Style-based Architectural Analysis for Migrating a Web-based Regional Trade Information System

Simon Giesecke
Software Engineering Group
Carl von Ossietzky University
26111 Oldenburg, Germany
giesecke@informatik.uni-oldenburg.de

Johannes Bornhold
Software Engineering Group
Carl von Ossietzky University
26111 Oldenburg, Germany
johannes.bornhold@informatik.uni-oldenburg.de

ABSTRACT

In this paper, we present the MIDARCH method for selecting a middleware platform in Enterprise Application Integration (EAI) and migration projects. Its specific contribution is the use of architectural styles (MINT Styles) as a vehicle for binding architectural knowledge. In addition, an ongoing case study is presented which applies the MIDARCH method to a web-based regional trade information system. The project involves the integration of three subsystems, which have been developed rather independently in the past, two of which are already web-based. The major motivation for migrating the system is to improve evolvability of the system and to make it more apt for the supply to a larger number of customers.

Keywords

ArchiMate, Architectural Description Languages, Architectural Style, Cocoon, Enterprise Application Integration, Java, Web Migration, xADL

1. INTRODUCTION

Software reengineering is concerned with the transformation of legacy software systems. Many reengineering projects aim to modernise systems that are based on outdated technologies, e.g. mainframe systems, that are no longer properly maintained. The transformation target of many business information systems are web-based platforms. Today, we are in the situation that reengineering projects are also concerned with systems that are already web-based, but need to be transformed for a particular reason. Typical reasons are:

- The employed implementation technologies are already outdated themselves.
- Requirements have changed and the chosen architecture is no longer adequate.
- Multiple systems are to be integrated.
- The implementation technologies have been used in an inadequate way.

These reasons are not exclusive to web-based software systems. In particular the latter reason appears like a generic maintainability problem. However, in the case of implementation technologies for web-based systems, such as Sun's Java Server Pages [24] or Apache Cocoon [1], many systems have been developed with only a premature understanding of the architectural style endorsed by these technologies.

The migration of such systems is often not possible without modifying the internal structure of the participating systems, because they do not expose adequate interfaces. Migration projects therefore offer an opportunity to restructure the participating systems, and enabling the integration using advanced middleware techniques. Such approaches are also termed Enterprise Application Integration (EAI), which is a special case of migration that involves multiple, heterogeneous systems.

In this paper, we propose the MIDARCH method for supporting the migration business information systems based on architectural descriptions and architectural styles that are induced by the middleware used (Middleware INTegration Styles, MINT Styles). The main feature of the method is the use of MINT Styles as a vehicle for enabling reuse of architectural design knowledge across multiple migration projects. In addition, we describe a case study of an application of the method which involves a migration project concerning a web-based regional trade information system, and present first results of the case study.

1.1 MIDARCH Research Project

The major goal of the MIDARCH research project [10] is the development and validation of a software engineering method for migrating business information systems based on architectural descriptions that exploit the benefits of architectural styles which are endorsed by the middleware used. The method is called MIDARCH (MIDdleware style based ARCHitectural integration). Architectural styles capture architectural knowledge and provide the basis to reason about families of related software architectures. The method supports the transfer of knowledge from one integration project to another by creating and analysing architectural descriptions that are explicitly based on some architectural style. Through this, experiences from one integration project do not remain constrained to the specifics of the concrete architecture of the subject system, but can be related to the MINT Style. Thus, integration

knowledge can be reused in other projects that consider the use of the same style.

1.2 Overview

In the remainder of the paper, we provide the details of fundamental topics that are required for the rest of the paper in Section 2. The setting of the case study is described in Section 4. The research approach taken is outlined in Section 5, which includes the outline of the general procedure of the MIDARCH method. Preliminary results of the ongoing case study are presented in Section 6. The paper ends with a conclusion (Section 7).

2. FOUNDATIONS

In this section, we discuss some foundations we deem necessary to understand the remainder of the paper. First, we introduce the general research areas of Enterprise Application Integration (Section 2.1) and Service-oriented Architectures (Section 2.2), which form the conceptual basis of our approach. Afterwards, we discuss the role of architectural styles for our research (Section 2.3) and present the Architectural Description Languages we use for modelling our case study (Section 2.4).

2.1 Enterprise Application Integration

Enterprise Application Integration (EAI) is a special form of software reengineering, concerning the integration of legacy business information systems. The term Enterprise Application Integration is essentially used in two different meanings: In one view, EAI is used in a restricted sense to denote a specific approach to the integration of information systems which employs off-the-shelf EAI components and is non-invasive with respect to the subsystems to be integrated [18]. EAI in this first view always leads to loosely integrated systems. In this view, EAI is distinct to (invasive) migration.

