions in professional settings. Thereafter, a description of the problem and a short introduction to web services, the chosen solution, is given. In this context, design questions of a web service façade for Compiere are discussed. The following section presents the implementation details for this façade. The results and implications of that approach as well as some related work supplement the previous sections. A conclusion is given in the final section and potential for future work is indicated.

REASONS FOR CONSIDERING COMPIERE

In addition to the well-known and stable open source applications like the Apache web server and the GNU/Linux operating system, enterprise resource planning (ERP) systems like Compiere and accounting systems like SQL-Ledger are finding their way into business environments (Greenemeier and Kontzer, 2005; cf. MySQL, 2004). Open Source Software itself can be distinguished from other types of software (traditional or proprietary software, public domain, freeware, shareware) by the license, the development model, and the possible business models. The most significant distinction is that the chosen license must comply with the Open Source Definition, which requires source code access, redistribution rights, and additional terms to determine if a product qualifies as being truly open source (Perens, 2006). Furthermore, a new development model was identified, the so-called bazaar-style development model (Raymond, 1999). This model contrasts the traditional model of software development to that of open source software development by means of an analogy between a cathedral and a bazaar. The cathedral represents the traditional development model, which is compared to the construction of a cathedral where only a limited number of professionals work. In contrast, open source software development is compared to trading at a bazaar, where all sorts of people could contribute to the developed software (Raymond, 1999). However, not all open source software is necessarily developed using this model. Due to licensing conditions, it is difficult to find a sustainable business model. At the moment, providing services centered around open source software seems the most common model.

Because of the licensing conditions, open source software offer some benefits when compared to proprietary solutions. In fact, costs seem to be the most influential factor. Nevertheless, modification and free redistribution of the source code are also important, e.g. during the required customization when introducing an ERP system. Product support is normally provided by members of the project's community, rather than by a professional organization. Especially in research the access to source code simplifies the learning process since questions can be answered by directly referencing the source code. These factors influenced the decision for choosing the Compiere ERP system. It may be argued that it is unrealistic to use an ERP system like Compiere, which lacks many of the features found in commercial ERP systems from SAP or Oracle. The point is that this product is already used in the business world and the functionality is satisfactory for many small to medium-sized businesses. Although Compiere itself is open source, using it does not provide a solely open source scenario as illustrated in (Dreiling, Klaus, Rosemann and Wyssusek, 2005). Although it depends on the proprietary Java programming language and the Oracle Database, it runs on open source operating systems such as the Linux operating system. Compiere itself is released under the Mozilla Public License and stems from a project sponsored by Goodyear developed for the tire retail market (Compiere Pilot, 2006). Later it became a full ERP/CRM (customer relationship management) system by embracing the Bazaar-style development model while hosting the project on SourceForge.net (http://sourceforge.net/projects/compiere). The business model behind Compiere is simple and typical for open source software: Compiere/ComPiere, Inc. generates revenue by selling support contracts, helping its customers migrate to new versions, and by licensing partners to provide consulting services.

WEB SERVICE FAÇADE FOR COMPIERE

Choosing the Compiere ERP system to represent partners of a supply network presented the following problem: How could business logic and data be accessed? In the case of Compiere, interfaces that allow asynchronous and automated access were not available but a solution was necessary. Furthermore, applications accessing the ERP system should not be concerned with Compiere's implementation details. The advantage in using an open source ERP system lies in the available source code and therefore access to the application programming interfaces (API). Proprietary ERP systems in general do not offer access to APIs. Another advantage is that open source offers the possibility to add potentially necessary functionality through additional code without impeding existing functionality. Web services, as a state-of-the-art, flexible, and effective method, were chosen to approach this problem.

In information systems (IS) literature the discussion about web services is divided into technical and business-oriented perspectives (Alt, Gizanis and Legner, 2005). The technical perspective discusses service descriptions, enterprise application integration (EAI) capabilities, and standards. On the other hand, more business-oriented research is conducted where web services could be used for outsourcing process elements to external service providers. The following sections are written in a more technical perspective but should also be seen as an end to facilitate the collaboration of business partners in supply networks.

