Integrating a Ruby on Rails Application with Siebel using Java Messaging System

Timo Jalonen, Matti Kokkola

Helsinki University of Technology
Timo.Jalonen@sun.com, Matti.Kokkola@iki.fi

Abstract. This paper presents an integration strategy for integrating a Ruby on Rails application that has a MySQL database with Siebel customer relationship management (CRM) -system. We present the problem domain, discuss different technologies and integration patterns and derive an integration strategy from the requirements. We also use architecture trade-off analysis method (ATAM) to evaluate the architecture. In the end a proof-of-concept implementation validating the architecture is described.

Keywords: Siebel, JMS, integration, Ruby, Ruby on Rails, MySQL, messaging, EAI

1 Introduction

This paper is the seminar paper for "Special Course in Information Systems Integration: Business Process Integration" at Helsinki University of Technology, Software Business and Engineering Institute. This paper is based on an actual project assignment. The customer in question is Finnish Red Cross.

1.1 Background

The Finnish Red Cross (FRC) uses a Siebel CRM system for a variety of records. One database is the database of its branch offices including their nominees. The branch offices need to update this information from time to time, mainly around year-end, but also occasionally at other times. For this purpose there is a web-application (trust-registry), which has recently been rewritten. The new application was implemented using Ruby on Rails.

The Siebel CRM -databases and the trust-registry application need to be integrated in such a manner that the data displayed in trust-registry is correct, and any modifications to it are written back to Siebel. Siebel cannot, however, be trusted to be always available, whereas there is a need for the trust-registry application to be available to users on more or less 24/7 basis.

Currently the integration of the trust-registry application and Siebel is implemented using Siebel Enterprise Integration Manager (EIM). EIM is a set of database tables that can be used to transfer information to and from Siebel together with some tools to facilitate the transfer (see section "Integration technologies available to Siebel" for more information on EIM). The current implementation is based on manually operated periodical batch runs, that extract the information from Siebel and transfer it to trust-registry, and vice versa. Possible conflicts are resolved manually. Figure 1 shows what types of information are transferred between trust-registry and Siebel.

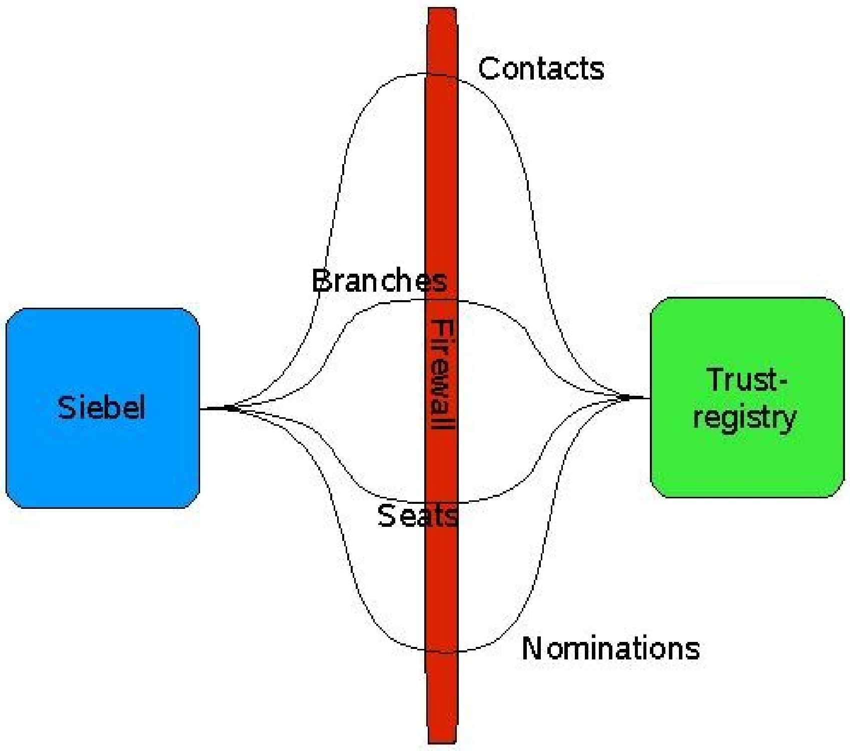

Fig. 1 Trust-registry's integration with Siebel

The result will be valuable for FRC, as the system built for its branch offices will be based on the findings of this study. As Siebel is one of the most widely used CRM systems, the findings of the study can be generalised to some extent. The results of the actual proof-of-concept (PoC) implementation can also be generalised if other organisations implement (or have a need for) technologically or functionally similar systems.

