A Procedure Model for a SOA-Based Integration of Enterprise Systems

Anne Lämmer, sd&m AG, Germany
Sandy Eggert, University of Potsdam, Germany
Norbert Gronau, University of Potsdam, Germany

ABSTRACT

Enterprise systems are being transferred into a service-oriented architecture. In this article we present a procedure for the integration of enterprise systems. The procedure model starts with decomposition into Web services. This is followed by mapping redundant functions and assigning of the original source code to the Web services, which are orchestrated in the final step. Finally an example is given how to integrate an Enterprise Resource Planning System and an Enterprise Content Management System using the proposed procedure model.

Keywords: ECM system; ERP system; integration; self-diagnosis; service oriented architecture

INTRODUCTION

Enterprise resource planning systems (ERP systems) are enterprise information systems designed to support business processes. They partially or completely include functions such as order processing, purchasing, production scheduling, dispatching, financial accounting and controlling (Stahlknecht & Hasenkamp, 2002). ERP systems are the backbone of information management in many industrial and commercial enterprises and focus on the management of master and transaction data (Kalakota & Robinson, 2001). Besides ERP systems, enterprise content management systems (ECM systems) have also developed into companywide application systems over the last few years. ECM solutions focus on indexing all information within an enterprise (Müller, 2003). They cover the processes of enterprise-wide content collection, creation, editing, managing, dispensing and use, in order to improve enterprise and cooperation processes (Koop, Jäckel, & van Offern, 2001; Kutsch, 2005). In order to manage information independently, ECM combines technologies such as document management, digital archiving, content
management, workflow management and so forth. The use of ECM systems is constantly on the rise (Zöller, 2005). This leads to an increasing motivation for enterprises to integrate the ECM systems within the existing ERP systems, especially when considering growing international competition. The need for integration is also eminently based on economical aspects, such as the expense factor in system run time (Schönherr, 2005). For a cross-system improvement of business processes, enterprise systems have to be integrated.

RELATED WORK

Service Oriented Architecture as an Integration Approach

A number of integration approaches and concepts already exist. They can be differentiated by integration level (for example data, functions or process integration) and integration architecture (for example point-to-point, hub & spoke, SOA) (Schönherr, 2005). This article presents an approach to integrating enterprise systems by way of building up service-oriented architectures. This integration approach is of special interest and will be described in more detail.

The concept of service orientation is currently being intensively discussed. It can be differentiated from component orientation by its composition and index service (repository). Additionally, SOA is suitable for a process oriented, distributed integration (Schönherr, 2005). However, the addressed goals of component orientation and SOA are similar: different enterprise systems are connected through one interface, and a cross-system data transfer and the reusage of objects or components is enabled. Thereby a service represents a well-defined function which is generated in reaction to an electronic request (Burbeck, 2000). The SOA approach offers a relatively easy way to connect, add and exchange single services, which highly simplifies the integration of similar systems (e.g., enterprise take-over). Moreover, SOA offers a high degree of interoperability and modularity (Behrmann & Benz, 2005), which increases the adaptability of enterprise systems (Gronau et al., 2006).

The SOA approach is based on the concept of service. The sender wants to use a service and in doing so the sender wants to achieve a specific result. Thereby the sender is not interested in how the request is processed or which further requests are necessary. This is the idea of SOA, where services are defined in a specific language and referenced in a service index. Service request and data exchange occur via use of predefined protocols (Dostal, Jeckle, Melzer, & Zengler, 2005; Küster, 2003).

This service orientation can be used on different levels of architecture. The grid architecture is a common example of infrastructure level (Bermann, Fox, & Hey, 2003; Bry, Nagel, & Schroeder, 2004). On the application level an implementation usually takes place in terms of Web services.

The use of Web services offers the possibility of reusing raw source code, which is merely transferred to another environment (Sneed, 2006). The benefit of this transfer is the re-usage of perfected (old) algorithms. The main disadvantage is the necessity of revising the raw source code in order to find possible dependencies (Sneed, 2006). This is also true for enterprise systems. It is not efficient to reuse the entire old system, but rather only significant parts of it. To accomplish this it is necessary to deconstruct the old enterprise system and to locate the source code parts which can effectively be reused. Our approach uses self-diagnosis for finding these source code locations. This analysis will be considered in the third integration step.