Web Services rely on several open standards including SOAP (formerly Simple Object Access Protocol, but the acronym was dropped in the latest SOAP specification), Web Services Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI). Web services provide one of the most popular means to realize a service-oriented architecture (SOA). A SOA can be defined as a kind of architecture that facilitates loosely coupled distributed systems that allow clients to use application functionality via provided services interfaces (He, 2003). SOAP RPC Web Services do not

really fit in a SOA because they are very difficult to prescribe in distributed environments, so our work can only be seen as a first step towards realizing a SOA (He, 2003). Nevertheless, we have at least a service provider, a service consumer, and a service registry. The web service façade (service provider) described in this paper is based on the façade pattern, a common object-oriented design pattern used to simplify interfaces to complex systems, reduce dependencies between clients and subsystems, and create layers of subsystems (Gamma, Helm, Johnson, and Vlissides, 1995).

Providing a simple interface allows clients to utilize a system's functionality without understanding the underlying implementation details. This gives the clients the freedom to concentrate on building higher-value extensions to the system without concentrating on the possibly numerous components and subsystems necessary to perform lower-level tasks. Using the façade pattern also provides a means of decoupling the client implementation from the subsystem implementation (Freeman and Freeman, 2004). This means that when the underlying systems undergo change, only the façade's implementation that interacts with this system must be updated. The external part of the façade, i.e. that visible to clients, remains constant. Clients normally do not recognize implementation changes behind the façade. This is important when working with systems undergoing frequent change, such as Compiere, which often requires numerous database structural changes even between point versions, e.g. ComPiere, Inc. estimates that moving from Compiere 2.5.0 to 2.5.1 involved "about 1200 database structure and 20,000 data changes" (Compiere Changes, 2006). Of course, if functionality is added or significant changes were made clients have to adapt to the changes.

An additional advantage of decoupling is that it allows the subsystems to be exchanged without making changes to the client code. In open source enterprise systems, many projects have a particular expertise on which they have concentrated development. For example, SugarCRM focuses on Customer Relationship Management (SugarCRM, 2006). Other packages, like Compiere, Open for Business, and ERP5 claim at least partial support of ERP, CRM, and even SCM functions but only to a limited extent (see Compiere, 2006; OFBiz, 2006; ERP5, 2006). Compiere, for example, provides only limited manufacturing and SCM support (Compiere Functionality, 2006). Using a façade to access these systems could allow them to be combined to supplement missing features in some packages.

A final advantage of the façade pattern is that it allows layers of subsystems to be created. In the current version of the web services façade, the subsystems have been grouped into administrative, product-oriented, and procurement areas. This reduces the complexity of the façades, since they may use other façades for access.

When applied to distributed enterprise systems, the façade pattern provides the additional advantage of reducing network traffic by allowing clients to access coarsely grained services. The Session Façade pattern, for example, is a common J2EE (Java 2 Platform, Enterprise Edition) pattern that allows clients to access Session EJBs (Enterprise JavaBeans) instead of working directly with the subsystem's business components (Alur, Crupi and Malks, 2003). Session Façades are often used to provide a services layer to the underlying system implementation. The web services façade created by the Mobile Agents in SCM project (MIB, 2006) provides the same benefits by allowing clients to access the underlying business components via web services. This simplifies access to the ERP system for clients who can access the services via proxies or dynamic invocation.

Compiere Architecture

To understand the design of the web service façade, a brief discussion of Compiere's architecture is necessary. Although Compiere is migrating to an n-tier J2EE architecture, in its current form, it can be best understood as a simple client/server application with Swing-based Java clients directly accessing a database on a remote server. Compiere allows its user interface to be customized by providing an application dictionary that contains metadata about the fields and their relationships. The application screens are then dynamically assembled at run-time by referencing the metadata stored in the application dictionary, which generally corresponds to the underlying database structure. Among other information, the application dictionary specifies how these fields map to database tables, their domains and constraints, and specifies any ancillary logic that must be executed when a button is clicked or a field changes. At run time, a screen is dynamically constructed using information from the application dictionary. The advantage to this approach is that is simplifies customizations like adding additional fields or field restrictions—these data-level changes can be easily propagated to the client's graphical interface without requiring code changes.

Although Compiere is refactoring some of its business logic to run within a middleware layer, several features still rely on Oracle PL/SQL (Procedural Language/Structured Query Language) stored procedures (Compiere General Limitations, 2006). The clients also differ as to how they perform their business logic, with different types of clients containing alternate code versions which accomplish the same task. This presents a challenge, since similar business logic can be located on the client, server, and potentially intermediary (e.g. an alpha JSP interface) layers. Compiere includes the JBoss J2EE application server in its distribution, but does not make heavy use of (EJBs) or message-oriented-middleware. Since the majority of the system

interacts directly with the database, the J2EE server is optional for most operations, and is only required for some background accounting operations. Compiere has written their own persistence mechanism instead of relying on container-managed persistence through entity EJBs or on object-to-relational (OR) tools like Hibernate, TopLink, or JDO (Java Data Objects).

