Cooking the Web-ERP

A Practical Recipe to Stir-up Monolithic Enterprise Information Systems Using DOC- and XML-Standards

Michael Gillmann, Joachim Hertel, Christoph G. Jung, Günther Kaufmann, and Michael Wolber

infor: business solutions AG,
Hauerstrasse 12, DE-66299 Friedrichsthal, Germany
http://www.infor-business-solutions.com/

Abstract. When it comes to controlling and optimising information- and value flows in manufacturing industry, *Enterprise Resource Planning (ERP)* is still the preferred option. Despite the immense potential, there is currently a particular caution to be observed in the market for ERP systems. One reason is a prevalent uncertainty about the ongoing 'e-revolution' of traditional business processes. Because the standard ERP systems regard themselves as the epicentre of any enterprise architecture, they cause exceeding consequential costs for reengineering a business. The term *ERP-II* has been coined to describe alternative information system architectures in which flexible and customized federations of smaller *business components* interact, even over enterprise and intranet boundaries, by means of a platform-neutral *communication bus*. It is the central challenge for the ERP vendors at the beginning of the new millennium to evolve and migrate their existing logic and customer installations towards this vision. In this paper, we would like to share our experiences in transforming a particular ERP product for mid-sized businesses into an ERP-II platform. To realise the proposed business bus, we chose a modern middleware based on the ubiquitous *eXtended Markup Language*. To realise a suitable, object-oriented runtime environment for business components, we chose the powerful *Java 2 Enterprise Edition*™. Both flavours have been conveniently blended into the notion of *Business Web Services* to cook a new generation of business processes inside the *Web-ERP*.

1 Introduction

1.1 The Demise of Monolithic ERP Systems

In the current age of business globalization and worldwide marketing, midsize companies too are forced to compete against a hoard of competitors from all over the world [1]. It is therefore imperative for them to increase the quality of both their value-added and information flows. With such growing requirements, one is able to observe a corresponding progression of *Enterprise Information Systems* (EIS) from *Inventory Control* (IC) via *Material Requirements Planning* (MRP), *Manufacturing Resource Planning* (MRP II) up to *Enterprise Resource Planning* (ERP) systems [13].

R. Meersman, Z. Tari (Eds.): CoopIS/DOA/ODBASE 2002, LNCS 2519, pp. 602–617, 2002.

When IC systems made the first steps towards the electronic administration of how to satisfy a company's raw material demand, MRP added conception and execution planning to that. Today, ERP optimises enterprise-wide and simultaneous production processes based on metrics such as warehousing costs, capacity utilisation, throughput and adherence to delivery dates. Figure 1 shows the most common architecture of ERP products, such as SAP R/3 [20] and infor:COM [9]. In these products, *business modules*, such as warehousing, scheduling, accounting, purchasing, sales, and operating data, are appended into a fairly complex software package.

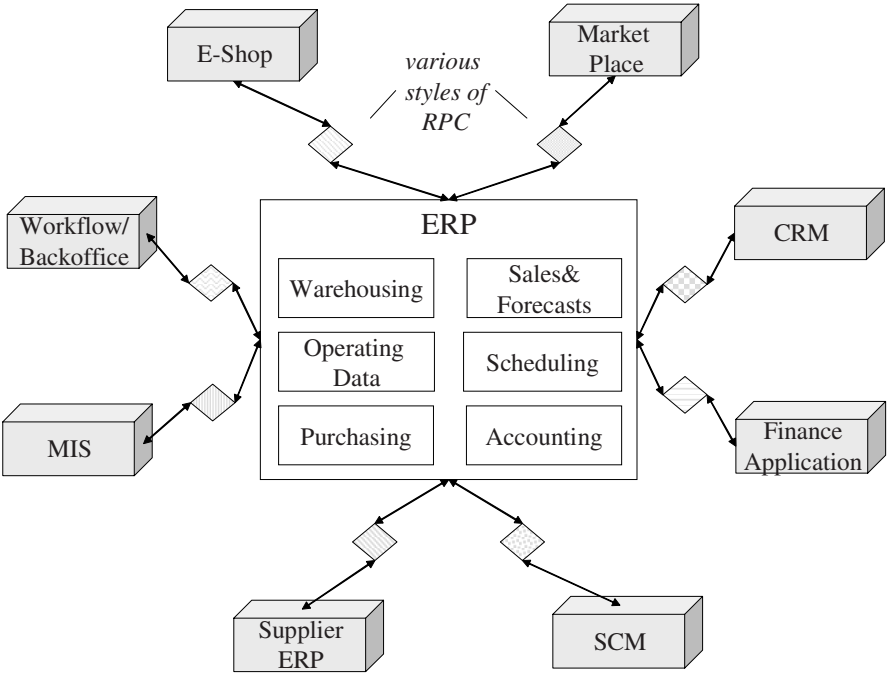

Fig. 1. Monolithic ERP system with extensions

Despite the existing potential especially in the mid-market, ERP sales figures have increased significantly less than analysts expected [15] and threatened the existence of some traditional companies with good reputations, such as Baan [3]. One reason for this phenomenon is the fear of exploding, unmaneagable costs that may occur in conjunction with ERP software projects and their further evolution [21, 22].

Indeed, monolithic ERP systems such as those shown in Figure 1 have serious disadvantages when business processes change regularly [24]. If legacy applications and software of other vendors (e.g. finance applications, backoffice products, e-shops, management information systems, supply chain queues) are to be integrated, the development and the maintenance of dedicated interfaces becomes necessary [23]. Technically, such interfaces often rely on various styles of *Remote Procedure Call* (RPC) [17] and easily degenerate into hardly maintainable 'spaghetti' landscapes centred around the central ERP system.