Self-Diagnosis

As just described, our approach uses self-diagnosis for location of useful source code. For this, the method of self-diagnosis will be presented and the differences to other approaches will be shown.

Some approaches for transformation of legacy-systems into a SOA exist already. However, these approaches see the whole system as one service. The system gets a service
description for using this service in a SOA. Our approach differs in that it deconstructs the system for a tailored need. For this, the method of self-diagnosis is used.

Self-diagnosis can be defined as a system’s capacity to assign a specific diagnosis to a detected symptom. The detection of symptoms and assignment are performed by the system itself without any outside influence (Latif-Shabgahi, Bass, & Bennett, 1999). The mechanism of self-diagnosis has been detected surveying natural systems; it can partly be applied to artificial systems as well.

The first step of self-diagnosis is the detection of symptoms. Usually the detection of one existing symptom is not sufficient to make an indisputable diagnosis. In this case, more information and data have to be gathered. This can be described as symptom collection. In a second step the symptoms are assigned to a specific diagnosis. Depending on the diagnosis, corresponding measures can be taken (Horling, Benyo, & Lesser, 2001).

Symptoms are a very abstract part of self-diagnosis. These symptoms can be a high network load in distributed systems, missing signals, or buffer overload of the hardware layer. For enterprise systems the symptoms can be the frequency of usage of user interface elements by the user or dependencies of code parts or components. Other types of symptoms are possible. In general, the answer to questions concerning the measure of interesting items provides hints for possible symptoms.

Self-diagnosis can be categorized by the symptom acquisition method. Active and passive self-diagnosis must also be distinguished. In this context, the program or source code is the crucial factor for a division between active and passive self-diagnosis. A fundamental basis for either alternative is an observer or monitor.

Using passive self-diagnosis, the monitor detects and collects symptoms and information. It can either be activated automatically or manually (Gronau et al., 2006). If you know which items need to be observed and the point where this information can be gathered, you only have to monitor this point. This is what passive self-diagnosis does. For example: if you want to know how often a button is pressed, you have to find where the button-event is implemented in the code and observe this button-event.

In active self-diagnosis, the program’s function or modules are the active elements. They send defined information to the monitor and act independently if necessary. The monitor is used as receiver and interprets the gathered information and symptoms. The main advantage of active self-diagnosis is the possibility of detecting new symptoms, even if no clear diagnosis can be made before the problems become acute and are forwarded to other systems. In contrast, using passive self-diagnosis, the monitor can only inquire about specific data. In this case, a response or further examination is only possible if the problem is already known. For example: if you do not know the location of all the buttons and or the code component for the button-event, you will have to recognize all events with their initial point and filter them with the monitor. The monitor does not have to know how many buttons exist or where their code is located, but the buttons have to “know” to register with the monitor. These are the requirements of active self-diagnosis.

The assembly of diagnosis points depends on the application context and the software system. The required time and effort cannot be specified; it depends on the design and implementation of the software system.

Self-diagnosis can also be employed for the examination of source-code usage and interdependences. Depending on the desired information, different points of diagnosis have to be integrated into the source-code.

Different points of diagnosis have to be determined in order to allow for the allocation of code parts to various fields and functions. Therefore context, programming language, and software system architecture must be considered.

Our approach uses this method to locate code parts that can be collected into components. As we will demonstrate later in this article, we need to locate functions and enterprise systems business objects.
This method can be used for the detection of code parts which are possible services. Diagnosis points must thereby be integrated into the system source code, and software dependencies analyzed.

As we discussed earlier in this article, the main challenges in integration of legacy enterprise systems like ERP and ECM are, first, the deconstruction and second, the allocation of code. To address these challenges, we have developed a procedure model which will be described next.

**PROCEDURE MODEL**

In the following, a procedure model that integrates general application systems within a company is presented. The procedure model begins with the deconstruction of systems into Web services. This is followed by a mapping of redundant functions and the assignment of original source code to Web services, which is orchestrated in the last step.

