Modernization of a Legacy Application: Does it Have to be Hard?

A Practical Industry Cooperation Case Study

Arne Koschel, Carsten Kleiner
Faculty IV, Dept. for Computer Science
Applied University of Sciences and Arts
Hannover, Germany
{akoschel | ckleiner}@acm.org

Irina Astrova
Institute of Cybernetics
Tallinn University of Technology
Tallinn, Estonia
irina@cs.ioc.ee

***Abstract*—Modernization of a legacy application is not very hard any more. Whereas this may have been true a couple of years ago, this paper describes a case study, which shows that the modernization is significantly easier if modern integration tools, a service-oriented architecture and Web services are used. This is by contrast to a common belief that the modernization is always hard, regardless of the technologies used. The case study, where bachelor students succeeded to carry out the modernization of a legacy application, shakes that belief. The students neither had previous experience with the technologies used in the legacy application nor with the ones used for the modernization. As major contributions this paper provides an overview of approaches to modernization, a full case study for the modernization (including details on business process analysis, architecture, and tools), and comprehensive 'lessons learned' to help for 'the practice'.**

Keywords—service-oriented architecture; mainframe; legacy integration; experience report; Web service

I. INTRODUCTION & MOTIVATION

Declared 'dead' for quite a while now, many legacy mainframe applications are still happily productive and will continue to be. Indeed, until today legacy applications that are based on mainframe database management systems (DBMSs) like Adabas and associated fourth generation programming languages (4GL) such as Natural, are still often in practical use. However, those applications are often only badly, if at all, integrated with newer enterprise applications. The integration of legacy application assets is required, e.g., due to joint use of functionality or data. It is an important task, which occurs frequently in industrial practice.

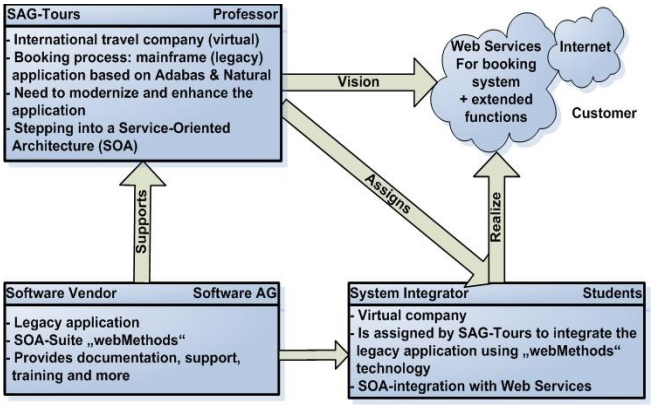

Figure 1. Overview of the SAG-Tours project

Initially, a punctual integration of the legacy assets was achieved by means of 'traditional' enterprise application integration (EAI) technology (cf. [2][6]). Nowadays a service-oriented architecture (SOA) [7][13] proposes a promising solution to this task.

This paper describes a case study for legacy modernization based on integration technology and Web services. Used in conjunction, they served as the base for integration of an existing legacy mainframe application (SAG-Tours) into an up-to-date distributed SOA.

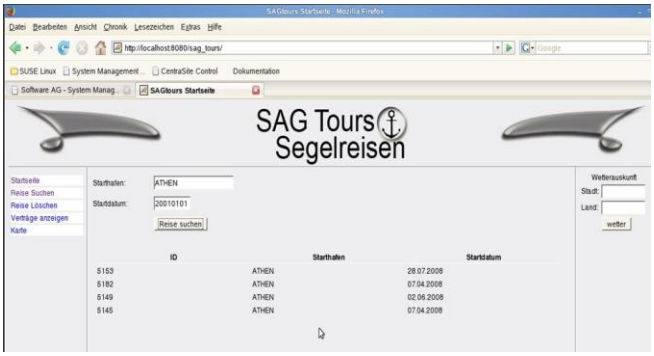

Figure 2. User interface of SAG-Tours application after modernization

The case study was done in scope of the SAG-Tours project (see Figure 1). This project involved research and industry cooperation with a German software company, Software AG (SAG). The goal of the SAG-Tours project was

2010, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

to integrate the SAG-Tours mainframe application into a modern Web environment – a requirement being driven by Software AG customers. The customers wanted the SAG-Tours application to become Internet-ready quickly (within a year), thus giving end users the possibility to access the application via a Web browser (see Figure 2). Previously, 'green screens' were the only way to access the application (see Figure 3). The SAG-Tours project team consisted of 10 final-year bachelor students supervised by 2 professors. The students had an average working effort of 1 day per week per person. The team was given some 'getting started' and 'configuration hotline' help from Software AG.

This paper provides two major contributions. First, it shows how such a technically complex integration task (where both old existing systems and several integration technologies are involved) can be undertaken. Second, it shows that this task could be carried out even with relatively inexperienced students under only moderate supervision of professors. This means the integration of at least functional-wise not too complex legacy applications into a SOA should not be a too difficult work any longer.

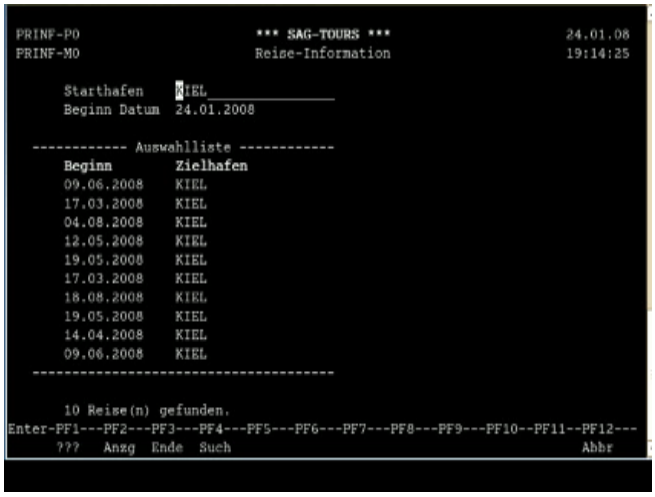

Figure 3. User interface of SAG-Tours application before modernization

This paper is an in-depth extension of our previous work [1]. The related work has been extended by providing several new references to related academic and industrial publications. We examine different possible approaches to modernization of a legacy application. Also, we explain details on our approach to modernization of the SAG-Tours application and our architecture that supports integration of the application into a SOA. Finally, we bring together all the lessons learned during the SAG-Tours project that can be useful for application in other similar legacy modernization projects.