Compiere's Peculiarities and Derived Architecture for the Web Service Façade

Compiere's architecture resulted in four problems: the lack of a business-level API, the self-management of multithreaded operations, the tight coupling between the GUI and the database, and the fine granularity of the data-level API.

The primary downside of Compiere's architecture for interfacing applications is that it lacks a business-level API. In a typical enterprise-level system like Compiere, reusable business components are leveraged across different user interfaces so that business logic can be maintained in one location. Compiere's implementation, however, places some business logic within database stored procedures, some within code modules, and some within the client's GUI. This eliminated the possibility of loosely wrapping an existing business-level API to provide to clients. Additionally, Compiere's logic often assumes that it is running within an application client container, and as such, violates the J2EE specification when run within an EJB container. For example, Compiere uses its own thread management for asynchronous processing, which works fine when running as a single-user application but can cause problems when run as a middleware component where user threads are centrally managed and one component may serve multiple concurrent requests. Since large numbers of clients—such as mobile agents—may access the system concurrently, access to potentially dangerous code should be centralized. Since Compiere's graphical user interface is closely tied to its database structure, another integration possibility was to access the database directly using JDBC (formerly meaning Java Database Connectivity). Unfortunately, the frequent database changes accompanying new versions of Compiere would break this type of tightly coupled interface code that depends directly on the database structure and data. Compiere provides a data-level API via its customized persistence code, so another integration option was to loosely wrap Compiere's existing data-level API with web services. In this scheme, we would create a code generator that would use the application dictionary and Compiere's existing database persistence generators to automatically generate a web services interface directly corresponding to Compiere's interface. However, the granularity of Compiere's data-level API was inappropriate for a networked application, particularly one with many clients. For example, dozens of network calls would be required to retrieve data about a simple product. Furthermore, since the business logic was not centralized, this API would have only allowed read-level access to Compiere data.

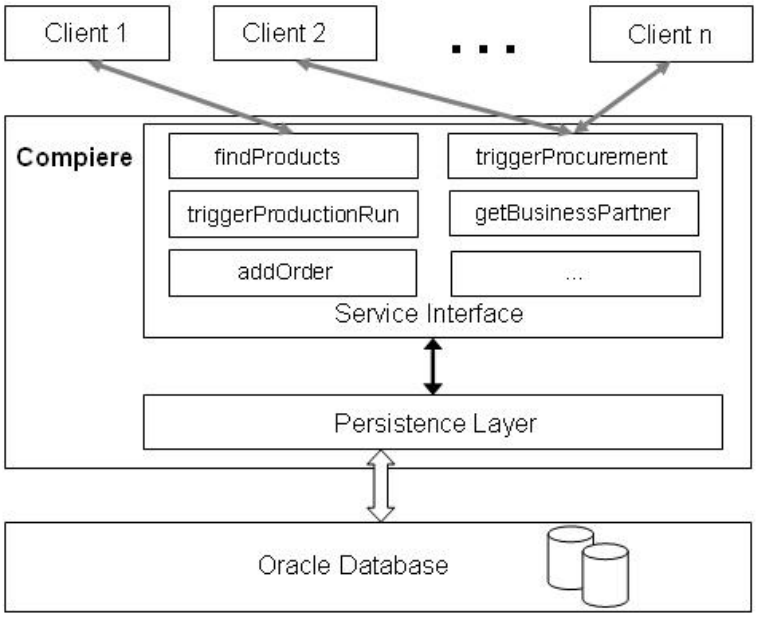

Figure 1. Compiere Architecture with Web Service Façade

In the end, the resulting architecture is outlined in Figure 1. The service interface or web service façade can receive service invocations from several clients and provides interaction with the ERP system via Compiere's persistence layer. In the case of Compiere, this is normally ERP data from the Oracle database. The web service façade is essentially an additional layer to the existing Compiere architecture.

WEB SERVICE FAÇADE IMPLEMENTATION

The specific Compiere services to expose (see Figure 1) have been chosen to support the requirements of the Mobile Agents in SCM project. To better understand the underlying needs a typical scenario could be the following: A member of the network wants to check if a required product is available and therefore checks it with the "findProducts" service (assuming that the supplier grants this right). The "findProducts" service provides information to answer the following questions. Is this product available from the supplier? If yes, how many items are on stock and is this product a single item or set consisting of other products? If the retrieved information matches the member's expectations then the "addOrder" service allows ordering the product(s). If the member is interested in the current order status he could check it with the "getOrder" service. With this service the necessary information including order details and order status are provided.