1.2 Research problem and objectives of the research

In this paper we aim to answer the following question: What are the possible means to integrate Siebel CRM and FRC’s Ruby on Rails -based trust-registry application and which one of them should be used for the actual integration?

To be able to answer this question we set one main objective and a set of sub-objectives. All the defined sub-objectives only served the main objective. For each objective, we defined a set of questions, which helped us in achieving the objectives.

The main objective was to define an integration strategy between Siebel CRM and FRC's trust-registry application. The first sub-objective was to define quality attributes for the integration strategy based on FRC's current and future needs. The questions to be answered we set as:

- What are the problems of the current integration?
- What are the quality requirements (security, performance, availability, etc.) for the new integration?
- What are the technology constraints of the integration?

We used interview as the research method to answer these questions.

The Second sub-objective was defined as finding and describing the possible integration techniques, strategies and patterns available in Siebel CRM. The questions related to this sub-objective were:

- What kind of integration interfaces does Siebel provide?
- What are the related integration patterns?

For this sub-objective we conducted a literature study.

Our third sub-objective consisted of evaluating the integration techniques, strategies and patterns against the defined quality attributes and defining a prototype architecture based on them. To this end we defined the questions as:

- What kind of quality trade-offs does each interface have?
- Which integration patterns will fulfil the selected quality requirements?

The research method for this sub-objective was a trade-off analysis.

Our fourth and final sub-objective was to implement a proof-of-concept construction of the defined architecture and evaluate it against the defined quality attributes. The question asked was: Is the proposed architecture feasible? To satisfy this sub-objective constructive research [6] was used.

1.3 Structure of this paper

This paper is structured as follows: in section 2 the research methods used are described, section 3 describes functional and quality requirements of the integration, and section 4 contains general discussion of integration styles and patterns. Section 5 discusses Siebel and integration techniques provided by it. Also the final integration style selection is introduced in the end of the section. Section 6 contains analysis of the design decisions and section 7 describes what was done during the proof-of-concept phase. Section 8 summarises the findings and proposes further research.

2 Research methods

During the project several different research methods were used. A brief summary of them is given below.

2.1 Constructive research

Main method of this work was constructive research. According to Kasanen et al, constructive research contains the following steps [6]:

1. Find a practically relevant problem which also has research potential.
2. Obtain general and comprehensive understanding of the topic.
3. Innovate.
4. Demonstrate that the solution works.
5. Show the theoretical connections and the research contribution of the solution concept.
6. Examine the scope of applicability of the solution.

To fulfil steps 1, 2 and 3 we interviewed customer's representative and made a small-scale literature study. For step 4, we implemented part of the innovated solution and demonstrated that it actually works. The solutions was also analytically analysed by applying the architecture trade-off and analysis (ATAM) method. Step 6 will be discussed in more detail in section 5.

2.2 Interview

An interview was conducted to elicit requirements of the actual integration. The interviewed person was Vesa Palmu, the system manager at FRC responsible for application development. He has very strong hands-on experience in software development.

The actual interview was made in a semi-informal way. The project team had prepared a set of questions beforehand. Questions were structured to follow different quality attributes. The interview was not recorded, but most of the discussion was written down during the session. In addition to quality requirements, also high-level functional requirements were elicited.

After the session actual requirements were elicited from the notes made during the interview. The list of requirements was then sent back to mr Palmu for validation and possible comments. Mr. Palmu seemed to be very busy during the project and thus our communication was limited to one meeting and one e-mail dialogue.

As only one person was interviewed, there is very strong possibility that the list of requirements elicited is not complete.

The interview was conducted on 22nd October 2007 in Espoo.

2.3 Literature study

Literature study concentrated mainly on Siebel manuals and literature on integration patterns. We also made searches to Google Scholar to find out scientific articles related to our work. The literature study can be seen to be rather small-scale. As focus of our work was to design and implement concrete integration strategy for a concrete and strictly scoped problem, the small scale should not be a problem.

2.4 Architecture trade-off analysis

All software projects are driven by some set of non-technical goals, usually business goals. On the basis of these goals another set of project goals are delivered: technical and quality goals, also known as non-functional requirements. ATAM is a method to determine whether these goals are achievable by the proposed architecture or not.

Naturally the evaluation of the architecture can not tell anything about the final product. If architecture has a certain property, can it be ensured that also the implementation has it? Most of the properties cannot be fully assessed, but indication of how the system will behave can be provided. [8]

The evaluation should be done before a lot of resources are allocated to the project to minimise costs caused by failure. For example, if the project was done according to the unified software development process, the evaluation would be carried out in the end of each iteration of the elaboration phase. No actual implementation should be done before the evaluation has proven the architecture suitable for the process.