The rest of the paper is organized as follows. Section II describes the related work. Section III describes the SAG-Tours application and its background technologies (viz., Natural and Adabas). Section IV analyzes existing approaches to modernization of a legacy application (viz. packaged applications, code conversion, re-hosting, re-architecting, and SOA enablement). Section V gives details on our approach to modernization of the SAG-Tours application (viz. SOA enablement). Section VI gives details on our integration architecture. Section VII describes lessons we learned during the SAG-Tours project; it is followed by conclusion and outlook to possible future work.

II. STATE OF THE ART

Related work especially regarding integration tools can be found in industry as well (see, e.g., SAP [19], IBM [20], Oracle [21][22], Software AG [23] and Microsoft [24]). However, because Software AG's integration technology stack was given us as a pre-requisite, we focus here on academic related work only.

Although Canfora et al. [17] use a wrapper approach as we do, they focus mainly on interactive functionality.

Englet [5] proposes a bottom-up integration approach, which is not restricted to interactive components. It is, however, suboptimal with respect to process modeling because it might not take process optimization into account.

Smith [12] discusses several ways to introduce a SOA into an enterprise, including legacy assets, but on a general level. Erradi et al. [4] discuss similar strategies. Instead of a general discussion, we focus mainly on concrete technical integration aspects.

Lewis et al. [9][10][15] develop a service-oriented migration and reuse technique (SMART) to assist organizations in analyzing legacy components in order to determine if they can be reasonably exposed as services in a SOA. SMART provides a preliminary analysis of viability of different migration strategies and the associated costs, risks and confidence ranges for each strategy. In particular, SMART gathers information about legacy components and produces the best migration strategy for a given organization. Thus, SMART helps organizations to select the right migration strategy. SMART can be used to analyze what legacy functionality can be re-used in a SOA. However, we do not need this analysis, because it was pre-defined for us, which legacy functionality had to be re-used. We consider this to be an everyday situation in practice.

Erl [14] introduces a pattern-oriented background for a SOA. While being helpful in general, more detailed work is required for a concrete integration task.

Sneed [18] proposes a salvaging and wrapping approach (SWA). This is a three-step procedure for creating Web services from a legacy application code. These steps are: (1) salvaging the legacy code; (2) wrapping the legacy code; and (3) exposing the legacy code as a Web service. SWA is effective in process and service integration. But it provides limited support for content integration by wrapping second-level Web services. This is similar to the second step of our approach.

Ziemann et al. [25] describe a business-driven legacy-to-SOA migration approach called enterprise model-driven migration approach (EMDMA). This approach is based on enterprise modeling, by introducing an elementary process model between the business function tree and the tree related to the legacy application, which is then aligned to the function tree of the legacy application. Finally, it applies a transformation from the legacy business process model to the

2010, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

SOA process model. EMDMA draws attention to the fact that aspects such as functional granularity, security, reliability, scalability, etc. are not taken into account sufficiently during the migration. Thus, there is a need for investigating how these aspects of the legacy application can be mapped to a SOA.

III. SAG-TOURS APPLICATION

SAG-Tours [11] is a legacy mainframe application written in Software AG's Natural. It is based on a 1-tier terminal mainframe architecture (see Figure 4). Functional wise one has the possibility to order fictitious cruises.

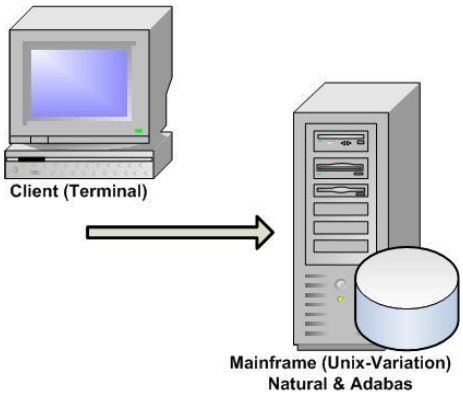

Figure 4. 1-tier system architecture of SAG-Tours application

Technically terminal emulations are used, which communicate with a Unix variation of Natural via telnet protocol. The SAG-Tours application itself has a connection to an Adabas database [3]. Adabas is a high performance mainframe DBMS, which is internally based on so-called inverted lists. An example for a Natural query would be as follows:

```
FIND CRUISE
    WITH START HARBOR= 'CURACAO'
```

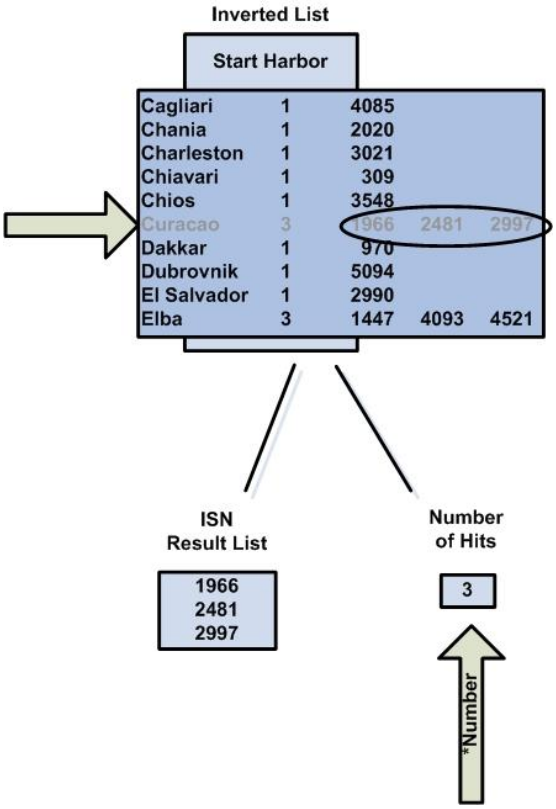

Figure 5. Internal structure of an Adabas example query using inverted lists

This query delivers all journeys with the starting harbor 'CURACAO'. Figure 5 shows '3' as the total hits number of query results. It also shows the resulting internal sequence numbers (ISNs). These ISNs can be interpreted as logical pointers to the relevant resulting tuples.

IV. APPROACHES TO MODERNIZATION OF LEGACY APLLICATION

Legacy modernization is the process to supplement or replace an organization's legacy applications and technologies using newer applications and technologies that are based on open standards, while retaining business logic [28].

There are five basic approaches to legacy modernization (that can be used alone or combined):

- Replacing legacy applications with packaged applications.
- Re-architecting legacy applications.
- Legacy application code conversion (also called automated migration of legacy applications [28]).
- Re-hosting legacy applications.
- Enabling SOA (also called enabling Web [29], re-interfacing [29] or business logic wrapping [26]).

There are advantages and disadvantages with all these approaches. An advantage of one approach is usually a disadvantage of another and vice versa. But "organizations that are SOA-enabling their legacy applications on the legacy platforms are outperforming those that are using any other

2010, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

approach. They report better productivity, higher agility, and lower costs for legacy modernization projects." [29].