One side effect is that the resulting conglomerate systems are nearly impossible to interlink with complementary business information systems of customers or suppliers via standardized formats over the Internet. This is urgently needed for, e.g. realising true supply chain automation. Another result is that common ERP systems are not able to be easily downsized. So, the customers buy and install a lot of business logic that they may never need.

Therefore a mismatch between the highly sophisticated economical know-how and an antiquated technical core of the ERP software exists that results in expensive maintenance and limitations.

1.2 ERP-II – The Business Backbone of (Virtual) Enterprises

A new approach to enterprise information systems has been cultivated under the umbrella title Enterprise Application Integration (EAI) [23]. EAI extends the use of engineering techniques such as those known in component-based programming [26] into the area of business logic in order to guarantee the customers a mid- and long-term return on investment [7,18,21].

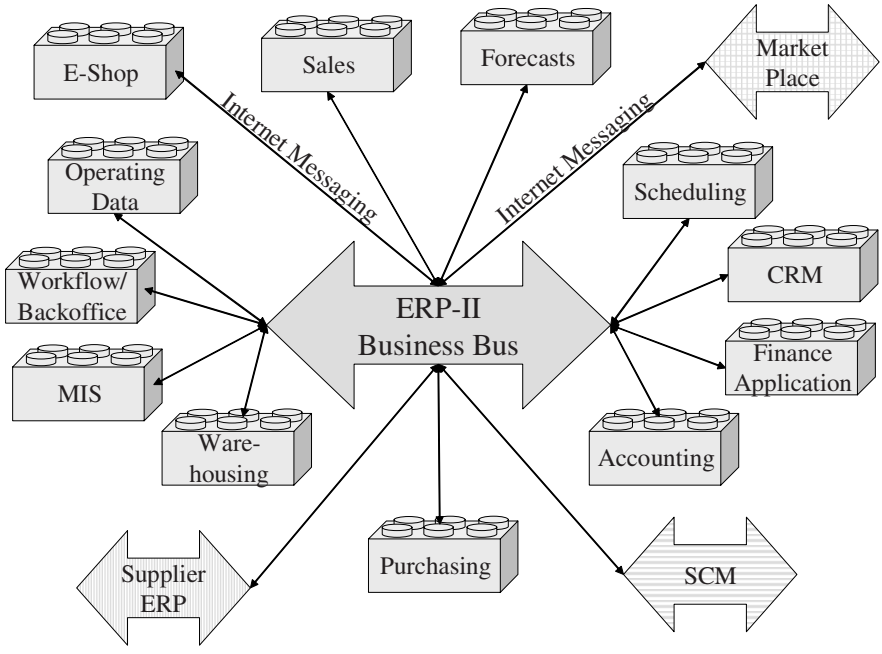

Fig. 2. ERP-II Bus with Business Components

From the perspective that we have depicted in Figure 2, a complex and customized application emerges by the interplay of an upwards and downwards scalable federation of individual software components, each optimally designed for partial functionality [13,24]. These components communicate by means of a universal

software bus that is a generalization of the *Distributed Object Computing* (DOC) model, such as pioneered by DCOM and CORBA [19].

The fundamental difference from the 'legacy approach' of Figure 1 is to stop thinking of the ERP system as quasi-middleware, but to dissolve, componentise, and decouple the ERP monolith successively by the middleware [18]. It is only by this approach that modifications in the business processes can be reasonably accommodated and supported by the ERP core. It is only by this approach that specific customer requirements beyond superficial GUI features may be suitably addressed. And it is only by this approach that future business components can be fully integrated into the so-called 'standard logic' to interact in the networked virtual enterprises of the future.

However, if we do not formulate further requirements to the new central notion of the business bus, which we now call *ERP II* [33], then this vision will not be achieved and has already been missed in a vast number of EAI projects [29]:

1. Firstly, it must be ensured that the horizontal view of Figure 2 coincides with the pertaining vertically-tiered architectures [32] to offer a suitable runtime environment for business components.
2. Secondly, such an enterprise bus transporting conceptually rich business messages over the Internet must not build on a connection-oriented communication protocol that is dependent on a particular computing platform. For example, DCOM is tightly bound to the Microsoft™ *Dynamic Network Architecture* and hence to the 32bit versions of the Windows™ operating systems.
3. And thirdly, the business bus must allow a suitable migration strategy from the huge and well established code base written in *4th Generation Languages* (4GLs, such as SAP´s ABAP, Microsoft´s VisualBasic, and infor´s LJ4) into the Distributed Object Computing world. This is certainly a strategy that lies between the two extremes of the 'big bang' and 'legacy' scenarios.

1.3 Contribution and Outline

In this paper, we document the experiences and results of a development project that has been run by infor business solutions AG between 2000 and 2001. In this project, the three aforementioned fundamental requirements on a business bus have been jointly addressed in transforming the existing ERP product infor:COM into a component-oriented ERP-II system that is particularly well-suited for mid-sized manufacturing companies:

1. A three-tier business component runtime environment (Section 2.1) has been based on the object-oriented principles of the *Java 2 Enterprise Edition* (J2EE™) [25]. The runtime core has a small footprint and is easy to extend by additional components.
2. Secondly, business components have been added to by Internet technologies, more specifically XML-based *Web Services* [6]. The resulting 'business federation' is highly suitable for building globally networked applications (Web-ERP, Section 2.2).