In order to invoke these web services, clients access a WSDL document to obtain the necessary message formats, protocols, and addresses. In most cases, clients will use tools supported by their development platform to generate interfaces, network stubs, and utility classes. These generated classes can then be used to exchange SOAP messages with the server. The majority of services support retrieving business objects from the server. Although saying "object" here, it is important to remember that these are actually just data snapshots of objects stored within Compiere and transferred using SOAP. Two examples for such objects (currency and product) are shown in Figure 2.

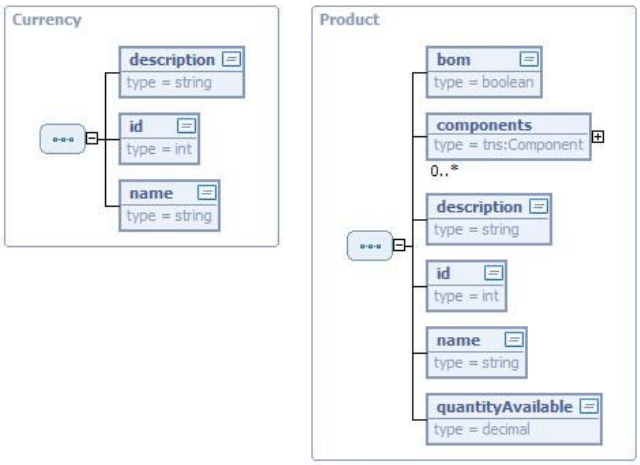

Figure 2. XML Schema representation of Currency and Product

A typical web service façade operation is "getProduct," which retrieves detailed product information including the bill of materials and available quantities. A client wishing to access this data would invoke the "getProduct" service by sending the SOAP request envelope in Figure 3, setting the string parameter to the desired product identifier. This request would be received by the web service façade and delegated to the product façade for execution. The product façade would collect the product data by making dozens of method calls to Compiere's persistence layer. To determine the current product availability, a database stored procedure included with Compiere is additionally executed for each item within the bill of materials. Next, a data transfer object is created and populated with the assembled product data. This object is then serialized into a SOAP response envelope and returned to the client.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns0:getProduct
      xmlns:ns0="http://compiere.scmws.mib.uniffo.de/">
      <String_1>1000001</String_1>
    </ns0:getProduct>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 3. "getProduct" SOAP request

To overcome the problems outlined in the section about Compiere's peculiarities, a strategy was adopted that accesses Compiere data through its customized persistence mechanism wherever possible. Since this persistence layer is generated based on the database structure, the change impact of new releases is minimized. Direct calls to the database are made only where necessary, such as when accessing logic found only within an Oracle stored procedure. Since the clients are accessing higher-level façade methods, they are isolated from any code changes that occur between the façade and Compiere. To reduce the amount of network traffic, the objects and services exposed to the client via the web services façade layer are more coarsely grained than those available from Compiere. As an example, consider the steps necessary to trigger a production run within Compiere. First, the products to be produced and the quantities necessary must be identified. Next, a production header must be created. After adding a production plan, the create production process triggers the generation of production lines. Finally, the postproduction process updates the quantities and accounting information to reflect the production. To hide the complexity from external clients within the web services interface, these processes are all contained within a single service called "triggerProductionRun".

Three Java packages contain all of the non-generated code for the web services façade, one for the web service façade interfaces and their implementations, one containing data transfer objects, and one for white box tests (i.e. tests that rely on knowledge beyond the public web service interface). The first package contains web service façades specific to three subsystems: procurement functions, product-related functions, and administration functions. Additionally, a higher-level façade groups the functionality of these subsystems and presents them to clients as a single interface. The second package contains data transfer objects, which are used to group several of Compiere's finely grained business objects. Data transfer object design patterns create a custom object containing all required data for a particular operation (Crawford and Kaplan, 2003; Alur et. al, 2003). We use these objects to transfer products along with their entire Bill of Material (BOM) in one method call.

RESULTS AND IMPLICATIONS

The implemented web service façade allows external clients to access and use ERP business logic and pertinent data via the Internet. Typical information useful for e.g. monitoring, SCM or decision support applications like inventory and order status can be obtained. The web service façade simplifies the way this data is obtained using web services based on standard web technologies like the Extensible Markup Language (XML) and the Hypertext Transfer Protocol (HTTP).