Since legacy applications are typically not architected with a SOA and Web services in mind, careful selection of an approach is required before modernizing a legacy application. Depending on the approach selected, the legacy application may require big, small or no change at all. Therefore, it was important to choose the right approach for the SAG-Tours application.

A. Packaged Applications

This approach [28] consists of replacing the legacy application with a packaged (commercial off-the-shelf) one made up of SOA components. These components can then be combined with re-architected components, re-hosted components and automatically migrated components using a SOA orchestration engine.

Because packaged applications are seen as sets of reusable components, the biggest advantage of this approach is the increased agility. However, this approach does not work in a situation where legacy applications have unique functionality that cannot be replicated by packaged applications. In this case, the business logic can be retained from the legacy application using one of other approaches such as re-architecting or re-hosting. Therefore, we rejected this approach.

B. Code Conversion

This approach [27][28] consists of converting the legacy application code into a new programming language (e.g., Java). It is often used in combination with replacing the legacy application with a packaged one.

The biggest advantage of code conversion is that the process of migrating, e.g., from Natural to Java can be automated; i.e., it can be carried out by a machine and require no human intelligence during the migration process.

Because a machine will carry out the migration process, it can be done quickly and consistently. But it only works if the gap between the legacy architecture and the new one is relatively small. E.g., it is not possible to convert the procedural design of Natural into the object-oriented design of Java. The fundamental design concepts of Java – e.g., a class and its behavior – are architectural concepts that require human intelligence to design. The designer of a truly object-oriented application will use these concepts in new ways that cannot be recovered from a legacy application designed using procedural techniques. Therefore, although the automatic migration of Natural code into Java can be done, but the resulting Java code will not be the same Java code that would be designed for a truly object-oriented application.

The biggest disadvantage of code conversion is that it is much more invasive to the legacy application than other approaches to legacy modernization such as re-hosting and SOA enablement. Therefore, we rejected this approach also.

C. Re-hosting

This approach [28] consists of migrating the legacy application to a lower cost platform. It can be used in combination with code conversion. E.g., during the re-hosting process, the legacy database calls to a mainframe database such as Adabas can be eliminated.

The biggest advantage of re-hosting is that it is non-invasive to the legacy application because the application is left "as-is".

Since re-hosting does not change the legacy application, one disadvantage of this approach is that it forces a continued reliance on legacy skills. Another disadvantage is that re-hosting retains much of the legacy architecture. This means that the implementation of Web services could be cumbersome. Therefore, we rejected this approach also.

D. Re-architecting

This approach [28] consists of extracting business logic from the legacy application, building a new application, integrating this new application with the legacy one, and finally, shutting down the legacy application.

The biggest advantage of re-architecting is that it maximizes the benefits of a SOA and new technologies. But it is the most expensive approach to legacy modernization – a legacy modernization project can span many years. Therefore, we rejected this approach also.

E. SOA Enablement

This approach [26][28] consists of wrapping business processes and presenting them as Web services to an enterprise service bus (ESB). This is the approach we selected.

The biggest advantage of SOA enablement is that it provides immediate integration of the legacy application into a SOA. In addition, this approach is relatively non-invasive to the legacy application. Therefore, legacy components can be used as part of a SOA with no or little risk of destabilizing the legacy application.

However, like re-hosting, SOA enablement forces a continued reliance on legacy skills. Another disadvantage of this approach is the need for communicating among disparate environments because the legacy components continue to reside on the legacy platform. However, using SOA enablement combined with re-hosting can eliminate the need for such communication because all the components – the re-hosted components that have been integrated into a SOA, the new components, the packaged application components, and the SOA orchestration engine that brings them all together – reside on the same new platform. This also makes it easier to convert the legacy components into a new programming language such as Java.

V. Conceptual Integration Approach and Business Process Model

In this section we will show how the domain specific business process model for the case study has been developed and present some conceptually important aspects of the result. We will also describe the conceptual integration approach we have chosen and the advantages of this approach.

2010, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

A. Conceptual Integration Approach

As explained above, for integrating the SAG-Tours application into a SOA, we decided to follow SOA enablement [26], which continues to use the application itself. We selected SOA enablement primarily because rather quickly (within a year) this approach can bring the application into a modern Web environment, where the application can be accessed via a Web browser. This is one of the biggest advantages SOA enablement has over other approaches to legacy modernization.

Our cooperation partner, Software AG, also offers tools that use screen scraping to extract complete work processes from a legacy application and make them available as Web services (see Figure 6). We did not use these tools either. On the one hand, we did that to stay technology-neutral as far as possible. On the other hand, these tools typically yield only a few rather coarse-grained services in the business process layer (cf. [7]) and no services in the underlying layers of business services and basic services. By doing so, increased speed in an integration of the legacy application can be reached; but the increased flexibility of process definitions by variable combination of services of the underlying layers, which is one of the main goals of introducing a SOA, may not be achieved. We will show below how the services of the business service layer have been variably composed into business processes in the case study.

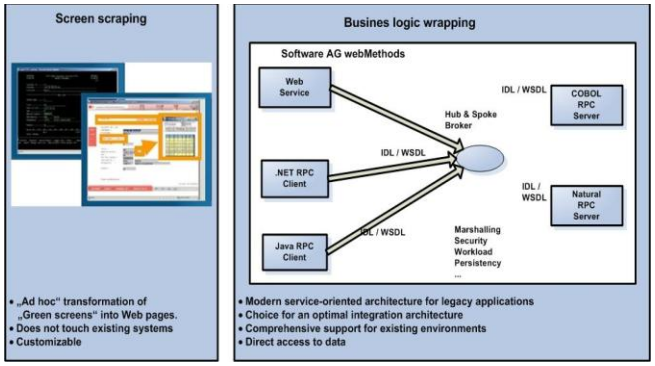

Figure 6. SOA enablement via screen scraping vs. Comprehensive mainframe integration based on services via business logic wrapping

Encapsulation of functionality from the SAG-Tours application in business services is shown in Figure 7. It shows the detailed flow of the business process for deleting a cruise, which is the result of the three-step integration approach (see Section V). It is very easy to identify the elementary tasks (such as finding and deleting of single business objects), which had been implemented as Natural programs in the SAG-Tours application. These programs will now be wrapped as business services so that the whole flow of the business process may be described as a composition of such business services.