3. Thirdly, due to the modularity of J2EE™ and the openness of the Web Service paradigm, we have been able to build a hybrid, albeit integrated product in which state-of-the-art Java components coexist and mutually interact with established 4GL code. In Section 3, we address technical and logical challenges of our ongoing work to incrementally port 4GL modules into coherent ERP-II components.

The presented ERP-II solution including an initial set of migrated and recently developed business components, such as Internationalization, Measures, and Sales Forecasting, is already installed and successfully tested at selected customer sites and will be officially released in 2003. We conclude in Section 4 by a discussion of the suitable (open source) tools that we used.

2 An ERP-II System Based on J2EE™ and Web Services

2.1 Logical Architecture and Runtime Environment

Our architectural blueprint (Figure 3) is based on a traditional three-tier model consisting of *Presentation*, *Application* and *Persistence*. Although the separation of client-side presentation control from server-side application logic goes back to early terminal-based systems, it is today receiving a lot of attention, again, because it enables the simultaneous access by browser-based user interfaces on the one hand and highly-integrated and user-friendly rich clients on the other hand [32].

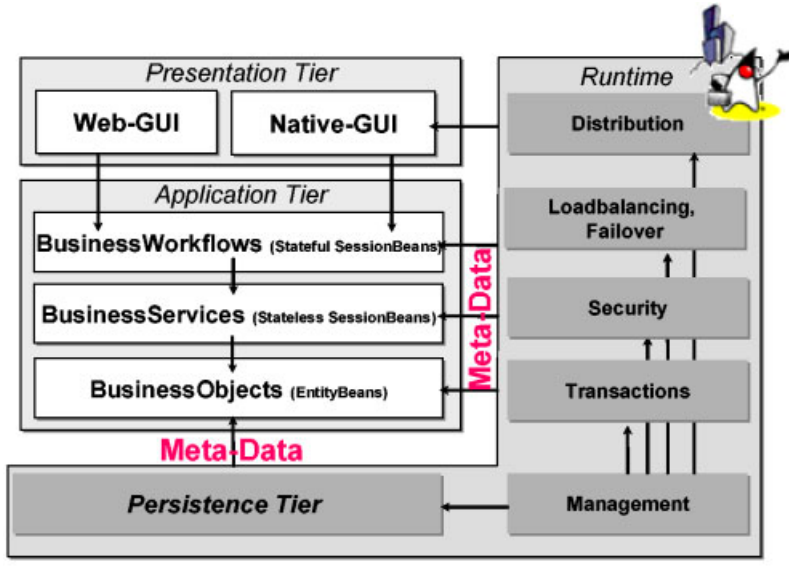

Fig. 3. Declarative Programming in the J2EE™ Model

Similar to other business architectures, such as *San Francisco*™ [2], our application tier is further divided into the *Business Object* (BO), the *Business Service* (BS) and the BusinessWorkflow (BW) tiers. Business objects such as documents, business partners, or sales items represent the persistent entities of an application. Business

services such as document administrators or warehouse services implement and publish *Business Methods* to manipulate the BOs. Examples for such business methods are the copying of a document, the retrieval of an archived invoice, or the checking of deliverability. Business methods correspond to individual steps of the *Business Processes*, such as the acceptance a new customer order, that are instantiated by concrete Business Workflows and hence supported by the overall *Business Application Programming Interface* (BAPI).

The visibility between the tiers – BWs can import BSs which can import BOs, but not vice versa – may appear distinctly 'un-OO-ish' at the first sight. However, it follows the well-known *'strategy'* design pattern [31]. Representations that stay constant over a long period of time have to be put into the centre of a software architecture. For example, in the recent, from a business standpoint very turbulent, years, typical business partner contact details have only changed by the addition of an e-mail address and a homepage URL. In contrast, there have been highly mutable parts of the business logic, for example for accomodating business partners in the still evolving e-procurement area. Following the theory, those parts are to be separated into a different service tier of an extensible architecture.

A similar argument holds for the unidirectional dependency from the persistence tier to the BO tier as seen in Figure 3. This means that BOs have no insight into, e.g. database- and transaction-related code packages. Instead, the persistence tier that is a part of the generic runtime environment is able to inspect BOs and to persist BOs state in a relational database with the aim of reinitialising it later 'automagically'. For the sake of performance and simplicity, the impact of this design decision has been neglected in many *object-relational* (OR) persistence frameworks in which the application must include customized database interaction code within the BOs.

The result is a cluttering-up of the business logic with driver programming, particular SQL dialects and error-prone transaction processing. This may be tolerable for small and short-lived applications, but leads to a quick 'death by maintenance' when applied to domains of the size of ERP. Furthermore, due to recent improvements in meta-programming in both Java and C#, building generic persistence frameworks is neither more complicated nor subject to severe performance drawbacks.

The elegant methodology of separating such technically detailed, non-business related and fault critical functionalities (as well as persistence we can also count security, transaction management, distribution, management, caching etc. to these) from the application code and to embody them via flexible code annotations, so-called '*meta-data*', into the runtime environment is called *Declarative Programming* [14][1] and has been successfully introduced by the *Microsoft Transaction Server* (MTS). Declarative Programming continued to grow into a quite substantial market especially through the *Java 2 Enterprise Edition* (J2EE™) [25].