The process includes taking the old ERP system; deconstructing it into different abstraction levels such as functional entities or business objects, searching for redundant entities, allocating the code fragments dependent on these functional entities and encapsulating them. This results in many independent functional entities, which can be described as a service. They have different abstraction levels and have to compose and orchestrate with, for example BPEL-WS. This composition and orchestration is the way of integration.

**Deconstruction of Systems**

First, the systems which are to be integrated are deconstructed into services. The challenge of this step depends on the number of particular services, which could span the range from one single service per system, up to a definition of every single method or function within a system as a service. In the case of a very broad definition, the advantages, such as easy maintenance and reuse and so forth, will be lost. In case of a very narrow definition, disadvantages concerning performance and orchestration develop; the configuration and interdependencies of the services become too complex.

This article proposes a hierarchical approach which describes services of different granular qualities on three hierarchical levels. *Areas of function* of a system are described as the first of these levels (Figure 1, Part 1). For example, an area of functions could include purchase or sales in the case of an ERP system or, in the case of ECM systems, archiving or content management. An area of function can be determined on the abstract level by posing questions about the general “assigned task” of the system. The differences between the three hierarchical levels can be discovered by answering the following questions:

1. **Question:** What are the tasks of the particular system? The answers resulting from this step correspond to services on the first level, which constitute the general task—for example sales, purchasing, inventory management or workflow management, archiving or content management. These tasks are abstract and describe main functionalities. They consist of many other functions which are the objects of the next level.

2. **Question:** Which functionality derives from every single task? The answers to this question correspond to the services on the second level that are contributed by the various functions. These functions are more detailed than the general tasks. They describe what the tasks consist of and what they do—for example, calculate the delivery time, identify a major customer, or constitute check-in and e-mail functionalities. For these functions the application needs data, which can found in the third level.

3. **Question:** Which business objects are utilized by both systems? This answer consists of the number of business objects that will be used as basic objects in both systems, for example article data, customer data or index data.
Figure 1. Procedure model for the integration of application systems

1. Task based decomposition of systems in three hierarchical levels

   Question 1: What are the tasks of the particular system?
   Result: The services on the first level which constitute the basic task.

   Question 2: Which functionality derives from every task?
   Result: The services on the second level which are contributed by the different functions

   Question 3: Which business objects are utilised by both systems?
   Result: Number of business objects which will be used as basic objects in both systems, e.g. article data, customer data or index data.

2. Preparation of the integration and mapping of redundant functions

   Question 1: Which tasks, functions and basic functions appear more than once?
   Result: List of possible redundant functions

   Question 2: Are they redundant, i.e. superfluous, or do they provide different services?
   Question 3: Can they be combined by an appropriate programming?

3. Detection and assignment of services to code fragments

   Step 1: Definition of concepts for the points of diagnosis depending on the systems and interesting information about the source code

   Step 2: Programming and integrating of the markers

   Step 3: Analysing the collected data

   Step 4: Reengineering of redundant services depending on the answer to question 3 of part 2

4. Orchestration of web services

   Step 1: Selection of a description language for web services (e.g. WS-BPEL)

   Step 2: Wrapping of the original source code into a web service

   Step 3: Modelling of the business process which is important for the integrating systems
In this procedure model, all possible levels of service deconstruction are addressed; yet the realization on all hierarchical levels constitutes an individual task.

The result of this step is a 3-stage model displaying the services of an application. The data-level, that is the integration of databases, is not further examined at this point since it is not an integral part of our model, the aim of which is to wrap functions as Web services without altering them or the original source code. The data level is not touched by this process.

**Preparation and Mapping**

The main advantage of Web service architecture is the high degree of possible reuse. By division into three hierarchical levels, a detection of similar functions is made possible, especially on the level of functionality and business objects. In some cases an adjustment of the functions is necessary in order to serve different contexts of use. Therefore, the next step consists of integration on different levels and the mapping of identical functions (Figure 1, Part 2). This step poses the following questions:

1. **Question:** Which tasks, functions and business objects appear more than once? For example, most applications contain search functions. Some applications have functions for check in and check out. ERP systems calculate the time for many things with the same algorithm under different names.