As mentioned before, specific requirements regarding an ongoing research project affected the implementation. Since the entire functionality was not needed, logistics and production functionality have been addressed more than e.g. financial aspects. Creating a generic façade would have not been feasible for this case, i.e. the façade is not built in a "plug and play" sense. Therefore, using the façade with alternate products necessitates API and code changes, e.g. tackling protocol dependencies in interacting with other systems. Since many ERP systems provide more information then necessary, additional data filtering would be required or the façade would have to be extended. This approach is available as open source so it is possible to change the façade based on project-specific requirements.

The question of service granularity is ambiguous. First, the grade of generalization and specialization must be specified. If the service design is too fine-grained, even small changes in the underlying system are able to interfere with the façade. If the services are too coarse-grained, there are difficulties in matching requirements with service functionality. Another question is how many details are necessary. If too many details are provided, then there is a risk in getting too product specific. If it is

designed too simply, then a generic approach would be possible, but it could quickly become trivial. At the current state, the presented solution is more or less a compromise and there are ongoing experiments on these issues. In the end, a business-ready approach needs to apply standards like RosettaNet (RosettaNet, 2006) to work in distributed heterogeneous environments.

On the implementation side, one would assume that tools have matured since web services are heavily used today. Still, perhaps surprisingly, the most challenging process in creating this interface was in creating and generating the necessary deployment descriptors. Despite the wide availability of tools, incompatibilities, interoperability problems, and generation bugs limited their utility. Issues found in both Apache Axis (a SOAP implementation) and the Java Web Services Developer Pack descriptor generation were eventually resolved by mixing and matching these tools via the Apache Ant build tool.

Nevertheless, several problems remain unsolved:

- One of our requirements is to simulate a supply chain with several member organizations, which are referred to as tenants in Compiere parlance. To accomplish this task, Compiere can be run in an ASP (Application Service Provider) mode that supports multiple concurrent installations of separate tenants. Currently the web service façade is unable to differentiate between separate tenant installations when Compiere is run in ASP mode.
- User-level access to Compiere is incomplete, i.e. to fully benefit from Compiere's login mechanism and ASP support, further enhancements of the services are necessary.
- Security issues exist because web service clients can request data from multiple Compiere tenants if they run on the same physical machine. Furthermore, there are currently no security mechanisms to ban bogus clients.
- The partial lack of Compiere functionality (as opposed to e.g. SAP's functionality) is sometimes a problem regarding the design of the façade.
- Web service semantics would be a significant enhancement.

RELATED WORK

Commercial ERP systems from the big vendors like SAP and Oracle already support web services and allow their use to expand ERP functionality. Although highly embraced by ERP users, this functionality cannot be taken for granted in the ERP sector at large (Knorr, 2004). Furthermore, a considerable amount of IS research in this area already exists, but is mostly concentrated on the enhancements of web service technology in itself or how it could be (successfully) applied to business problems (Alt et al., 2005). Nevertheless, several approaches exist to enhance ERP systems with a web service interface (e.g. Sharma and Kitchens, 2003; Jankowska and Kurbel, 2005). One of the main problems that these approaches face is how to aggregate the business functions into coarse-grained services for external clients—potentially including mobile devices. As opposed to these approaches, this research sought a working solution for EAI in the area of SCM under the limiting factor of Compiere's sometimes insufficient functionality, where the application of web services seems to be a promising approach. To overcome drawbacks in using web services, ongoing research in this field continues to seek solutions to more complex EAI scenarios using semantic web service principles (Haller, Gomez, and Bussler, 2005). This research could enhance our findings and solve typical EAI problems like dynamic service discovery and invocation.

CONCLUSIONS AND FUTURE WORK

Open source business applications like the ERP system presented are close to their commercial counterparts. Although sometimes lacking in functionality, they nevertheless find their way into business environments. When enhancing these systems with a web service façade, their open access to source code allows exploring the implementation directly. This greatly simplifies creating coarsely grained services that aggregate lower-level functions.

The proposed façade for the Compiere ERP system is specific to the requirements, i.e. in our case to facilitate SCM application access to required ERP business logic and data. As the number of services continues to increase, additional façades may be necessary. As the requirements of this façade mature, it may be necessary to create a more robust architecture, such as using a web service broker to coordinate and control service execution. Depending on the amount of orchestration required to support new interactions between services, an additional possibility is to use the Business Process Execution Language (BPEL) to combine these services into even higher levels of abstraction or coordinate interaction between different services to support more realistic business processes.