---

[1] This is, at the first sight, not to confuse with the identical term used in *Logic Programming* (LP). But interestingly, the underlying equation of "Algorithm = Logic + Control" [34] applies to our domain in exactly the same manner.

In J2EE™, business workflows and business services are called (stateful and stateless, respectively) session beans; BOs have a natural counterpart in the so-called entity beans. Because J2EE™ is merely a specification that is to be followed by a number of compliant and differently targetted products (application servers), we could build a small and mostly vendor-independent bridge that allows to run our BO/BS/BW-based ERP logic in various application servers. With the advent of J2EE™ 1.3, the capabilities of server-managed entity bean persistence have been substantially increased and now compete with stand-alone OR products.

For our target market of mid-sized businesses, we currently prefer the highly customizable Open Source application server JBoss [10] to the high-end commercial products, such as BEA WebLogic [30]. This is because we needed to slightly enrich the *Enterprise Java Beans*™ (EJB) programming model in two respects. First, object- and component models coincide in EJB™. Secondly, the binary and strongly-typed *Internet Inter-Orb Procotol* of the CORBA specification [19] is proposed as the standard communication language.

We found it initially very hard to assemble a complex ERP-II system from such a finely granular and highly coupled model. We found it even harder to interface the resulting EJB™ logic from windows-based native and web clients. However, as we will describe in more detail in the following section, it is possible to extend these concepts into a suitable business bus architecture.

2.2 Business Components and Web Services

The definition of a *Software Component* in computer science [26] describes a self-contained logical unit that can be individually developed, produced, distributed, installed and run in loose coupling with other components by a suitable runtime environment. In the ERP case, we define a *Business Component* such as sales order processing, rough planning, and warehouse capacity, as a coherent set of business objects, services, workflows and their corresponding meta-data which together form a reasonable partial functionality of the total system (see Figure 4).

Business components are hence considerably smaller than the standard ERP modules such as sales, scheduling or warehousing, whose grandiose amount of logic calls for redundancy-diminishing distribution and flexible decoupling. Business components are, on the other hand, significantly larger than a single and by itself insufficient JavaBean as proposed by the initial EJB™ component specification.

In the J2EE world, a business component has rather a matching concept in terms of an *Enterprise Application,* which is a form of bundling a complete installation in a single ZIP file. Originally, individual applications are isolated from each other, i.e. though they may run within the same virtual machine and shared memory, they may not directly exchange byte code, mutually invoke methods, etc.

As Figure 4 illustrates, our recent additions to the JBoss application server introduce this facility in order to let J2EE applications cluster together with each other, with external components, and also with various user front-ends. Within each of the business components shown (sales order processing, rough planning, and warehouse

capacity in Figure 4), we therefore keep up a purely object-oriented view dealing with interfaces, objects and method invocation, e.g. the business services inside the order processing component interact with each other via normal Corba-based method calls.

However, this form of communication cannot be chosen whenever source and target component of such an interaction differ significantly in byte code, class definitions, basic forms of representation (e.g. Java™ versus C#™ classes) or even the runtime environment (e.g. Sun´s JRE™ versus Microsoft´s Common Language Runtime™).

Hence, any interaction either over tier borders, such as the access from a Win32-based administration GUI or a browser-based e-shop, or over component borders, such as calling a deliverability computation inside a warehouse service in order to create a request approval in a sales service or even automatically transmitting a material order to a supplier system via a Biztalk™ B2B Server, has to be mediated through an additional transport layer. For this transport layer, a representation must be found which bridges the different platforms for implementing business logic (compiled versus interpreted, object-oriented versus functional, stateless versus state full) and which enables the 'pluggability' with other software busses, such as needed for B2B supply chain management and electronic marketplaces. As depicted in Figure 4, this is the ideal application of the Simple Object Access Protocol (SOAP) [16].

SOAP is a W3C specification that is based on the eXtended Markup Language (XML) [27] and describes how distributed procedure calls, in our example in Figure 4 a call of the business method `reserveItem(Item item, int quantity)` in the `WareHouse` service, are to be encoded independently of their concrete (OO) representations in an XML document which can then be shipped synchronously via the Hypertext Transfer Protocol (HTTP, HTTP/S) or asynchronously via the Simple Mail Transfer Protocol (SMTP).

The choice of HTTP and SMTP as transport protocols has many advantages over dedicated RPC protocols. Because they are standard Internet protocols, there is the possibility to easily tunnel business messages through existing firewall architectures. This would be very difficult to configure for, e.g. CORBA ORBs. Furthermore, due to the messaging character behind HTTP and SMTP, the resulting IP sessions are furthermore very short-lived which saves resources and diminishes the vulnerability of the connections against hostile attacks.

The choice of a tagged XML text format as the basic form of representation also has advantages over, e.g. the typed IIOP byte format. XML is not bound to a particular form of memory structures to be serialized from and deserialized into. XML can encode a rich set of data relationships including inheritance and polymorphism. XML does not require a special-purpose parser; event-based XML parsers are already a part of even the most embedded platforms. XML documents are very modular to synthesize. XML can be easily transformed using the *XML Stylesheet Language* (XSL). And finally, there are a standardized meta-data formats, such as XML Schema (XSD) [8] and the Web Service Description Language (WSDL) [5] which are by themselves bootstrapped through XML.