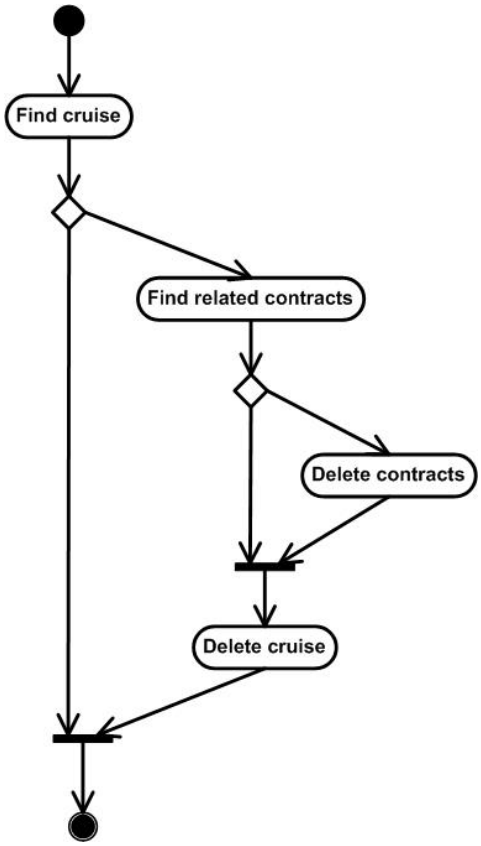

Figure 7. Sample business process 'Delete journey' modeled as activity diagram showing the integration of several existing business services

In the case study, the modeled business processes were translated manually into executable code on the ESB since there were only rather few processes. However, in the real-world scenarios (even in medium-size legacy modernization projects) an automated generation tool fitting for a particular technology required by the ESB should be used.

B. Business Process Model

In the case study, we started to set up a business process model of the domain. Since we chose a combination of top-down and bottom-up approaches to integrate the SAG-Tours application into a SOA, at first the optimal target processes had to be identified (top-down part). Since there have been quite a large number of processes, in order to check that the approach would also hold for larger legacy modernization projects, we grouped the identified processes into packages. The packages have to be formed based on domain-specific criteria; in the case study packages could be formed based on the domain entities, on which the processes primarily operated. Figure 8 shows the package model of processes. After identifying all processes and assigning them to packages, it was also possible to define dependencies between packages based on the underlying dependencies of the processes. This yields another helpful structuring of the whole set of processes and is also shown in Figure 8.

2010, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

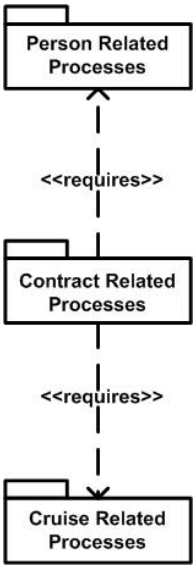

Figure 8. Packages for business process services and business services for structuring of the process model domain

After identifying the target processes, we identified the functionality of the SAG-Tours application that would be required in the processes and thus had to be modeled as business services (bottom-up part) in order to compose the processes in the final step. Identifying the legacy functionality and assigning it to business services had been a rather easy task. Most of the functional components could be directly derived from the existing Natural programs of the SAG-Tours application. However, some of the functional components had to be implemented anew in Natural (based on the Adabas database) in order to achieve a technologically homogeneous implementation of the foundation. For example, there was a Natural program DRINF-N0, which computed and returned all available cruises for a given start date and start harbor. This was encapsulated within the business service **BS_FindCruise** and could subsequently be used in different business processes.

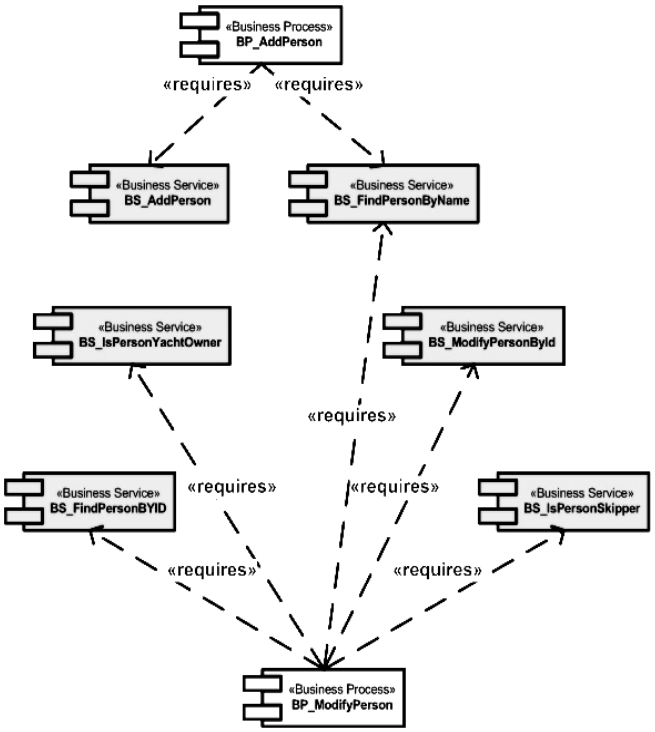

Figure 9. Business process model for person related processes and services

In the third and final step, the business services obtained in the second step could be composed to detail the business processes defined in the first step. For example, Figure 9 shows a part of the obtained business process model, which contains the processes and services related to person entities. The business process **BP_AddPerson** to add a new person in any role to the SAG-Tours application is, e.g., composed of the services **BS_AddPerson** and **BS_FindPersonByName**, which directly correspond to the functional components of the application.

As expected, most of the identified business services could be used in several different business processes. This can already be concluded from Figures 9 and 10, even though only part of the corresponding package models are shown there. For example, the business process **BP_ModifyPerson** to change information about an existing person in the Adabas database is composed of the services to find and modify a person by his or her ID or name as well as the services to obtain further information about the roles of the person.

2010, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

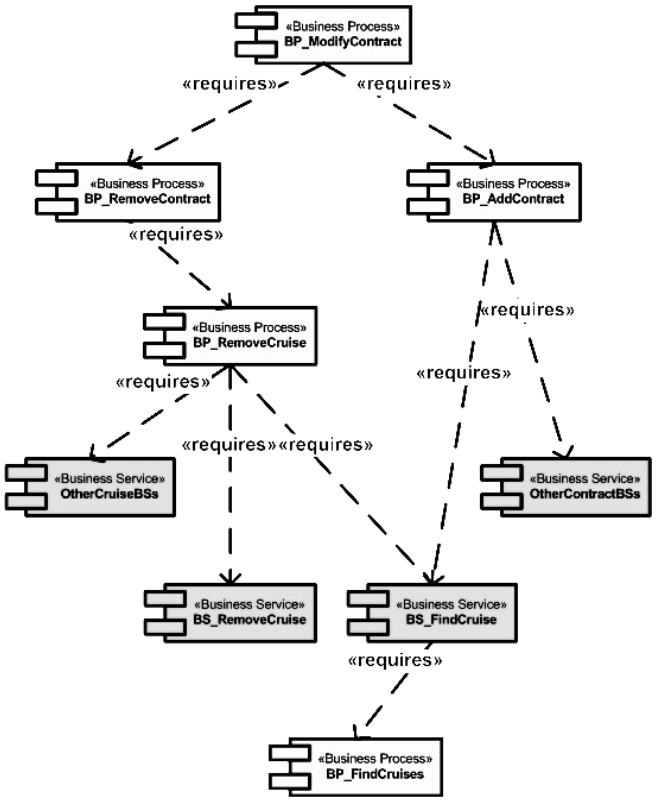

Figure 10. (Part of) the business process model for cruise and contract related processes and services