The other view on EAI refers to any approach to the integration of information systems at the application level as EAI [11], which is the view we take as well. In this view, EAI is a special case of migration that involves multiple, heterogeneous systems. However, in our work, we focus the aspect of the middleware that is used for integration. We have a wide view of middleware, i.e. we regard any software layer that it used for enabling communication of (often, but not necessarily, remote) subsystems or components as a middleware platform. In the case of web-based applications, e.g. Apache Cocoon or Servlet containers are considered middleware platforms.

In [11], three architectural levels are distinguished: Business architecture, application architecture and technology architecture. Integration at each of these levels is described as inter-organisational processes, Enterprise Application Integration and middleware integration, respectively. We are concerned with the latter two levels: The selection of a middleware platform provides the infrastructure for the implementation of the application architecture, and thus for achieving Enterprise Application Integration.

2.2 Service-oriented Architectures

Service-oriented Architectures (SOAs) [12, 20] can both be regarded on a concrete, technical and on an abstract, conceptual level. The first possibility involves the realization of components using specific technologies creating a service infrastructure as web services and the use of technologies such as WSDL, SOAP, UDDI, etc. [29]. The conceptual view generalises this approach and does not necessarily require a specific service-oriented realization, but uses services at the elements of architectural description. One option for implementing a conceptual service-oriented architecture is, of course, using explicitly service-oriented technologies, but other technologies may be used as well. In the latter case, a well-founded mapping of service-oriented concepts to the concepts endorsed by the implementation technology should be provided (cf., e.g., [17]).

2.3 Architectural Styles

Architectural styles [23] and architectural patterns [3] are similar concepts, which we deem equivalent for the purposes of this paper and only use the former term, in order to better distinguish them from lower-level design patterns [7].

Classical general-purpose architectural styles are the pipe-and-filter, blackboard, layered, and event-based styles [23], and variants thereof. These styles are often used in an informal manner to establish a common vocabulary for architectural design elements. In the context of ADLs, formal specifications of architectural styles are used, which define families of architectures or impose constraints on concrete architectural configurations. A special case of architectural styles are those induced or endorsed by an implementation platform, especially by middleware platforms which make use of high-level abstractions [6]. We refer to such architectural styles as MINT Styles.

2.4 Architectural Description Languages

Over the last 15 years, many Architectural Description Languages (ADLs) have been developed with different goals and approaches [22]. There is no broad agreement on a definition of an ADL, for example there have been some debates on whether UML qualifies as an ADL – be it UML as such or a specific usage of UML [8, 21]. We do not intend to provide a rigorous definition here either. However, we briefly describe two ADLs, which play some role in our research project: xADL 2.0 and ArchiMate.

2.4.1 xADL

xADL 2.0 [5] (we will use the brief form xADL in the following) is an ADL which evolved from a traditional line of ADLs at the University of California at Irvine. xADL is a collection of extensions to the xArch [4] core ADL, which is meant to be a "standard, extensible XML-based representation for software architectures" [4]. xADL was designed in a modular and extensible fashion which is based on the modularity and extensibility of XML and XML-Schema [28]. Tool support is available on different levels. On the syntactical level, xADL benefits from its XML basis. Generic tools can be used out of the box, e.g. XML validators can validate xADL and also custom extensions. Another example are syntax-based editors, which can understand the schemas and adopt to custom extensions automatically. Specific xADL tools [26] are a data binding library and a generator which automatically generates a custom data binding library from a XML-Schema. Additionally some higher-level tools are available.

2.4.2 ArchiMate

ArchiMate [19] is not a traditional ADL, insofar as it does not focus exclusively on software architectures, but is used to describe enterprise architectures, which place – in the definition of ArchiMate's developers – software architecture in the context of the organisation(s) using the software. An enterprise architecture is "a coherent whole of principles, methods, and models that are used in the design and realisation of an enterprise's organisational structure, busi-

ness processes, information systems, and infrastructure" [19, p. 3]. The language closely resembles the UML, and can be mapped onto the UML, but it is not merely an extension of the UML metamodel. ArchiMate supports a layered modelling approach in essentially three layers: business, application and technology architecture (similar to [11]). It is based on the concepts of SOA, so services play a central role on each of the layers.

3. MIDARCH-METHOD

In this section we describe the generic MIDARCH method. The activities proposed by the generic MIDARCH method are shown in Figure 1. The activities shown are quite coarse-grained and must be described on a more fine-grained level to be effectively implementable. Furthermore, no backsteps that might be necessary are indicated in the figure, but the application of the method will be very iterative in practise.