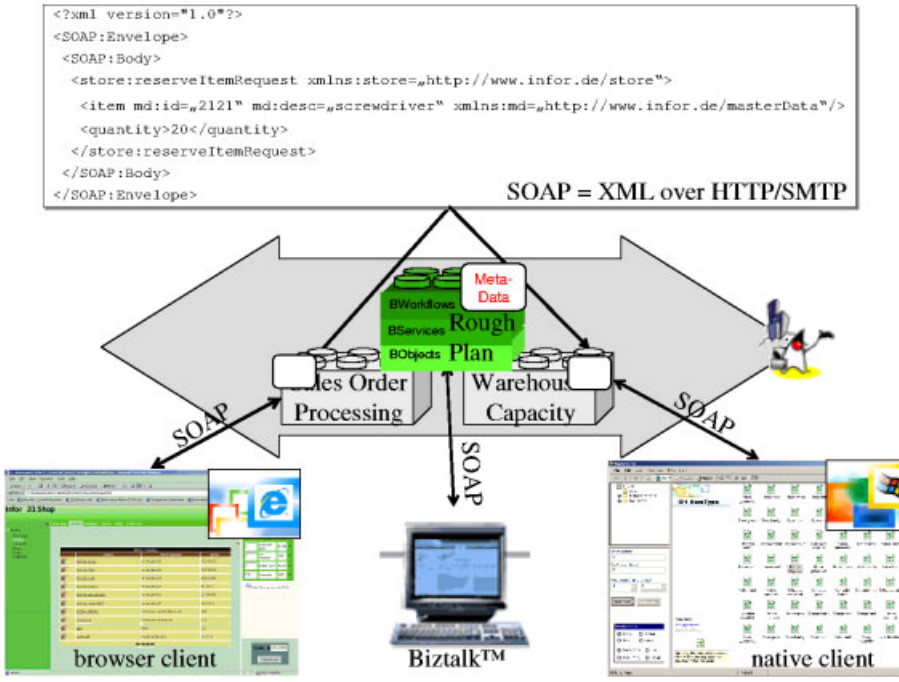

Fig. 4. Business Components ‘talk SOAP’

In Figure 5, we have demonstrated the application of these meta formats to our ERP-II architecture. Business objects, such as the `Item` entity bean, expose their data structure in the form of corresponding *XML Types* (`<Item/>`) defined in XML Schema documents (“Item.xsd”). One could look at an XML Type as obtaining a particular, platform-independent view onto the business object quite like the business object itself operates as a view into the underlying database model.

Business Workflows and Business Services, such as the `Warehouse` session bean, expose their implemented logic in the form of corresponding *Operations* (`<reserveItem/>`) defined in WSDL documents (“Warehouse.wsdl”). In a way, WSDL just puts operational semantics on top of XSD because the operations are specified by well-formed request and response messages which themselves build upon the XML schemas of the arguments, return types, and exceptions such as defined “Item.xsd”. In WSDL, operations are bundled into so-called *Port Types*, the platform-independent version of Java interfaces, and finally *bound* to particular low-level transports, such as http endpoints located on particular machines in the network.

XSD, SOAP and WSDL currently are part of all the major business platforms ranging from Microsoft .Net or popular scripting languages up to many J2EE™ products. As demonstrated, they therefore establish the platform-independent BAPI that we were looking for in order to specify the representations on the ERP-II software bus. To fully exploit the concept of globally visible *Business Web Services*, we can finally register the meta descriptions with the increasingly common capability of *Universal Description Discovery Integration* (UDDI) registries [28].

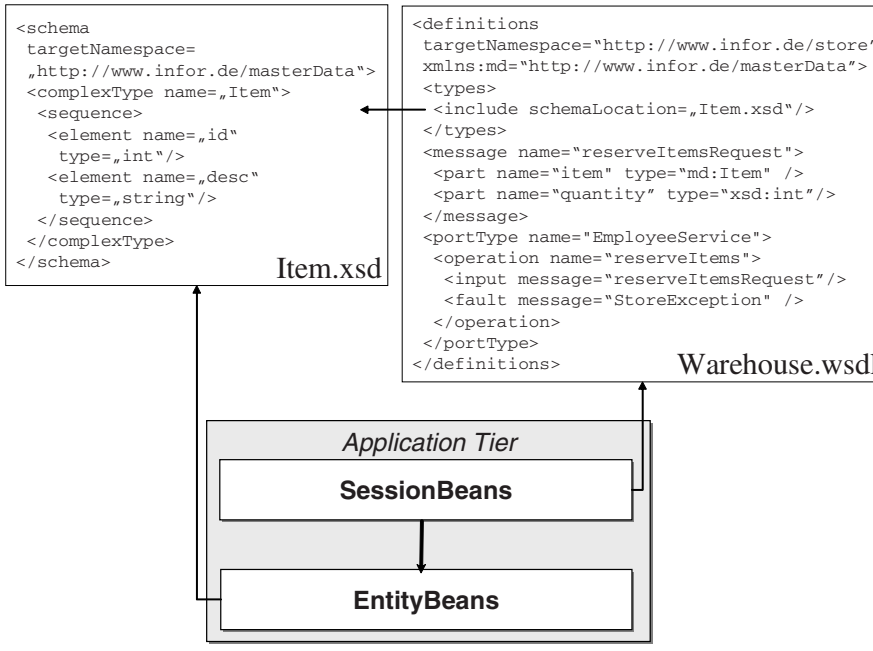

Fig. 5. WSDL and XSD describe a platform-neutral BAPI on top of J2EE™

In infor:COM, or rather JBoss3.0, the mapping between the JavaBeans and the universal XML structures and registries is another example of the convenient and highly productive declarative programming approach. In the JBoss.net project [11],an existing SOAP messaging product is incorporated to complement the existing CORBA invocation engine with an additional SOAP layer. This layer operates transparently to the application logic and is controlled by additional meta-data annotations in the deployable business components.