2. **Question:** Are these multiple functions and objects-redundant, that is superfluous, or do they provide different services? Some functions may have the same name but perform different tasks.

3. **Question:** Can these multiple functions and objects be combined by way of appropriate programming? For the functions ascertained in Question 2 to be similar functions with different names the possibility of integrating them into one has to be analyzed.

The advantage of this mapping is the detection of identical functions, which may by only named differently while completing the same task. In doing so, the benefit of reuse can be exploited to a high degree. Additionally, this part of the survey allows for a minimization of programming, due to the encapsulation of multiple functions. Only those functions which share a high number of similarities, but nevertheless complete different tasks, have to be programmed differently; they can be merged by reprogramming.

It is important to note that for this part the deconstruction consists of an abstract level and is in the functional view. In the following step, this will change from a functional view to a code view.

**Detection and Assignment of Services to Code Fragments**

The next step brings the biggest challenge, namely the transformation of existing applications into service oriented architecture. Until now, services have been identified by their tasks, but the correlation to existing source code still needs to be done. This is going to be accomplished in the next step (Figure 1, Part 3).

Self-diagnosis is used at this point to integrate earlier defined points of diagnosis into the source code. These points of diagnosis actively collect usage data and facilitate conclusions concerning the fields and functions via their structure. The structure of the points of diagnosis depends on the context of their application and on the software system. It is not possible to describe the complexity of the process, which also depends on the structure and programming of the software systems.

As we discussed earlier in Section 2.2, the points of diagnosis depend on what needs to be observed. Here we want to know which code fragments share correspondences and execute the identified functions in the functional view. From this follows the necessity of a monitor. For example, the points can be every method call in the source code of an ERP system. If the user calls a function, the points of diagnosis have to inform the monitor that they were called. The
monitor has to recognize and to analyze which method calls belong together.

Now the code fragments are analyzed and assigned to the functions identified in Part 1, and the wrapping of code fragments into Web services can be started. This step necessitates the usage of the existing source code and the description of relevant parts with a Web service description language, making possible the reuse of source code in service oriented architecture.

If redundant services have been detected in Part 2, which need to be reengineered, then the reengineering happens now.

**Orchestration of Web Services**

The results of stage 3 are the described Web services. These have to be connected with each other depending on the business process. This orchestration takes place in several steps (Figure 1, Part 4).

First, the context must be defined; second, the service description language has to be selected; and third, the Web services need to be combined.

A four-stage procedure model for a service-oriented integration of application systems has just been described. This process holds the advantages of a step-by-step transformation. The amount of time needed for this realization is considerably higher than in a “big bang” transformation, however, a “big bang” transformation holds a higher risk and therefore requires high-quality preparation measures. For this reason, a “big bang” transformation is dismissed in favor of a step-by-step transformation.

There is yet another important advantage in the integration or deconstruction of application systems into services when carried out in several steps. First, a basic structure is built (construction of a repository, etc.). Next, a granular decomposition into Web services occurs on the first level, thereby realizing a basic transformation of a service oriented concept. Following this, Web services of the second and third hierarchical level can be integrated step-by-step. This reduction into services provides high quality integration.

The procedure model we just presented is very abstract. Therefore, a practical example for two enterprise systems, ERP and ECM, will be given in Part 4.

**EXAMPLE OF APPLYING THE PROCEDURE MODEL**

It is necessary to develop a general usage approach and to test it on ERP and ECM systems since no concrete scenario of these technologies in regard to praxis as of yet exists (Issing, 2003). The aim of this example of use is to describe the integration of both company-wide systems, ERP and ECM, using our presented approach.

In what follows, we present a case study of a situation of integration of two systems: an ERP and an ECM system. A German manufacturer of engines and devices administers a complex IT landscape. This IT landscape includes, among others, two big enterprise systems. One of them is the ERP system “Microsoft Dynamics NAV” and the other is the ECM system “OS.5|ECM” of Optimal Systems. The ERP System includes modules such as purchasing, sales and inventory management. The ECM system consists of modules such as document management, archiving and workflow management. In the current situation a bidirectional interface between both systems exists. One example of a business process in which both systems are used is the processing of incoming mails and documents. In order to scan and save the incoming invoices of suppliers, the module of the ECM System “document management” is used. The access to the invoice is made possible through the ERP system.