It should be noted that reuse is possible not only on the level of business services. As shown in Figure 10, there is also potential for reusing complete business processes within other business processes. For example, the business process BP_ModifyContract, which is executed to change any aspect of a booking by a person for a specific cruise, can be composed of the business processes to add and remove contracts, which in turn are composed of the business process to remove a cruise among others. Consequently, we can see that reuse on both levels is greatly simplified by the integration approach. Of course, business services may not be composed of business processes internally due to the definition of the according layers.

VI. INTEGRATION ARCHITECTURE AND STEPS

In this section, we will describe technical steps we have taken to bring up an integration architecture for the SAG-Tours application. At first, we will describe a general integration architecture, which is then mapped to a concrete integration tool suite. This is followed by an overview and more detailed technical steps one has to follow to use this integration architecture.

A. Integration Architecture

Since in the case study we followed the comprehensive mainframe integration approach (see Figure 6) that fits into a SOA as well, an N-tier integration architecture [2][7][13] was the appropriate means of choice. In this general integration architecture (see Figure 11), an end user uses a Web browser, which acts as a client front-end to a Web server. The Web server hosts presentation preparation logic, which (in a 'traditional' Web technology environment) prepares HTML pages for the end user's GUI and uses HTTP to interact with the end user's Web browser. On the other side, this logic accepts a service access protocol (e.g., SOAP over HTTP in the case of Web services) to access integrated services. In this case, an encapsulated DBMS is accessed, again, by means of some service access protocol, say, an ordinary remote procedure call (RPC).

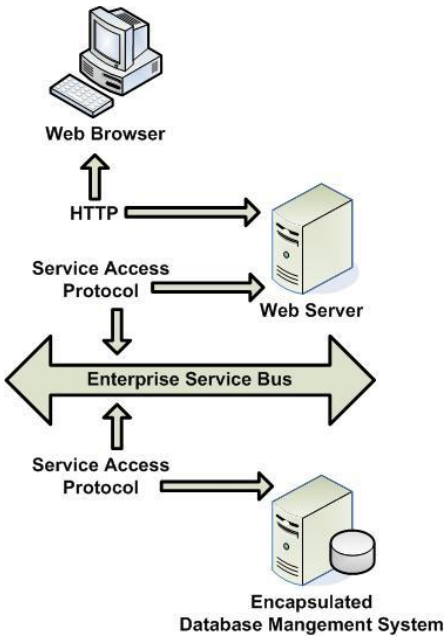

Figure 11. General N-tier system architecture using Web technology and an enterprise service bus

Since the integration technology stack was pre-defined for the SAG-Tours project, integration architecture for the SAG-Tours application was based on Software AG's integration tools such as EntireX Broker (see Figure 12).

At the lowest level of the integration architecture, there is a persistence layer with an Adabas DBMS. Above it, there is an application layer with a Natural runtime engine, a Natural RPC server (which calls that engine), a Software AG's EntireX Broker (which acts as an ESB that 'understands' different service access protocols) and – optionally – a so called integration server (which actually is an execution engine for a specialized business process execution language). At the highest level, there is a server-side Web presentation layer (also called GUI layer). Here the Apache Web server and the Servlet engine Tomcat are used. Like in the general integration architecture in Figure 11, a typical Web browser is used for the end user client access.

The integration can take place in three layers:

- Persistence layer. Here calls to the legacy database (e.g., Adabas) are replaced with Web services that issue the same native calls and return the requested data. These calls may be further modernized by

2010, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

allowing SQL calls to be issued instead, even though data is still stored in the legacy database.

- Application layer. Here calls to the legacy procedures and programs (e.g., Natural subprograms and programs, respectively) are replaced with Web services that issue the same calls. The legacy procedures and programs that are called by the legacy application may have been written as reusable components and are candidates for reusable services.
- Presentation layer. Here 'green screens' are replaced with Web services that drive the legacy application the same way the original screens did. 'Green screens' are good candidates for the integration because many legacy applications use them to drive a single transaction; e.g., deleting a journey, adding a person / contract, etc. The integration in this layer often involves screen scrapping.

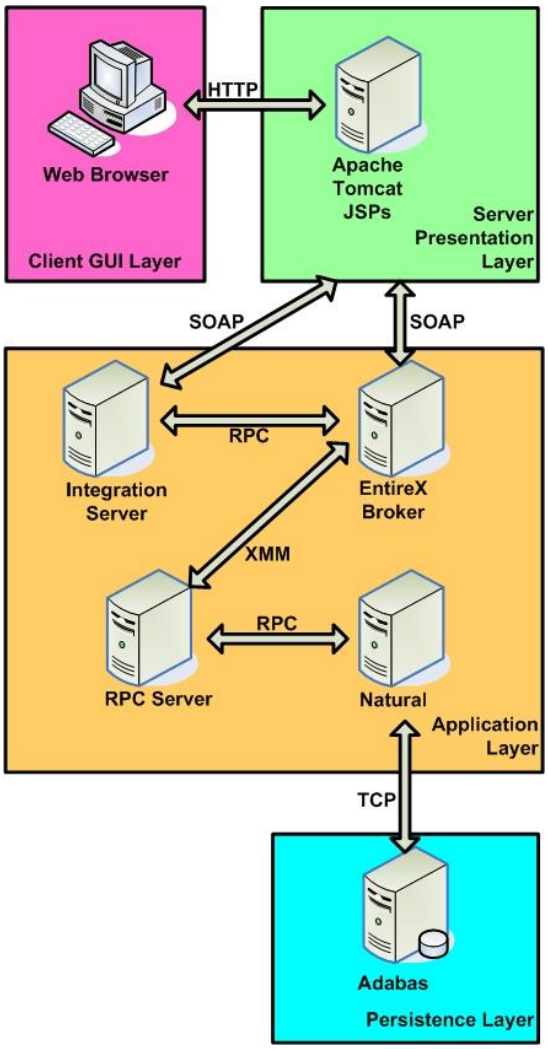

Figure 12. 3-tier system architecture that supports integration of SAG-Tours application into SOA and uses Software AG's integration tools

As shown in Figure 12, in the case study the integration took place in the application layer. This was feasible because the source code of the SAG-Tours application was available for us.