3 The Smooth Migration from ERP to ERP-II

We have already sufficiently discussed that the monolithic ERP architecture shown in Figure 1 has serious disadvantages in system extensibility and maintainability [13,24] especially in the current age of Internet technology and global virtual enterprises. So it has to be migrated to the new technological state of the art that we have developed in the Section 2 above. However, a big bang strategy to completely replace the legacy ERP by a brand-new ERP-II approach would significantly increase dangers and result in large (financial) risks, and not only for the ERP vendors.

Typically ERP systems such as infor:COM consist of several million lines of 4GL source code and so it is safe to say that the development of a subsequent product release takes years. Furthermore, long-term customers whose systems have been used

for years do not want to trash their experiences and familiar working processes gained in using the ERP software. Further to learn the new system from scratch. What they want is to exchange only those components that require the new functionality [12].

The solution that we have taken with infor:COM in order to resolve this opposition is a smooth transformation from an ERP system to an ERP-II system (Figure 6). This transformation comprises a technical and a logical migration path. Technically it is meant to tightly couple the runtime environments of the ERP system and the ERP-II system, i.e. making the existing 4GL logic a proper part of the freshly installed business web service bus. We will elaborate that issue in Section 3.2.

3.1 Logical Migration

Logically a smooth migration means identifying and publishing service-relevant interfaces from the existing 4GL code base. This is the harder and more resource-intensive task. However, once the technical migration problem is resolved, the logical transformation need not be done at once, but can be done incrementally and on-demand by identifying coherent portions of the ERP business logic, defining suitable replacement interfaces for them, implementing those interfaces by new ERP-II components, and finally deactivating the former 4GL code.

Figure 6 shows, for example, the detachement of a new ERP-II sales forecasting component from the older ERP sales module that we have already successfully performed in infor:COM. Through the business bus, the ERP-II component now looks at the still existing ERP modules as standard, albeit legacy components. The ERP modules interact with the new ERP-II component transparently through technical interface code that replaces the former implementation. This technique can also be be used to extend particular 4GL process logic which should not yet be totally replaced, but needs some temporary upgrade, e.g. to access partner products, perform additional messaging, etc.

The customer is thereby provided with frequent product updates (in our case starting after two initial years of research) of a modern, hybrid product. This ERP-II product is not restricted to a partial functionality of the ERP core as it would have been necessary when building it completely from scratch. In this product, even the 4GL logic not yet migrated gains value through the increased (Internet) business process capabilities of the embedding ERP-II platform as it would not have been possible, if treating the complete ERP as a single, monolithic legacy component. The classical ERP system is therefore slowly dissolved. Its enterprise-critical core functions will smoothly transform into ERP-II components.

3.2 Technical Migration

It must be noted that the success of this migration approach depends highly on the quality of the 4GL code base and the technical measures chosen. Whenever the code base becomes too big , too cluttered or cannot be processed with the help of modern

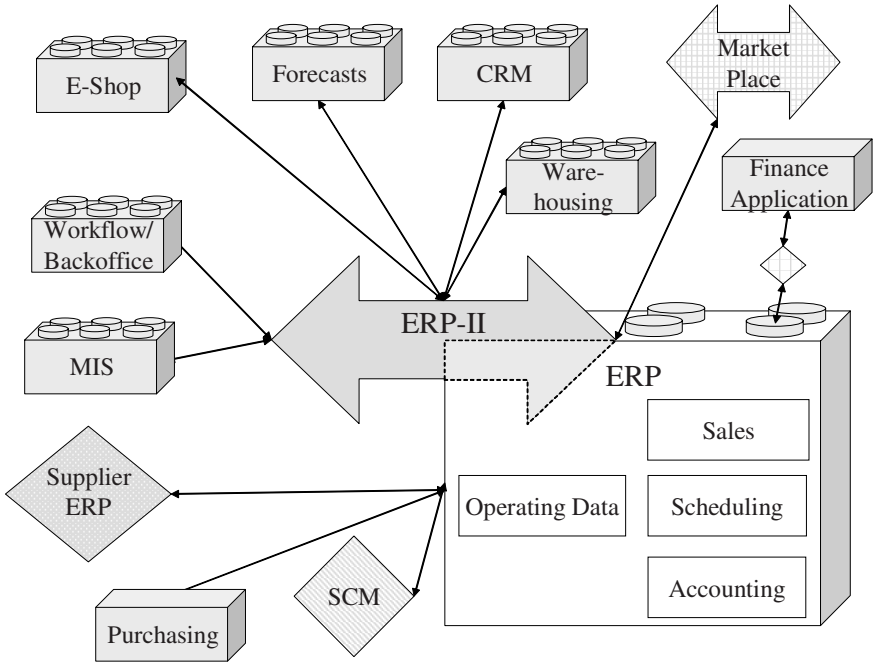

Fig. 6. Coexistence of ERP and ERP-II in a hybrid system

code analysis and cross-compilation tools, even the first step of identifying separable interfaces may fail. In infor:COM, the 4GL language is an in-house development called *LJ4*. Because of our experience with appropriate compiler technology and the size of the product which is targeted to mid-size companies, we are quite confident of the road taken. This has been confirmed by our initial successes in migrating, e.g. the Sales Forecasting logic, within the hybrid product.