The steps can be structured into four activities, which consist of several subactivities:

Activity 1: Scoping and Goal Definition The first activity consists of two subactivities: Scope Definition and Requirements Elicitation.

Define Scope Scope Definition involves creating a list of (sub)systems to be integrated.

Determine Current and Future Requirements
Requirements Elicitation involves the determination of the future requirements on the system, which motivate the need for the integration, in detail, as well as the current requirements on the system. Current requirements may already be documented, but it must be ensured that they are documented in a form that can be compared to the future requirements.

As part of the requirements delta, high-level goals of the integration are identified, which are important for the second activity.

The requirements elicitation process is influenced by the scope determined in the previous step, e.g. because current requirements can only be determined on the basis of a specific system scope.

Afterwards, it must be determined if the requirements match the functionality of the systems to be integrated. In this case, scoping must be reconsidered. There may either be functionality missing, in which case it must be determined whether another (internally or externally available) system can be considered in the integration. If some functionality is not available in an existing system, it must be planned to be newly implemented. There may also be (sub)systems that are not needed to fulfil the future requirements.

Activity 2: Preparation The second activity consists of two subactivities preparatory with respect to Activity 3.

Develop Project-Specific Quality Model A project-specific quality model is developed, which is focused on the migration goals identified in the previous activity.

Model Current Architecture The current architecture is modelled using a suitable modelling language/method. One goal of the overall research project is to evaluate the suitability of different architectural description languages for this purpose. While probably no single modelling language is suited for modelling any system, we contribute to the body of knowledge on the use of modelling languages, and thus provide support for the selection of modelling languages in the future. Suitability here involves the ability to express distinctive features of the current and future architectures (which should be modelled using the same notation and method to ensure commensurability) and to analyse the system or architecture characteristics that occur in the quality model.

Activitiy 3: Architecture Exploration The third activity models and explores different architectural alternatives. It may be considered the core of the method and consists of four subactivities.

Choose/Model MINT Style In each iteration of this activity, one or more MINT Style(s) may be considered. At least in the first iteration, multiple styles should be considered to enable a meaningful assessment in the fourth subactivity. In the method description, we assume that only one style is considered for better readability.

The style description may be either taken from a taxonomy of styles or may be specifically created. One goal of the research project is to provide a taxonomy of MINT Styles and an initial body of knowledge which supports the selection of styles with suitable quality characteristics.

A further goal of the research project is to evaluate the usefulness of different levels of rigour of style descriptions, most importantly informal style descriptions that may include example architectures as opposed to formal style descriptions in an architectural description language as a constraint for concrete architectures in the same language.

Model Candidate Architecture The candidate architecture is modelled on the basis of the chosen style and the current architecture to reflect future requirements.

Evaluate Candidate Architecture The Candidate Architecture is evaluated against the quality model using a scenario-based architectural evaluation method such as ATAM [16].

Assess Evaluation Results The evaluation results of the candidate architectures developed so far and the current architectures are assessed. If the results are found to sufficiently support the integration goals, the activity ends, otherwise further styles and architectures must be considered.

Activity 4: Architecture Selection and Adoption The last activity is not considered within the method in detail, but is included here to make the method complete within the context of its intended application.

Choose Target Architecture Based on the results of the previous activity, a target architecture is chosen, which is based on the best architecture(s) that were identified in the last step of Activity 3. If necessary, details which have been left out in the previous activity are amended.

Adopt Target Architecture The systems are integrated and possibly modified according to the chosen target architecture.