In the future, a SOA-based integration of both enterprise systems can be reasonably expected under the aspect of business process improvement. Referring to the example mentioned above, the “portal management” component could be used to access, search, and check-in all incoming documents. What follows is a description, in four parts, of the integration based on the procedure model we presented in Part 3.
Part 1: Segmentation of ERP and ECM Systems into Services

According to the procedure model (Figure 1), the individual systems will be separated into independent software objects, which in each case complete specified functions or constitute business objects. The segmentation is structured in three bottom-up steps (Figure 2).

Identification is based on the answers to questions concerning main tasks of specific systems. The basic functions of an ERP system are purchasing, sales, master data management, inventory management and repository management. Document management, content management, records management, workflow management and portal management are basic functions of ECM systems. Subsequently, the areas of functions are disaggregated into separate tasks. Business objects are classified such as the business object “article” or “customer”. Thus, segmentation in areas of functions, tasks of functions and business objects is achieved and a basis for the reusage of services is created.

Part 2: Preparation of Integration/Mapping

The results of the first step of the segmentation are separation of services of differentiated granularity per system. According to the procedure model, the mapping on the different areas will be arranged in the second step. For that purpose, the potential services described will be examined for similarities. On every level of hierarchy, the functional descriptions (answers to questions in Part 1) of services are checked and compared with each other. If functions or tasks are similar, they will have to be checked for possibility of combination and be slotted for later reprogramming. One example of such similarity between functions is “create index terms”. Most enterprise systems include the function “create index terms” for documents such as invoices or new articles. The estimation of analogy of different functions, particularly in enterprise systems where implementation is different, lies in the expertise of the developer. Another example is the service “check in/check out”. This service is a basic function of both ERP and ECM systems and is now to be examined for possible redundancy. After determining

Figure 2. Segmentation of services
that the services “check in” or “check out” are equal, the service will be registered as a basic function only once. Services which are not equal but related will be checked in another step and either unified with suitable programming or, if possible, split into different services. The results of this step are the classification of services from ERP and ECM systems into similar areas and the separation of redundant services. The following table shows examples of separated services.

By this separation of both enterprise systems, a higher degree of re-usage and improved complexity handling of these systems is achieved. For the application of services, a service-oriented architecture (SOA) which defines the different roles of participants is now required (Burbeck, 2000).

Table 1. Examples of separate services of ERP and ECM systems

<table>
<thead>
<tr>
<th>Separate services</th>
<th>ERP</th>
<th>ECM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Functions</td>
<td>Purchase</td>
<td>Content management</td>
</tr>
<tr>
<td></td>
<td>Sales</td>
<td>Archiving</td>
</tr>
<tr>
<td></td>
<td>Article management</td>
<td>Document management</td>
</tr>
<tr>
<td></td>
<td>Repository management</td>
<td>Workflow management</td>
</tr>
<tr>
<td>Areas of Functions</td>
<td>Check in</td>
<td>E-mail connection</td>
</tr>
<tr>
<td></td>
<td>Identify delivery</td>
<td>Save document</td>
</tr>
<tr>
<td></td>
<td>Check out</td>
<td>Create index terms</td>
</tr>
</tbody>
</table>

Part 3: Detection and Assignment of Services to Code Fragments

As already described in the general introduction, the identification of functions to be segmented in the source code constitutes one of the biggest challenges in a transfer to service-oriented architecture. As part of this approach, the method of self-diagnosis is suggested. Appropriate points of diagnosis will be linked to the source code in order to draw conclusions from used functions to associated class, method or function in the original source code. Through the use of aspect-oriented programming, aspects can be programmed and linked to the classes and methods of the application system. Necessary data, such as the name of the accessed method, can be collected by accessing the respective classes and methods (Vanderperren, Suvée, Verheecke, Cibrán, & Jonckers, 2005).