However, these successes would not have been possible without carefully preparing a technical 'bed' for the now unified J2EE™ and 4GL runtime environments (Figure 7). When comparing the vertical architectures, we recognise that the modern concepts of BSs, BOs, and the persistence tier have 'outdated' counterparts in the form of LJ4 procedures operating on Vtabs which are implementations of typical relational SQL datasets. But we also recognise that infor:COM stems from a traditional two-tier background in which presentation logic and application logic reside on the same physical computing device.

To enable business processes to freely flow between the two sides, we hence need elaborated technical integration methods for each of the tiers shown (Figure 7). For example, to couple the ERP database with the ERP-II persistence engine, we have employed a distributed transaction manager to synchronize the database operations.

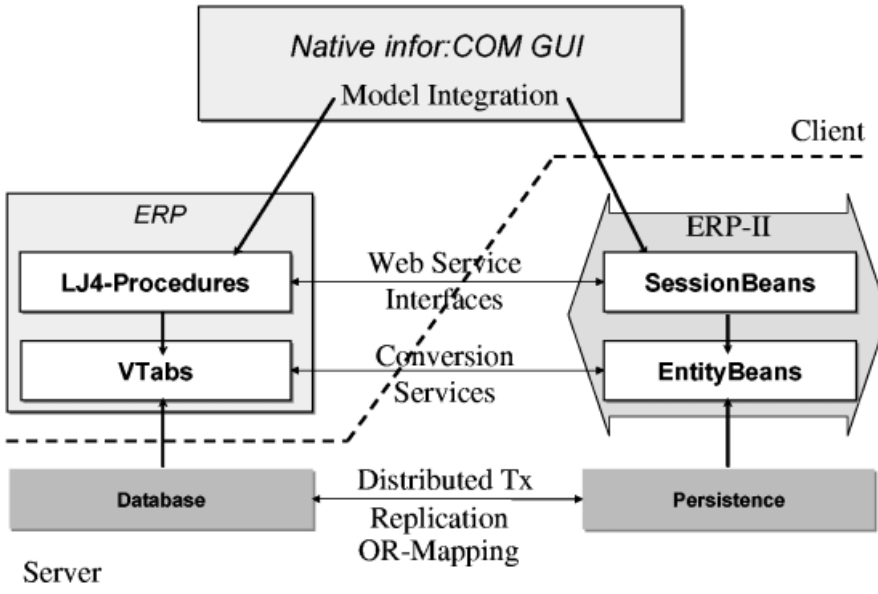

Fig. 7. Methods for Technical Migration

Where data must be present in fundamentally different schemas to feed both the ERP and ERP-II logic, we employ replication techniques. In the easier case, it is possible to map dedicated BOs onto the existing tables in the ERP system.

This is complemented on the next tier by technical services which can convert Vtab memory structures into entity bean structures and vice versa. Using the already described, platform-neutral Web Service layer on the BS and BW tiers already described, it is then possible to exchange method/procedure calls, including the exchange of data, between both platforms.

Finally, we have decided not to build two separate clients, but to retain the existing native application to which a broad community of users has been well accustomed by now and which supports their typical use cases in an optimal manner. For that purpose, we have managed to extend the existing presentation model in order to cope with the new business representations of the Web-ERP platform.

4 Conclusion

In this paper, we have presented the experiences gained while preparing infor business solutions AG's upcoming release of the infor:COM product. infor:COM 6.3 is based on a component-based extensible ERP-II architecture (Web-ERP) with a small system footprint. New and globally networked business components, such as our currently piloted Sales Forecasting component, are built upon a combination of modern DOC- and XML-ingredients. They coexist in a hybrid, albeit technically-

integrated environment with established 4GL code, such as for resource planning, which will be incrementally transformed through an ongoing logical migration effort.

The work that we have described has been eased by the consistent use of and contribution to the Open Source community. In the recent past, Open Source software has made quite a progression from operating systems (Linux), via tools and libraries (GNU), web servers (Apache) up to the first business logic environments (OpenEJB, Jonas and JBoss [10]). It is now a viable alternative to costly in-house developments as well as to ready-made, but closed-source commercial middleware.

This way, independent business software vendors can share their development efforts with respect to the application server while still competing at the level of the commercial business logic and development tools. For example, infor has developed an extension to the UML tool TogetherJ™ (Figure 8) which allows the developer to seemlessly specify, architect, implement, assemble and even debug infor:COM business components. Because of standard Java interfaces and the open deployment formats, the corresponding code is even compliant with the strict rules of the Lesser General Public License (LGPL).