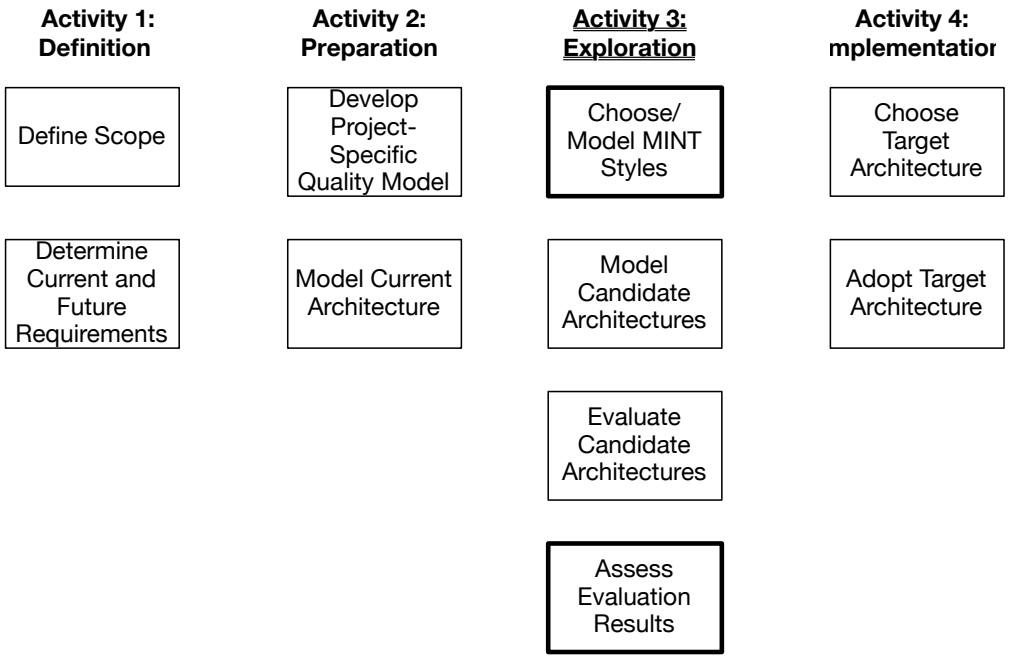

Figure 1: Activities of the MIDARCH method

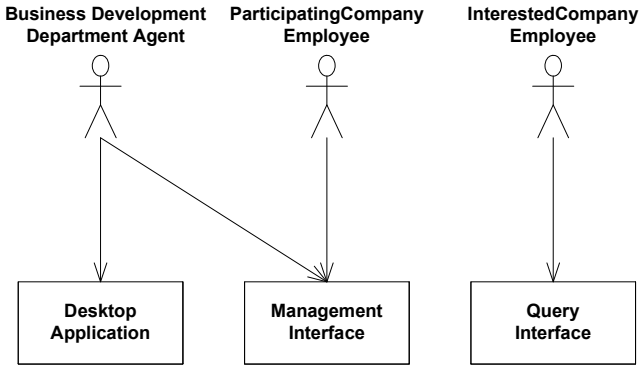

Figure 2: User roles and their relationships to the system's interfaces

4. CASE STUDY

In this section we give a brief overview of the regional trade information system which is the subject of the case study (Section 4.1) and the migration goals which shall be achieved (Section 4.2).

4.1 System Purpose

The trade information system is provided as a supporting tool for sustainable regional development. The general idea behind this system is to make information on the economic potential of a region available to companies to increase regional business collaboration.

As indicated in the introduction, the subject system of the case study is separated into three subsystems which have been developed rather independently in the past. Each of these subsystems currently provides a distinct user interface. Two of these interfaces are already web-based. There are three roles of users accessing these interfaces. The relationships of user roles and interfaces are shown shown in Figure 2. The business development departments of the counties and municipalities in the covered region collectively form the current customer, to which our cooperation partner provides the service.

First, an access-controlled web interface is used to collect and manage the data about the participating regional companies. It has two main groups of users. The first group represents the agents at the business development departments. These users can administrate the data of their district's companies and manage the users of the second group, which represent the participating companies themselves.

Second, a web-based query interface is publicly available. It presents information about companies which are located in the covered region. The ability to query this information by different filter criteria facilitates finding potential collaboration partners among regional companies, and thereby supports building regional business networks. The data about each company consists of statistical and address data as well as information on offered technologies, special skills and cooperation interests.

As a kind of glue to the data-management interface the presentation of each company's data contains a hyperlink to edit it. Through this link, new company users can use a registration mechanism to request a login to the system and the necessary rights to edit their data records. In addition the business development department agents have the ability to export their data in a spreadsheet file format.

The third user interface is provided by a desktop application which goes back to a point in time before the development of the other subsystems. Only part of the functionality offered via this interface is still in use. It allows to manage private additions to the data records, which are used only internally by the business development departments. Currently, the new management subsystem provides an export facility which allows the users of the desktop application to manually download the up to date data in the desk-

top application's proprietary file format and afterwards import it to update their locally stored data.

The desktop application was originally also used to manage the data, which is now managed through the web-based user interface. Originally, the data was sent by the business development departments to the service provider by email and the service provider manually combined the data fragments to feed the query subsystem.

4.2 Migration Goals

There are three main migration goals: First, the system shall be made ready for use by multiple customers. Second, it shall be made more evolvable. Third, the availability of the system should be improved.

The first goal must be seen in the context that this system was originally developed to be used in a single instance for a single region and therefore no effort was made to support multi-customer capabilities and customer-specific customisation needs. In the future, this system shall be offered to multiple customers (i.e., other regions). This means on the one hand that a greater effort must be put on support the adaptability to special customer needs with a manageable amount of human resources. On the other hand, special care must be taken in the product development process to either support hosting of multiple instances and a (semi)automated update-mechanism to new releases of the product, or to add multi-customer capabilities to a single instance of the system.