Based on a defined service, “order transaction”, all the names of methods which are necessary for the execution of “order transaction” must be identified. To wrap the service “order transaction”, for example to combine it with a Web service description language, the original methods need be searched for and encapsulated. Additionally, the reprogramming of redundant functions is part of the phase of identification and isolation of services. This, as well, is only possible if the original methods are identified.

Part 4: Orchestration of Web Services

The last integration phase is used to compile Web services. The previous steps had to be completed in preparation for the procedure model. The Web services now are completely described and have a URI to be accessed. Now, only the composition and the chronology of requests of the specific Web services are missing. For the orchestration the Web service business process execution language (WS-BPEL) is recommended. The WS-BPEL was developed by the OASIS-Group and is currently in the process of standardization (Cover, 2005). If the Web
services present a function with a business process, the WS-BPEL is particularly suitable for orchestration of Web services (Lübke, Lüecke, Schneider, & Gómez, 2006). Essentially, BPEL is a language to compose (Leymann & Rolfer, 2000) new Web services from existing Web services with help of workflow technologies (Leymann, 2003). In BPEL, a process is defined which is started by a workflow system in order to start a business process.

Web services are addressed via a graphical representation with a modelling imagery of WS-BPEL. The business process is modelled independent from the original enterprise systems. Since in the first integration step, the systems were separated by their tasks and functions, now all of the functions are available for the business process as well.

CONCLUSION

The procedure model for the integration of application systems as it has been presented in this paper is an approach that has been successfully deployed in one case. Currently the assignment ability and the universality are being tested. The self-diagnosis, that is the assignment of source code to services via aspect oriented programming, constitutes a bigger challenge.

A verification of costs and benefits cannot be given sufficiently; however, several examples show convincing results and suggest a general transferability. The complexity in such a realization cannot be specified. Particularly for bigger and complex systems, the cost-to-benefit ratio has to be verified. Despite this, it must be recognized that the assignment of code fragments to functions is not an easy task. If one observes every method call a high number of calls must be analyzed. Visualization can be helpful for analyzing, since method calls belonging together will build a cluster in the emerging network. The observation of method calls is possibly not the most optimal way for very complex systems. If the functional view of services in Part 1 is not part of the business object layer, but only of the general task layer, one can reduce the numbers of diagnosis points.

The possibilities depend on the programming language and their constructs.

Finally, the approach presented above describes a procedure model for service-oriented integration of different application systems. The integration proceeds using Web services which thereby improve the integration ability, interoperability, flexibility and sustainability. The reusable Web services facilitate the extraction of several functions and combination of these into a new service. This allows for reuse of several software components.

Altogether, Web services improve the adaptability of software systems to the business processes and increase efficiency (Hofman, 2003). To give an example of the realization of the procedure model, an integration of an ERP- and ECM-system was chosen. The reasons for this choice consist in targeted improvement of business aspects and increasing complexity of both application systems. Dealing with this complexity makes integration necessary. Through mapping, redundant functions can be detected and as a consequence, a reduction of the complexity is made possible. Regarding the adaptability and flexibility of affect application systems, Web services are a suitable approach for integration. In particular, it is the reuse of services and an adaptable infrastructure which facilitate the integration.

In addition to all of this, we expect to discover additional advantages concerning maintenance and administration of affected application systems.

REFERENCES


Anne Lämmer (Dipl.-Inform.) is a software engineer at sd&m AG in Germany. Her research activities concentrate on the development of robust and adaptable software architectures. The theme of her doctoral thesis is in the field of self-organisation and self-diagnosis in enterprise systems.

Sandy Eggert (MSc Wi.-Inform.) is a research assistant at Potsdam University and scientific editor of the journal “ERP-Management”. Her research activities concentrate on the areas of adaptability of enterprise content management systems and the integration of enterprise systems. Before her job at Potsdam University she worked as a consultant at BearingPoint GmbH, focussing on enterprise information portals and knowledge management.

Norbert Gronau (Univ.-Prof. Dr.-Ing.) studied engineering and business administration at Berlin University of Technology, Germany. Currently he holds a chair of business information systems and electronic government at the University of Potsdam. His main research activities concentrate on the areas of Corporate Architectures and Business Management.