Further research effort is needed to make the proposed web service façade capable of participating in a dynamic environment like today's supply networks with dynamic service discoveries and invocations. Nevertheless, this work provides some first

steps on how the Compiere ERP system can interact with and be integrated into service-oriented platforms like SAP NetWeaver.

ACKNOWLEDGMENTS

This work is supported by the Federal Ministry of Education and Research of Germany via the research project "Mobile business processes and user interfaces based on Wireless Internet".

REFERENCES

1. Alt, R., Gizanis, D. and Legner, C. (2005) Collaborative Order Management: Toward Standard Solutions for Interorganizational Order Management, *International Journal of Technology Management*, 31, 1/2, 78-97.
2. Alur, D., Crupi, J. and Malks, D. (2003) Core J2EE Patterns, Best Practices and Design Strategies, 2nd Edition, Sun Microsystems Press, Palo Alto.
3. Compiere (2006) http://www.compiere.org, Accessed 2006-02-14.
4. Compiere Changes (2006) Release 2.5.1, http://www.compiere.org/product/rel251.html, Accessed 2006-02-14.
5. Compiere Functionality (2006) http://www.compiere.org/about/whatitdoes.html, Accessed 2006-02-14.
6. Compiere General Limitations (2006) http://www.compiere.org/product/openItems.html, Accessed 2006-02-14.
7. Compiere Pilot (2006) http://www.compiere.org/about/success/pilot.html, Accessed 2006-01-25.
8. Crawford, W. and Kaplan, J. (2003) J2EE Design Patterns, O'Reilly, Sebastopol.
9. Dreiling, A., Klaus, H., Rosemann, M. and Wyssusek, B. (2005) Open Source Enterprise Systems: Towards a Viable Alternative, in IEEE Computer Society (Eds.) *Proceedings of the 38th Hawaii International Conference on System Sciences*, Hawaii, USA, http://doi.ieeecomputersociety.org/10.1109/HICSS.2005.473, Accessed 2006-01-25.
10. ERP5 (2006) http://www.erp5.org, Accessed 2006-02-14.
11. Freeman, Eric and Freeman, Elisabeth (2004) Head First Design Patterns, O'Reilly, Sebastopol.
12. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995) Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading.
13. Greenemeier, L. and Kontzer, T. (2005) Open Source Goes CORPORATE, *InformationWeek*, 1057, 38-45.
14. Haller, A., Gomez, J. M. and Bussler, C. (2005) Exposing Semantic Web Service principles in SOA to solve EAI scenarios, In *Proceedings of the Workshop on Web Service Semantics, Towards Dynamic Business Integration*, International Conference on the World Wide Web (WWW2005), Chiba, Japan, http://sw.deri.org/~haller/publications/SSOA-WWWW2005.pdf, Accessed 2006-02-03.
15. He, H. (2003) What is Service-Oriented Architecture? O'Reilly XML.com, http://www.xml.com/pub/a/ws/2003/09/30/soa.html, Accessed 2006-02-03.
16. Jankowska, A.M. and Kurbel, K. (2005) Service-Oriented Architecture Supporting Mobile Access to an ERP System, in Ferstl, O. et al. (Eds.) *Wirtschaftsinformatik 2005 - eEconomy, eGovernment, eSociety*, Heidelberg, Germany, 371-390.
17. Knorr, E. (2004) ERP Springs Eternal, *Infoworld*, 26, 12, 44-54.
18. MIB (2006) M-Agenten im SCM, http://mib.uni-ffo.de/index.php?id=12&type=1, Accessed 2006-02-14.
19. MySQL (2004) Quickpoll: Which ERP Applications are currently in use or will be used at your company?, http://mysql.mirror.luxadmin.org/tech-resources/quickpolls/erp-applications.html, Accessed 2006-01-20.
20. OFBiz (2006) http://www.ofbiz.org, Accessed 2006-02-03.
21. Perens, B. (2006) The Open Source Definiton, http://www.opensource.org/docs/definition.php, Accessed 2006-01-25.
22. Raymond, E. (1999) The Cathedral and the Bazaar, *Knowledge, Technology & Policy*, 12, 3, 23-49.
23. RosettaNet (2006) http://www.rosettanet.org/Rosettanet/Public/PublicHomePage, Accessed 2006-04-26.
24. Sharma, S. K. and Kitchens, F. L. (2003) A new approach for mobile communications: web services using flexible services architecture, *International Journal of Mobile Communications*, 1, 3, 301-311.
25. SugarCRM (2006) http://www.sugarcrm.com, Accessed 2006-02-14.