The second major goal is to increase the system's evolvability [9, ch. 2.2.5.2]. The system itself and its parts evolved over time. Adoption to new requirements has become a challenging task which requires involved developers to be familiar with many parts of the current system. Evolvability is enabled at the architectural level, which must be adequately reflected in the system's implementation.

The third goal is to increase the availability of the system, e.g. by introducing redundant components. Availability becomes more important when more customers are using the system.

5. APPROACH

In this section we describe the MIDARCH method's adaptation to the case study. This section is like section 3 structured according to the activities of the MIDARCH method.

Activity 1: Scoping and Goal Definition

Define Scope In the case study we selected the three interfaces *QueryInterface*, *ManagementInterface* and *DesktopApplication* which are described in Section 4 and the subsystems they depend on.

Determine Current and Future Requirements In the case study, the requirements are elaborated on the basis of different internal documents. These documents contain information on the long-term vision for the software system, non-functional requirements and use cases.

Activity 2: Preparation

Develop Project-Specific Quality Model We use an approach based on GQM (goal/question/metric) [27] to create the quality model. Software quality has different aspects: the internal (cf. [25]) and external (cf. [2]) quality of the software architecture description itself, and the internal and external quality of the software system it represents.

Model Current Architecture We plan to use ArchiMate and xADL (see Section 2.4) to model both the current and the candidate architecture in the case study. ArchiMate provides us with the ability to see the architecture in an organisational context and to connect the application domain with both the business and the technology domain. With xADL, on the other hand, we are able to model the architecture on the application level in detail and to integrate the xADL architecture description with the implementation artefacts. The connection between both languages is done on the application level, enhanced with relations to the other levels (within the ArchiMate description) and related to the development artifacts (within the xADL description).

Activitiy 3: Architecture Exploration

Choose/Model MINT Style In the case study, we are exploring the suitability of xADL to describe architectural styles.

Model Candidate Architecture This architecture shall be modelled in xADL and ArchiMate analogously to the model of the current architecture from Activity 2.

The last two steps of this activity **Evaluate Candidate Architecture** and **Assess Evaluation Results** do not need any special adaption to the case study.

Activity 4: Architecture Selection and Adoption

Choose Target Architecture In the case study, one detail which must be added to the chosen target architecture is the information which implementation artifacts correspond to the architecture components, interfaces and connectors. This shall be achieved through the Java extensions which are part of xADL.

Adopt Target Architecture A prototype of the chosen architecture is created which shall show how the xADL model of this architecture can be connected to the implementation artifacts and thus be integrated with the future steps in the development process. This is also the last step taken in the case study. The adoption of the examined systems to the target architecture is a task out of the scope of the case study.

6. PRELIMINARY RESULTS

In this section we describe the current state of the case study. Activity 1 has been virtually completed and we are currently in Activity 2. The current architecture has been partially modelled, i.e. coarse-grained components, their dependencies and the information flow have been identified. These are described in Section 6.1. An example of an ArchiMate model of a part of the system is presented in Section 6.2. In this model, the coarse-grained components are refined and linked with information on the business and technology levels. Finally, we describe the current middleware technologies and their usage in Section 6.3, and identify potential problems with respect to the migration goals.

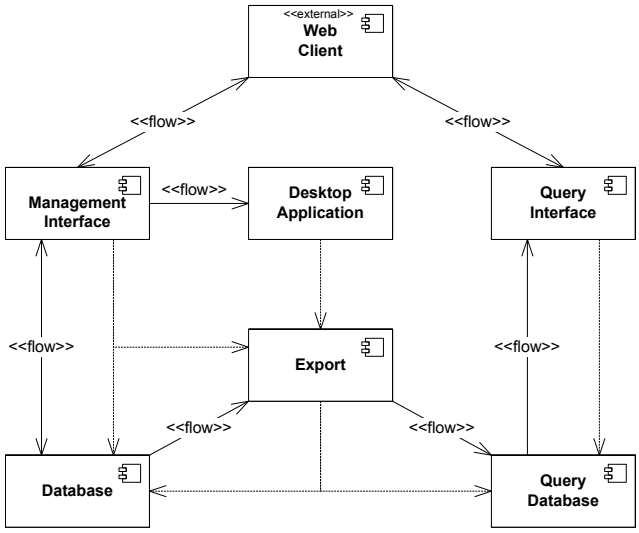

Figure 3: Component dependencies and information flow

6.1 Dependencies and Information Flow

An overview of the component structure and the information flow of the subject system is shown in Figure 3. The two viewpoints are combined in this diagram, because at this abstraction level there are only few elements and the diagram can still be understood.