B. *Integration Steps*

Based on the components in Figure 12, the following technical integration steps have to take place:

1. Import Adabas database structures into a repository.
2. Map / import Adabas database structures for Natural subprograms.
3. Re-use existing Natural subprograms if possible. Otherwise, write suitable new ones based on the business process analysis from the previous section.
4. Define Natural subprograms to be accessible via the Natural RPC server. This server in turn is called by the EntireX ESB (also known as EntireX Broker).
5. Generate Web services stubs (here Java-based) for imported subprograms, thus exporting those stubs as Web service definitions from EntireX Broker.
6. Access those Web services from Java programs using JavaServer pages, e.g., via Axis / JAX-RPC.
7. Send the results from the JavaServer pages to the end user's Web browser.

The components as well as their usage within those integration steps are described below in more detail.

C. *Steps 1-2: Accessing Adabas from Natural via RPC*

Following the above steps, initially the existing Adabas database needs to be accessed by Natural programs. For existing Natural programs (which are re-used directly), there is no extra work just because the SAG-Tours application does this already. For new or re-written Natural programs, however, the existing Adabas database structures, which they want to access, need to be imported into a repository from the Natural tool suite.

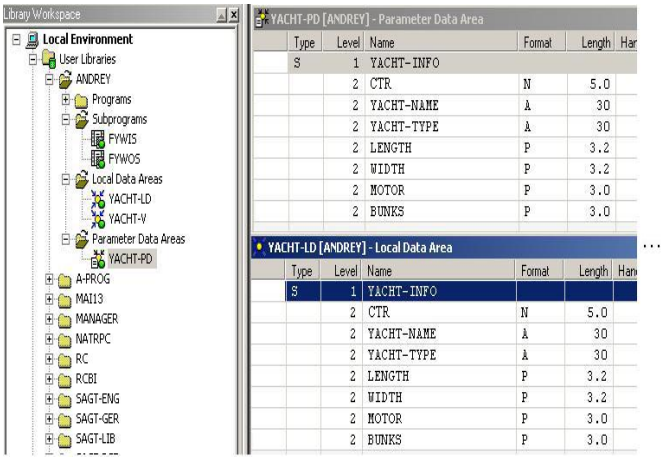

Figure 13. Data structure definitions

Natural programs and subprograms can query the repository to define their database access data structure. Such data structures are so called DEFINE DATA areas in Natural. Within such data areas, local Natural program variables are defined. Moreover, views to an underlying

2010, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

database are defined by means of the USING clause within a DEFINE DATA area definition. Such a view serves as a common storage area between the Adabas DBMS and the calling Natural program. (It should be noted that Natural is not restricted to Adabas. Rather, it can also be used with other DBMSs like DB/2 and Oracle.)

Figure 13 shows a screen shot of the Adabas data mapping development environment from the Natural tool suite. As can be seen, Natural data types are defined for local and parameter data areas. These data areas have two purposes. First, they are required for the Natural subprogram itself as the communication structure with Adabas. This is done by the LOCAL DATA area (YACHT-V and YACHT-LD in Figure 13). Second, they specify the output data from the Natural subprogram in the PARAMETER DATA area (YACHT-INFO within YACHT-PD). This is then used as the mapping input by the EntireX Broker (see Section D below for more details).

The following little excerpt shows some Natural subprogram code, which fills a data area called 'YACHT-VIEW' using a FIND statement that searches for 'HUGE' yachts. This statement accesses the Adabas database from the SAG-Tours application.

```
DEFINE DATA
PARAMETER USING YACHT-PD
LOCAL USING YACHT-V
      1 COUNTER (N5.0) INIT <0> 1
      1 LIMIT (N5.0) INIT <0>
END-DEFINE

FIND ALL YACHT-VIEW WITH
      YACHT-VIEW.YACHT-TYPE = 'HUGE'
      . . .
      [code to check and increment COUNTER
      and to expand the size of the output
      data structure YACHT-INFO if required]
      . . .
      MOVE YACHT-VIEW.YACHT-NAME TO
      YACHT-INFO.YACHT-NAME(COUNTER)
      . . .
END-FIND

MOVE COUNTER TO YACHT-INFO.COUNTER
END
```

The query result in the 'YACHT-VIEW' data area is then moved to the output PARAMETER DATA 'YACHT-PD / YACHT-INFO' area. From this area, a 'YACHT-INFO' data structure in Figure 13 is filled. The 'YACHT-INFO.COUNTER' variable is filled with the number of tuples, which were returned from the FIND statement. Eventually, the newly filled YACHT-INFO data structure (conceptually similar to a 2-dimensional array, technically several 1-dimensional arrays) is returned as the Natural subprograms result.

D. Steps 3-4: Accessing Natural via RPC

Following the above steps, the Adabas database can be accessed by means of Natural programs. This is pretty much the way, the SAG-Tours application works. Now in order for this application to be re-used as services within the application modernization context of the SAG-Tours project, those programs need to become technically accessible from the outside. For this purpose, the (Natural) remote procedure call (RPC server in Figure 12 is used. Natural subprograms (which run on remote machines) are accessed via a RPC. This is conceptually comparable to protocols such as Java Database Connectivity (JDBC), which are frequently used for Java-based remote access to relational databases.

Within the integration architecture in Figure 12, the EntireX Broker actually uses a RPC for such remote Natural access. The EntireX Broker is Software AG's integration turn table, which thus conceptually serves as the core of an ESB as it is known from a SOA-based integration architecture (see [7][13] for more detail).

Two major pre-requisites for this approach exist. First, he Natural programs need to be callable as subprograms. This just requires a well-written Natural subprogram with clearly defined input / output data areas. Another option is to have Natural programs as a base, which can reasonably easy be modified to fulfill this requirement. However, one of those options is due to Natural coding practices not to seldom given for existing Natural code and it does hold for the SAG-Tours application.

Second, the semantic structure of the existing subprograms must be 'good enough' to be re-usable in a modernized business context. To ensure this, we did the business process analysis of the existing Natural programs as described in the previous section.

Since most of the existing programs were easily understandable, e.g., comparable to functions like 'DELETE JOURNEY' or 'FIND-AVAILABLE-YACHTS', this was a manageable task for us (see [16] for more details). However, such an analysis might not be an easy task for more complex existing Natural code.

It should be noted that as a 'side effect', all the students were able to read and write Natural code afterwards (even though they had no previous experience in Natural coding).