Because the evaluation is usually based on non-formal architectural models, ATAM cannot precisely predict behaviour of each quality attribute. Instead, ATAM aims to identify trends by recording risks, sensitivity points and trade-off points of the architecture.

Aim of the trade-off analysis was to make structural analysis to validate the proposed design. ATAM was chosen, because team members had previous experience on it. Due to small scale and very tight schedule of the project, the ATAM process was downscaled to fit our case better. The ATAM process we applied contains the following steps:

1. Present quality goals of the system.
2. Present architecture.
3. Analyse architecture and elicit architectural approaches that address the quality goals. During this step, also trade-off points are identified

Main difference with the full ATAM process is that we present quality requirements through a simple list, which is not expanded to contain sub-attributes of each high-level attribute. For example, sub-attributes of performance are latency, throughput, capacity and modes. This is why requirements are communicated with a simple list, not with a utility tree as proposed by the standard ATAM documentation. Amount of elicited quality requirements was so small, that there was no need to prioritise them. Also, quality requirements were not specified down to the level of scenarios with stimuli and response.

2.5 Scope and validity

From the beginning we decided to limit the scope of this research. Firstly, we decided that only the requirements of FRC would be taken into account. Secondly, we decided that we would not try to implement the proof-of-concept as a production-quality system. Thirdly, we decided that only the most relevant features of the integration would be implemented in the proof-of-concept. Fourthly, we allowed ourselves to use whatever tools we would find most suitable. Finally, we decided to study only a subset of Siebel-interfaces in detail.

Because of the constraints and the nature of the project, the results are directly valid only in the context of FRC's problem domain. However, as we will demonstrate, we managed to create an integration strategy, which is not directly tied to any implementation or functionality, and thus the strategy presented in this paper can be applied to any problem where an application having it's own relational database needs to be synchronised in online fashion with Siebel. It is worth noting, that if some of the most important non-functional requirements of FRC are not present in another environment, some other strategy may be better suited.

3 Requirements

Before the actual evaluation can take place, all quality attribute requirements have to be stated. Also a clear architectural specification is needed. This section contains requirements of the actual integration to be designed. Requirements are structured according to quality attributes of software architecture. Quality attribute based requirement list is needed to analyse the resulting design with ATAM During the interview the essential quality attributes of this case were elicited. They were: performance, availability, security, modifiability, manageability, scalability, reusability, extensibility and data integrity.

3.1 Description of the functionality

FRC's organisation consists of central administration, regional offices and local branches. Each local branch has a council, which consists of nominees, e.g. chairman, vice-chairman, members, auditor(s) etc. The main functionality of the trust-registry application is to allow the local branches to browse, update, add and delete members of its council and their information. The trust-registry's database contains replicated data, the master of which is located in Siebel. Trust-registry is based on Ruby on Rails technology and data is stored into MySQL database. Ruby on Rails is a free web application framework based on Ruby programming language.

Data can also be modified directly in Siebel. This will cause possible conflicts, if the same data is simultaneously modified through the web application and through Siebel. However, 80% of the updates are at the moment done through the web interface. Most of the data is usually updated annually, usually during November and December, after annual elections of each branch. Of course there are some minor updates during the rest of the year.

During the peak-period, there are roughly 1000 data elements to be synchronised per week. Structure of the database is rather simple. It consists of four main tables, namely *branches, contacts, nominations* and *seats.* Each of them contains ten to twenty attributes.

There already is a rough ad-hoc based implementation of the integration component. It is based on Siebel EIM. EIM is Siebel's ETL-tool (Extract, transform, load) and it is used for exchanging data between databases. The current integration is based on batch jobs, which are run once a week. FRC was interested in hearing experiences and other integration options.

FRC has about 600 local branches with a total of about 4000 nominees.

3.2 Non-functional requirements

From the interview we elicited the following requirements: performance, availability, security, modifiability, manageability, scalability, reusability, extensibility and integrity. Each of these is discussed in more detail below.

Performance: As performance has not been an issue so far as the amount of data is rather small, there is no need for real-time data synchronisation between Siebel and the trust-registry. There are certain requirements for a web-service, but it is out of the scope of this project.