The different parts of the system have been developed at different times. The oldest part is the desktop application which was originally used to manage the data. The agents at the business development departments used this system to collect the data of their region. Periodically they sent their data to the Application Service Provider (ASP) of the query interface where this data was merged manually and fed into the query database. Now, the direction of the information flow is inverted. The desktop application is updated from the database of the management interface. It contains an export facility which creates a snapshot of the data in the desktop application's proprietary file format. The users can download the file and use it to update their local application. This is shown in Figure 3 by the flow lines from *Database* to *ManagementInterface* and from there continued to *DesktopApplication*. Because the *Export* is needed to create the file, there is a dependency from *DesktopApplication* to *Export*.

The data management interface is the youngest part of the system. It has a database of its own. Note the dependency from *ManagementInterface* to *Database*. The users (agents at the business development departments and employees of the participating companies) can update the data through its web-based interface. The collected data is then transferred periodically into the *QueryDatabase*. This results in a delay before updates of the data are reflected in query results. The information flow is shown by the bidirectional flow lines between *WebClient*, *ManagementInterface* and *ManagementInterface*, *Database*. The propagation to the *QueryInterface*, and thus to the query results, can be read by the directed flows from *Database* to *Export* continued to *QueryDatabase* and finally reaching *QueryInterface*. There is no information flow in the reverse direction.

The last part is the query interface. For historical reasons it has its own database and a slightly different database schema than the management interface. As shown in Figure 3 it is only direct dependent on the *QueryDatabase*. But as mentioned above, its data is updated by the information provided by the *Export*, so to be useful over a longer time, it needs the *Export*, this can be concluded from the flow line between *Export* and *QueryDatabase*.

6.2 ArchiMate Model

Figure 4 shows an example excerpt from an ArchiMate model of the current architecture, which shows the parts necessary for the registration of new users. Its layout is based on an example given in [14]. When a new company's employee requests a login to manage the data about his company, he is in the role *Company* and uses the *RegistrationService* to request his new login. This service is realised on the business layer by the business process *Registration* which depends on the *PostOfficeService*. This service is realised on the application layer by the *PostOffice* and responsible for informing the right person in the *BusinessDevelopmentDepartment* role (*BDD*) to *Check* and possibly *Accept* this request. In the bottom part, this figure shows that the *PostOffice* component needs an available *EmailService* which, in the current case, is realised by a *MailServer* on the technology layer which is installed on some device that in not further specified. This example demonstrates ArchiMate's ability to show the relations between the different architectures on the business, application and technology layers.

6.3 Middleware Technologies and Their Usage

We focus on the subsystems of the regional trade information system which already provide a web-based interface, i.e. the query and management interfaces. Both subsystems are based on Apache Cocoon [1] but use the technology in different ways.

Apache Cocoon is a "a web development framework built around the concepts of separation of concerns and component-based web development" [1]. Cocoon is designed as a Java Servlet. Requests are processed in a pipeline in which several components (*filters*) are hooked together, i.e. it uses a variant of the pipe-and-filter architectural style. Within the pipeline, filters communicate via a stream of SAX events. The entry to the pipelined processing is a *generator* followed by an arbitrary number of *transformers* and finalised by a *serialiser* which typically serialises the SAX events into an HTML output. Apart from this basic concept, Cocoon has the facility to read from and to serialise to many data formats like XML, graphic formats, etc. Many extension filters are provided off-the-shelf, which can be integrated into the pipeline and further support the development of web applications. With regard to the regional trade information system the most important extensions are a framework for form handling and extended control flow support (CForms).

The query interface of the regional trade information system does not use special extensions of Cocoon. Most of its functionality is embedded in XSP documents (a Cocoon-specific language similar to Java Server Pages) which allow Java code to be embedded into XML documents. From these XSP documents, direct queries to the underlying database are made and the results are written into an XML representation of the query result which is then further processed by the following filters. These filters transform the query result into an appropriate HTML representation.

From a very abstract point of view, the management interface works in a similar fashion. The first filters perform some operations on the data and the following filters transform the result into a HTML rep-