E. Step 5: Re-using integrated legacy code as Web services

Having integrated the Natural subprograms using EntireX Broker, one now wants to re-use them within non-Natural contexts. In the case of the SAG-Tours project, Web services are the means of choice. Thus, a Web Services Description Language (WSDL) based service interface definition and SOAP access to the integrated Natural 'services' needs to be enabled.

Easily enough, the EntireX Broker development environment can generate all the required code. It utilizes the PARAMETER DATA-areas form the above steps to enable a mapping specification from the Natural procedure parameters to XML data types. Of course, this requires a suitable PARAMETER DATA areas for each Natural

2010, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

subprogram (either newly written or existing), which is to be mapped to another service description.

The mapping itself is based on a XML mapping specification (XMM), which is used by EntireX Broker at runtime for data type conversions (see XMM in Figure 12).

Now that the task of technically integrating the existing Natural subprograms is achieved utilizing the EntireX Broker, those programs can directly be accessed as Web services based on SOAP. EntireX Broker does the internal mapping and provides a WSDL based service endpoint for each of the exposed 'Natural'-Web services.

F. Step 6-7: Using Web services from JSP clients

As shown in Figure 12, we then just used Java code within JavaServer Pages (JSPs), which in turn called the Web services above.

Instead of hard-coding JSPs and Java code, the Web services could also be called using Software AG's business process engine. This component in Figure 12 is called integration server. We tried this exemplary as well – it worked fine except for some minor data type issues. But we did not explore it in more depth due to given time limits – the SAG-Tours project was limited to one year.

Eventually the JSPs allowed for an easily developed GUI front-end for the end user. In the real-world scenario, there is also the possibility to call the Web services from other external programs. Since our legacy components are completely enclosed as Web services, they can easily be embedded in a larger SOA environment.

VII. LESSONS LEARNED

Looking back at the SAG-Tours project, we can derive a good bit of experiences, which might also be valuable for other legacy modernization projects. These experiences are described below as learned lessons (both technical and pedagogical).

A. Technical Lessons

From a technical point of view, the most interesting insight gained by the SAG-Tours project has been related to mainframe legacy software. In particular, it was interesting to see how easy or difficult it was to integrate a legacy mainframe application into modern software architecture and how object-oriented programmers could cope with this task. (The students involved in the SAG-Tours project were reasonably experienced in Java coding.)

1) Complexity of integration:

- *Integration effort*: The integration of existing legacy mainframe applications into a SOA is not too hard any more. Whereas this may have been true a couple of years ago, the SAG-Tours project showed that the integration is rather simple. This has been proven in the case study where final-year bachelor students succeeded to carry out the project. The students had no previous experience in mainframes and the technology used in the SAG-Tours application.
- *Tool support*: The integration of Adabas / Natural legacy applications is very well supported by Software AG's integration tools. Since Software AG's integration technology stack was given us as a pre-requisite, a general conclusion on tool support cannot be drawn. It will be interesting to evaluate this aspect in follow-up projects; i.e., whether integration tools provided by different vendors can also be used to implement our integration approach.
- *Effort dependencies*: As expected, the exact effort required depends heavily on the size and number of the target processes and (even more) on the amount of knowledge of and documentation on the existing code in the legacy application itself.

2) Integration Approach:

- Regarding the conceptual integration approach to the legacy modernization, we used a combination of top-down and bottom-up approaches. However, this may not be viable in all situations. Factors that have to be taken into consideration in order to choose the right approach include:
  - *Quality of Service (QoS)*: the screen scraping approach can never yield better QoS than the original application at best. This time, the comprehensive mainframe integration approach might open up possibilities for improving the QoS externally.
  - *Time to market requirements*: if it is important to have the legacy application usable in a service environment as fast as possible, regardless of the technology used, the bottom-up approach will be a better choice.
  - *Effort (time and money)*: In most cases, comprehensive mainframe integration (which is based on services) will be more costly from a short project point of view because of the conceptual complexity. This effort could, on the other hand, well pay off in the long run because such integration has the potential to increase re-using of components and may simplify software maintenance.
  - *Knowledge about existing code*: if there is only a black-box like knowledge of what the existing application components do and not how they do it in detail, then the comprehensive mainframe integration approach may not be feasible at all. This is especially true if existing components have to be modified or extended for the integration (as in the case study). This scenario which seems unrealistic at first sight can actually be found in many organizations nowadays.
- In summary, we feel that our combined integration approach is ideal for many situations because it joins the long-term potential of the top-down approach with the technological ease of the bottom-up approach.

2010, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

B. Pedagogical Lessons

From a pedagogical point of view, there are several lessons learned as well, which are interesting from higher computer science education perspective. Within the SAG-Tours project the following conclusions were derived:

1) Usage of complex technologies in bachelor projects

- *Possible project type*: A technology environment can involve different technical skills such as different programming languages, several tools, and different hardware and networking technologies, as well as conceptual knowledge in software / system architectures, project management techniques, etc. The usage of such a complex environment is still feasible, although certainly not an easy project for students.
- *Enhanced motivation*: Not only was the SAG-Tours project feasible, but also the students were highly motivated because they felt the 'real world' characteristics of the project. During a visit to Software AG, they were shown recent integration software improvements and a roadmap, and they got a guided tour including visiting Software AG's real IBM mainframes. All these things enhanced student motivation very much. At the end of the SAG-Tours project, the students highly ranked the value of the project for their computer science education.
- *Team dependency*: Although legacy modernization projects are doable by students, at least reasonable motivated students are required, who want to dig into the game. The skills of students involved in the SAG-Tours project have been 'average' to 'above average'. A project team of 'below average' students would likely not have finished the project with this success. As said before, the students had a good knowledge of Java, and database and networking technologies but no prior exposure to Web services, integration software or even mainframe technology.
- *Reasonable environment*: As known from software projects in general, a proper organizational framework is required. This includes a project room with dedicated project team time and at least a drawing board, dedicated machines, occasional pizzas, a project poster to show etc.

2) SOA / Mainframe technology

- *SOA principles / Web services*: As expected usage of Web services and SOA principles such as service-oriented design, process analysis, well-defined interfaces, etc., resulted in a reasonably better 'interface-oriented' software and system design.
- *'Interesting mainframes'*: Mainframe technology in general was seen as a quite interesting topic, especially when the students recognized that many topics such as transaction processing have been around for quite a while in computer science history.
- *Willingness to 'struggle through it'*: Especially the interplay of all the technologies and a mostly new terminology for (although partially known) concepts did require from the students some willingness to 'struggle through it'. This clearly demanded student motivation. But once that motivation had been raised, the SAG-Tours project clearly became a self-runner for the supervising professors.