Availability: Siebel system is not available 24/7, which limits availability of the integration component. At the moment the desired availability level is 95%

Security: Siebel and the trust registry are hosted by 3rd parties, but at different locations from each other. Connection between them is not secure as they are not located in the same LAN. The data transferred between Siebel and trust-registry must be secured with SSH or SSL (or some other technology providing at least same security), as it contains personal information of FRC personnel. That kind of data is classified to be sensitive by the government. In the existing solution SSH based data protection has been applied.

Modifiability: FRC continuously develops new self-service systems for their field-personnel and thus the proposed solution must be easily modified and further developed.

Manageability: The servers are located in different LANs separated by firewalls. The synchronisation sequence must be initiated from the Siebel side, as it is located behind a stronger firewall, which does not allow inbound connections. The synchronisation sequence can be started manually, as an administrator must check the log of the run to solve possible conflicts. There is no need for automation, good documentation is enough.

Scalability: The amount of synchronised data will diminish in the future, as some local branches will be merged together. However, if the same integration component will be used by other applications, the amount of data will be larger.

Reusability: The same integration technique will need to be used by other projects. This will be the first project, where Siebel is integrated with a web-based application. If experiences if this integration are encouraging, the same integration strategy will be applied also by other projects.

Extensibility: All the web-based systems of FRC will be developed with Ruby on Rails frameworks in the future and thus the integration technology should fit to a Ruby based framework.

Integrity: As described before, the data can be modified at the same time through the web interface as well as through Siebel. If these modifications are in a conflict, the result of the synchronisation must not be broken. I.e. data integrity of the master-data must not be compromised.

4 Integration styles and patterns

An integration style in the context of a software design is analogous to an architectural style in buildings. The motivation behind the use of architectural styles - as well as architectural patterns and design patterns - is to promote design and code reuse. Patterns itself are proven and well-understood solutions to common problems. [9]

Design of a single integration seldom follows only one integration style. If multiple styles are followed, the system is called heterogeneous. If two styles are merged, their constraints may conflict with each other's and their topologies might totally different and thus it would be impossible to implement both of them. [2]

According to Hohpe and Woolf [5], the following decision criteria should be considered when selecting integration style: application coupling, intrusiveness, technology selection, data format, data timelines, data or functionality, remote communication and reliability

None of the integration styles can address all criteria equally well and thus selection of the applied style always causes design trade-offs. The various integration styles can be summed up as follows [5]:

- File transfer, applications produce and consume files containing data to be shared.
- Shared database, the applications store the data they need to share in a common database.
- Remote procedure invocation, have each application expose some of its procedures to other applications, which are able to invoke those procedures.
- Messaging, have each application connect to a common messaging system and exchange data and invoke behaviour using messages.

An integration pattern is not as predominant as integration style is. The scope of integration patterns is narrower, as they usually cover a couple of classes forming one independent module or component. [4]

Integration patterns describe a solution improving re-use and maintainability of modules. A negative aspect is that they usually make design more complex, which usually implies some degree of performance penalty making it harder to understand existing code and design. [2]

5 Siebel

Siebel CRM is the most well known product of Siebel Systems Inc., nowadays owned by Oracle Corporation. Siebel was founded in 1993 and it has grown very successfully: in 2002 its CRM market share was 45%. [11]

As of December 2007, the latest version of Siebel CRM is 8.0. In this work, version 7.8 was used.

5.1 Siebel integration strategies

Siebel provides a multitude of integration possibilities. Several factors have to be considered while choosing the right integration strategy for integrating Siebel with other applications. The type of integration can be data replication, data sharing or presentation layer -integration. In data replication both parties hold their own data repositories, and data is replicated between them. In data sharing data is always held in one place, either in Siebel or in the other system, and accessed from both systems. In presentation layer -integration the user interface of one application is partly or wholly integrated with the UI of the other application. [10]

Data sharing and presentation layer integration are by nature real-time integration techniques, but with data replication one of the important considerations is whether the integration should be online (real-time) or batch -based. Online integration is suitable in cases where there is a need for up-to-date information exchange and the amount of data exchanged between parties at one time is moderate, whereas batch-style integration is to be considered when large amounts of data need to be transferred at one time and/or the information doesn't necessarily need to be up-to-date. [10]

Another important decision is whether the integration style should be a loosely or a tightly coupled one. Generally loosely coupled integration techniques are to be preferred because of their greater flexibility, but in some cases, if it is known that the integration is between certain two applications only, there may be advantages in choosing a tight integration, such as greater versatility in the integration tools. [10]

One factor also influencing the choice of technologies is whether Siebel is the client or the service in the integration scenario. [10]

5.2 Integration technologies available to Siebel[1]

Siebel provides multiple ways to integrate it with other applications. To find out all the relevant options, Siebel integration manual [10] was studied. Siebel provides the following techniques: Enterprise Integration Manager (EIM), Object Interfaces for COM and Java, EAI Connectors, Web-Services, Outbound HTTP, Java Business Service, Java Connector Architecture (JCA), Application Services Interfaces (ASI), Virtual Business Components, External Business Components and ActiveX controls. This section describes each of the interfaces in more detail.