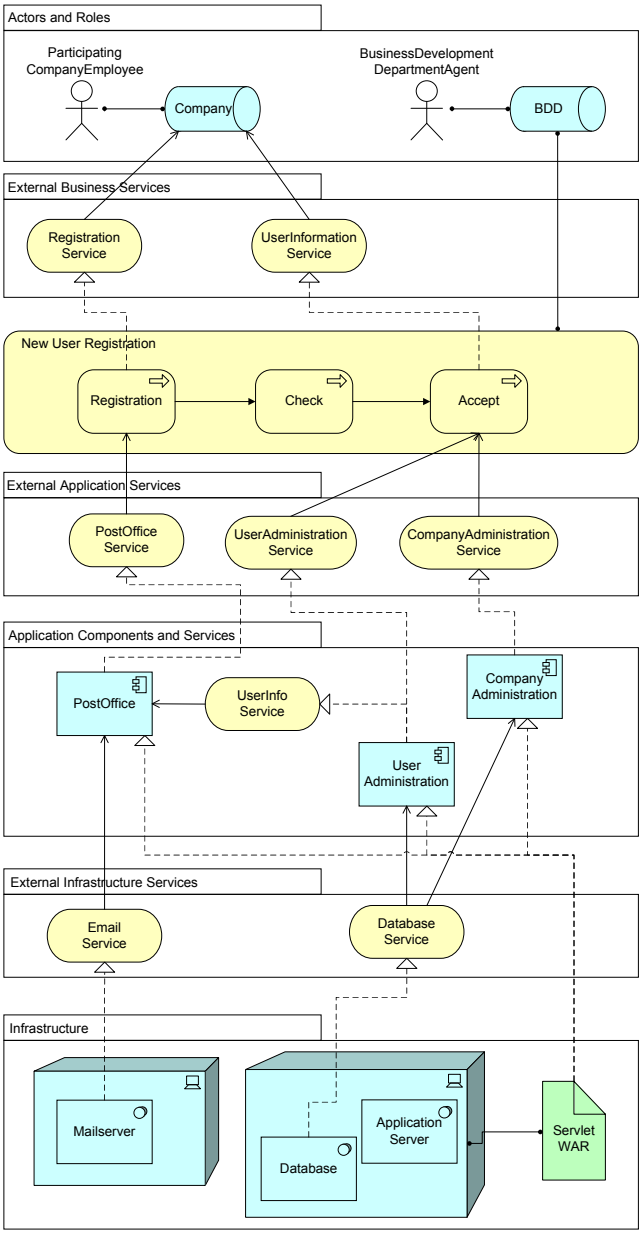

Figure 4: Partial ArchiMate Model of the Case Study System

resentation. Differences appear with a closer look to the first part of the pipeline. The management interface makes use of Cocoon's form framework, which allows for better handling of form data and constraints, and of Cocoon's control flow framework, which allows to send forms with a blocking function call and to formulate control flows in an explicit, closed form. This framework is realised through a JavaScript API which provides access to underlying Java objects. The second difference in comparison to the query interface is the way, data is accessed. In the management interface, all data queries and manipulations are performed with Java objects which map to the underlying data storage (object-relational mapping using Hibernate [13]).

Potential Problem Areas

There are several problems with the current architecture with respect to the migration goals. First, especially the query interface is closely coupled to its underlying database, which is one of the reasons why it still uses a database of its own with an old schema. Because of this, it is technically hard to adopt new requirements that have an impact the database schema, and the effort is difficult to estimate.

Second, the mechanism which transforms the intermediate results to a HTML representation has been identified as another difficulty in practice. An own proprietary language has been developed for the intermediate results which has grown over time and is nearly unmaintainable now.

As a third problem area there are many tight couplings within the Java implementation of the data model, so that requirement changes often result in changes at many different places of the implementation which makes it harder to parallelise development tasks. For this reason, it is not easy to isolate the data tier from the presentation tier in the query and management subsystems, which is why we did not split up the coarse-grained *QueryInterface* and *ManagementInterface* components is Figure 3.

Part of the implementation of the query interface has been reused in the management interface, but has been modified afterwards. Modifications must be ported manually in every case.

7. CONCLUSION

In this paper, we presented the MIDARCH method for integrating heterogeneous business information systems on the architectural level. Many integration projects are performed ad-hoc, i.e. without using a systematic method specifically supporting the integration process. Reuse of experience from other projects thus remains entirely implicit. A few other methods for integration projects have been proposed: Kazakov [15] proposed a semi-automated method for software integration, which requires specifications of the involved software components in the SHIQ description logic.

We do not specifically aim to automate integration efforts, but primarily intend to make reuse of integration knowledge more effective. Tool support for this process is a subsidiary part of the overall research project.

Methods for the development and composition of web services, e.g. Semantic Web approaches, are not in the focus of our work, since we are dealing with pre-web-service legacy applications.

We presented the current state of a case study which evaluates the MIDARCH method. One of the next steps is the modelling of the

current and the endorsed usages of Cocoon explicitly as a MINT Style. This and future case studies will provide feedback that will be used to improve the method, and contribute to the knowledge base on the quality characteristics of architectural styles that is necessary for effective application of the method.