VIII. Conclusion and Future Work

There is a common belief that SAO enablement on the application layer (also known as business logic wrapping) is a very expensive approach to modernization of a legacy application in case of a lack of legacy skills because of huge efforts required to understand the legacy application. However, the case study showed that this approach can require rather few efforts if the right technologies are used.

In other words, integration of legacy applications into a SOA is neither impossible nor too complex. Simple evidence of this fact is that in the scope of the SAG-Tours project, final-year bachelor students were able to do this within a year, with an average effort of one day per week for 10 students. The students were experienced in Java coding and network technologies in general. But they had no previous experience in Natural coding, mainframe technologies (Adabas in particular) or integration of legacy applications.

However, for a full SOA, we have to add some more components to complete the integration architecture in Figure 12. But the fact that it was possible without any major problems in the context of the SAG-Tours project carried out by bachelor students [8] shows that it is not necessary to have a disproportional knowledge or to make huge efforts for such integration.

Whether a particular integration would be possible in a heterogeneous system environment (e.g., without using newly offered components from the same vendor) or what efforts would be required could be evaluated in future work within another legacy modernization project.

Acknowledgments

We would like to especially thank our students [16] from the SAG-Tours project, who were instrumental in implementing the above concepts.

Irina Astrova's work was supported by the Estonian Centre of Excellence in Computer Science (EXCS) funded mainly by the European Regional Development Fund (ERDF).

References

[1] A. Koschel, C. Kleiner, and I. Astrova. Mainframe application modernization based on service-oriented architecture: a practical industry cooperation case study. Proceedings of IARIA Computation World: Future Computing, Service Computation, Cognitive, Content, Patterns, ComputationWorld 2009, 298 – 301, 2009.

[2] S. Conrad, W. Hasselbring, A. Koschel, and R. Tritsch. Enterprise application integration: Grundlagen - Konzepte - Entwurfsmuster – Praxisbeispiele. Spektrum Akademischer Verlag, 2005.

[3] C. Date, An introduction to database systems 5th Edition, Volume I, 1992.

2010, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

[4] A. Erradi, S. Anand, and N. Kulkarni. Evaluation of strategies for integrating legacy applications as services in a service oriented architecture. In IEEE SCC, pages 257–260. IEEE Computer Society, 2006.

[5] S. Englet. Wiederverwendung von Legacy Systemen durch einen bottom up Ansatz bei der Entwicklung einer SOA. In H. Hegering, A. Lehmann, H. Ohlbach, and C. Scheideler, editors, GI Jahrestagung (1), volume 133 of LNI, pages 96–100. GI, 2008.

[6] W. Keller. Enterprise Application Integration. Erfahrungen aus der Praxis. Dpunkt-Verlag, 2002.

[7] D. Krafzig, K. Banke, and D. Slama. Enterprise SOA: Service-oriented architecture best practices. Prentice Hall, 2005.

[8] C. Kleiner and A. Koschel: Legacy vs. Cutting edge technology in capstone projects: What works better? Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE09), 2009.

[9] G. Lewis, E. Morris, and D. Smith. Analyzing the reuse potential of migrating legacy components to a service-oriented architecture. In CSMR '06: Proceedings of the Conference on Software Maintenance and Reengineering, pages 15–23,Washington, DC, USA, 2006. IEEE Computer Society.

[10] G. Lewis, E. Morris, D. Smith, and L. O'Brien. Serviceoriented migration and reuse technique (smart). In STEP '05: Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice, pages 222–229, Washington, DC, USA, 2005. IEEE Computer Society.

[11] Software AG, SAG-Tours: SOA integration projects, Application Document (Technical Report), Darmstadt, 2007.

[12] D. Smith. Migration of legacy assets to service-oriented architecture environments. In ICSE COMPANION '07: Companion to the proceedings of the 29th International Conference on Software Engineering, pages 174-175,Washington, DC, USA, 2007. IEEE Computer Society.

[13] G. Starke and S. Tilkov (Edts.). SOA-Expertenwissen: Methoden, Konzepte und Praxis serviceorientierter Architekturen. Dpunkt-Verlag, 2007.

[14] T. Erl. SOA Design Patterns, Prentice Hall, 2009.

[15] G. Lewis, E. Morris, and D. Smith. Analyzing the reuse potential of migrating legacy components to a service-oriented architecture. In CSMR '06: Proceedings of the Conference on Software Maintenance and Reengineering, pages 15–23,Washington, DC, USA, 2006. IEEE Computer Society.

[16] A. Koschel, C. Kleiner, M. Safris, A. Budina, O. Efimov, A. Morozov, W. Schaefer, D. Schaefer, S. Kirstein, W. Zlobin, A. Kolesnikov, and A. Brockwitz. Abschlussdokumentation für das Bachelor-Projekt 'SAG Tours'. FH Hannover, Fakultät IV. 2008.

[17] G. Canfora, A. Fasolino, G. Frattolillo, and P. Tramontana. A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures. Journal of Systems and Software, 81(4):463–480, 2008.

[18] H. Sneed. Integrating legacy software into a service-oriented architecture. CSMR'06, IEEE CSP, 2006.

[19] SAP. NetWeaver Open Integration Platform. https://www.sdn.sap.com/irj/sdn/developerareas/netweaver Last access: June 2010.

[20] K. Channabasavaiah and K. Holley. Migrating to a service-oriented architecture. IBM White paper, 2004.

[21] Oracle. Oracle IT modernization series: The types of modernization. Oracle White paper, 2006.

[22] F. Mohammed. Oracle SOA Suite. Sys-Con XML Journal, 2007.

[23] Sofware AG. webMethods ApplinX. https://www.softwareag.com/ApplinX Last access: June 2010.

[24] Microsoft Corporation. Enabling real-world SOA through the Microsoft Platform. 2006.

[25] J. Ziemann, K. Leyking, T. Kahl, and W. Dirk. Enterprise model driven migration from legacy to SOA. Software Reengineering and Services Workshop, 2006.

[26] A. Erradi, S. Anand, and N. Kulkarni. Evaluation of strategies for integrating legacy applications as services in a service oriented architecture. In IEEE SCC, pages 257–260. IEEE Computer Society, 2006.

[27] T. Suganuma, T. Yasue, T. Onodera, and T. Nakatani. Performance pitfalls in large-scale java applications translated from cobol. In OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object oriented programming systems languages and applications, pages 685–696, New York, NY, USA, 2008. ACM.

[28] Oracle. Oracle IT modernization series: Approaches to IT modernization. Oracle White paper, 2009.

[29] T. Laszewski. SOA and the mainframe: Two worlds collide and integrate. http://www.theserviceside.com/tt/articles/content/SOAandtheMainframe/article.html, 2009 Last access: June 2010.

2010, © Copyright by authors, Published under agreement with IARIA - www.iaria.org