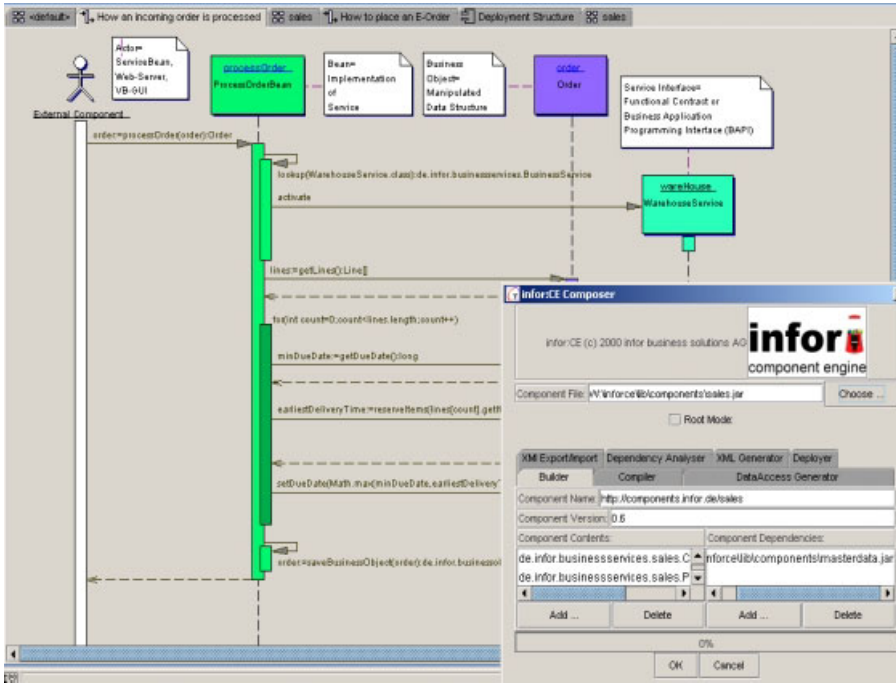

Fig. 8. Developing Business Components with UML

Acknowledgements. We would like to thank Michael Abel, Johannes Beck, Ralf Immer, Georg Röver, Darius Schier and Andreas Schörk for contributing much to the work presented. Most valuable comments to this paper have been given by Colm Stephens and Richard Hunn. Furthermore, we would like to thank the many talented contributors to the JBoss project.

References

1. R. Adhikari, ERP Meets The Middle Market, IndustryWeek, 1999
2. K. A. Bohrer, Architecture of the San Francisco frameworks, IBM Systems Journal, Vol. 37, No. 2, 1998
3. S. Baker, M. Spiro, S Hamm, The Fall of Baan, Business Week International, August 2000
4. Communications of the ACM, Vol. 43, No. 4, 2000
5. E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, Web Service Description Language (WSDL) 1.1, W3C Note, 2001, http://www.w3.org/TR/wsdl
6. E. Cerami, Web Service Essentials, O'Reilly, February 2002
7. T. H. Davenport, Realizing the Promise of Enterprise Systems, Harvard Business School Press, 2000
8. D. Fallside, Xml Schema Part 0: Primer, W3C Recommendation, May 2001
9. infor business solutions AG, A Fitness Boost for your Business, http://www.infor-business-solutions.com/cms/solutions
10. JBoss Group, !Simpler, Cheaper, Better!, The Java App-Server reference implementation, http://www.jboss.org
11. JBoss Group, JBoss.net: Integrating J2EE™ with Web-Services, http://www.jboss.org/developers/projects/jboss/dotnet.jsp
12. M. Kremers, H. van Dissel, ERP Systems Migrations, in [4]
13. K. Kumar, J. van Hilleghersberg, ERP Experiences and Evolution, in [4]
14. J. MasterMann, Transactional Programming, MSDN Magazine, June 2000, http://msdn.microsoft.com/msdnmag/issues/0600/advbasics/advbasics0600.asp
15. J. Menezes, ERP vendors´ bubble bursts, Computing Canada, 1999
16. N. Mitra, SOAP Version 1.2, W3C Working Draft, December 2001
17. The Open Group, CAE Specification, DCE1.1: Remote Procedure Call, Document C706, 1997, http://www.opengroup.org/public/pubs/catalog/c706.htm
18. A. Parker, ERP's future lies in modules rather than monoliths, ComputerWeekly, 1999
19. G. S. Raj, A detailed comparison of CORBA, DCOM, and Java/RMI, OMG Whitepapers, The Object Management Group, 1997
20. SAP AG, SAP R/3 Enterprise, White Paper, http://www.sap.com/solutions/technology/ pdf/SAP_R3_Enterprise_WP.pdf
21. A.W. Scheer, F. Habermann, Making ERP a Success, in [4]
22. D. Slater, The Hidden Cost of Enterprise Software, CIO Magazine, 1998
23. D. Slater, Middleware Demystified, CIO Magazine, 2000, http://www.cio.com/archive/051500_middle.html
24. D. Sprott, Componentizing Enterprise Application Packages, in [4]
25. Sun Microsystems Ltd., The Java 2 Enterprise Edition Specification, 2000, http://java.sun.com/j2ee/download.html
26. C. Szyperski, Component Software Beyond Object-Oriented Programming, Addison-Wesley / ACM Press, 1998
27. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, Extensible Markup Language (XML) 1.0 (Second Edition), The World-Wide Web consortium, 2000
28. The UDDI organization, UDDI Executive White Paper, http://uddi.org/pubs/UDDI_Executive_White_Paper.PDF

29. J. Vowler, You cannot afford to ignore integration, ComputerWeekly, 1999
30. Bea Systems, Managing Complexity with Application Infrastructure, Whitepaper, http://contact2.bea.com/bea/www/wli/paper.jsp?PC=WLILG-WLP
31. E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: elements of reusable object-oriented software, Addison Wesley, 1995
32. J. Noack, H. Mehmanesh, H. Mehmaneche, A. Zendler, Architectures for Network Computing (in German), Wirtschaftsinformatik, Vol. 42, No. 1, 2000
33. Y. Genovese, B. Bond, B. Zrimseck, N. Frey, The transition to ERP-II: Meeting the Challenges, Gartner Group, R-14-0612, Sep. 2001
34. R. A. Kowalski, Algorithm = Logic + Control, Communications of the ACM, Number 22, Volume 7, pages 424-436, 1979