Additional future work includes the creation of a taxonomy of middleware platforms based on the MINT Styles they endorse. This taxnomy would allow the stepwise refinement of integration techniques within the exploration process (Activity 3 of the MIDARCH method).

8. ACKNOWLEDGEMENTS

Thanks to Bernd Kramer and Kai Bruns at Regio GmbH, Oldenburg. This work has been partially supported by the German Research Foundation (DFG), grant GRK 1076/1.

9. REFERENCES

[1] Apache Foundation. Apache Cocoon. `http://cocoon.apache.org/`, 2006.

[2] F. P. M. Biemans, M. M. Lankhorst, W. B. Teeuw, and R. G. van de Wetering. Dealing with the complexity of business systems architecting. *Systems Engineering*, 4(2):118–133, 2001.

[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, 1996.

[4] E. Dashofy, D. Garlan, A. van der Hoek, and B. Schmerl. xArch, 2006. `http://www.isr.uci.edu/architecture/xarch/`.

[5] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. A comprehensive approach for the development of modular software architecture description languages. *ACM Trans. Softw. Eng. Methodol.*, 14(2):199–245, 2005.

[6] E. Di Nitto and D. Rosenblum. Exploiting ADLs to specify architectural styles induced by middleware infrastructures. In *Proceedings of the 21st international conference on Software engineering*, pages 13–22. IEEE Computer Society Press, 1999.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[8] D. Garlan, S.-W. Cheng, and A. J. Kompanek. Reconciling the needs of architectural description with object-modeling notations. *Sci. Comput. Program.*, 44(1):23–49, 2002.

[9] C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[10] S. Giesecke. A method for integrating enterprise information systems based on middleware styles. In *International Conference on Enterprise Information Systems (ICEIS'06) Doctoral Symposium*, 2006. Accepted for publication.

[11] W. Hasselbring. Information system integration. *Commun. ACM*, 43(6):32–38, 2000.

[12] M. N. Huhns and M. P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.

[13] JBoss Labs. Hibernate, 2006. `http://www.hibernate.org/`.

[14] H. Jonkers, M. M. Lankhorst, R. van Buuren, S. Hoppenbrouwers, M. M. Bonsangue, and L. W. N. van der Torre. Concepts for modeling enterprise architectures. *Int. J. Cooperative Inf. Syst.*, 13(3):257–287, 2004.

[15] M. Kazakov and H. Abdulrab. Semi-automated software integration: An approach based on logical inference. In *3rd International Conference on Enterprise Information Systems (ICEIS)*, pages 527–530, 2004.

[16] R. Kazman, M. Klein, and P. Clements. Atam: A method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University, 2000.

[17] I. Krüger and R. Mathew. Systematic development and exploration of service-oriented software architectures. In *WICSA '04: Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA'04)*, pages 177–187. IEEE Computer Society Press, 2004.

[18] R. Land and I. Crnkovic. Software systems integration and architectural analysis – a case study. In *Proceedings of the International Conference on Software Maintenance*, pages 338–. IEEE Computer Society, 2003.

[19] M. Lankhorst et al. *Enterprise architecture at work*. Springer, 2005.

[20] C. M. MacKenzie et al. Reference model for service oriented architecture 1.0. Public Review Draft wd-soa-rm-cd1, OASIS SOA Reference Model TC, Feb. 2006. `http://www.oasis-open.org/committees/download.php/16628/wd-soa-rm-pr1.p%df`.

[21] N. Medvidovic, D. S. Rosenblum, D. F. Redmiles, and J. E. Robbins. Modeling software architectures in the unified modeling language. *ACM Trans. Softw. Eng. Methodol.*, 11(1):2–57, 2002.

[22] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Trans. Softw. Eng.*, 26(1):70–93, 2000.

[23] M. Shaw and D. Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., 1996.

[24] Sun Microsystems. Java Server Pages Technology. `http://java.sun.com/products/jsp/`, 2006.

[25] W. B. Teeuw and H. van den Berg. On the quality of conceptual models. In S. W. Liddle, editor, *Proc. ER'97 Workshop on Behavioral Models and Design Transformations*, 1997.

[26] University of California at Irvine. xADL 2.0 – A highly extensible architecture description language for software and systems. `http://www.isr.uci.edu/projects/xarchuci/index.html`.

[27] R. van Solingen and E. Berghout. *The goal/question/metric method : a practical guide for quality improvement of software development*. McGraw-Hill, 1999.

[28] World Wide Web Consortium. Extensible markup language (XML), 2006. `http://www.w3.org/XML/`.

[29] O. Zimmermann, M. R. Tomlinson, and S. Peuser. *Perspectives on Web Services*. Springer, 2005.