The *Enterprise Integration Manager* (EIM) is a Siebel tool for loading data from external sources to Siebel or vice versa. It can also be used to delete data from Siebel or merge data between Siebel and the other application. EIM uses special EIM tables in Siebel database to provide this functionality. The auxiliary applications are not allowed to access Siebel database directly, but the communication must be done through the EIM tables, and the EIM process is used to transfer data between the actual Siebel database tables and EIM tables. The EIM process is controlled by a configuration file, which contains instructions for the EIM process. To use EIM the EIM tables are populated and the EIM configuration file edited, after which the EIM process is run. The results can be checked from a log file. [10]

Object Interfaces are interfaces to Siebel objects that are provided for external programs. Siebel provides five different types of external object interfaces: COM Data Control, Java Data Bean, Web Client Automation Server, Mobile Web Client Automation and COM Data Server. The *COM Data Control* is a component provided by Siebel that conforms to Microsoft Component Object Model (COM) specifications. The component can be used in any application that is capable of including COM components and it provides interfaces to Siebel business objects. The COM Data Control -component communicates with Siebel Object Manager residing in Siebel server software. The Siebel Object Manager handles sessions with all COM Data Controls. [10]

The *Java Data Bean* is similar to the COM Data Control, but written in Java and thus usable from any programming environment supporting Java. *Web Client*

[1] As of December 2007

Automation Server and *Mobile Web Client Automation* provide access to Siebel user interfaces from Web and Mobile Web clients. The COM Data Server is a dynamic link library (DLL), which can be embedded in the external application. It contains Siebel application, business component and business object interfaces to access Siebel Data. [10]

Siebel EAI Connectors are pre-built connectors that can be used to access certain other systems such as SAP. These connectors use the external systems protocols such as SAP Intermediate Documents (IDOC) or Business API (BAPI) to communicate with the external system. [10]

Siebel can access business logic written Java/J2EE in three different ways. Firstly, if the external application has a *Web Service* -interface, this can be imported into Siebel, where a Business Proxy service representing the external service is created. When the proxy service is invoked the Object Manager generates a Web Service message and calls the external service. Secondly, if the external application doesn't support Web Service interfaces, but provides a proprietary protocol over HTTP, the Siebel *Outbound HTTP Transport Adapter* can be used in a similar manner. Thirdly the Siebel *Java Business Service* allows sending of messages using Java Message Service (JMS) -interface. [10]

Similarly, external Java/J2EE applications can access Siebel with a variety of ways. Siebel provides Web Services interfaces the external applications can use and JMS can also be used to send messages to Siebel. In addition, Siebel Resource Adapters are adapters based on the Java 2 Enterprise Edition (J2EE) *Connector Architecture* (JCA), which can be used to access Siebel from an application running in a J2EE application server. [10]

The Siebel business processes can use *Application Services Interfaces* (ASI) to both outbound and inbound integration. Inbound ASIs are Web Services interfaces external applications can use to launch Siebel business processes, whereas outbound ASIs are used from within Siebel business processes to call external services. Siebel workflows can be scheduled to be run at a certain time or interval. This way the business processes and their interaction with the external applications can also be used to implement batch-based integration. [10]

Virtual Business Components are as their name suggests business components in Siebel, but they are virtual in the sense that they access data that resides outside Siebel database. Virtual Business Components can use a variety of standard transports to access the outside systems data, such as MQSeries, HTTP, MSMQ and the XML Gateway Service. [10]

External Business Components are similar to Virtual Business Components, but they are regular Siebel business components that access data in an external database. External Business Components can use Siebel database connectors to access the data. [10]

ActiveX Controls can be used to capture the screen of an external application and to display it within the Siebel user interface. This integration method provides only a view of the Siebel data to the user. [10]

5.3 Style & technique selection

To choose the most appropriate technique for integrating FRC's Siebel with their trust-registry application we first examined the type of integration needed. Since both systems are based on the assumption that they have their own database, doing a data sharing type of integration would most likely involve drastic changes in either of both applications. Also technical limitations affected this decision: since Siebel cannot be trusted to be always available and it is protected by a firewall, it would be difficult for the trust-registry application to access Siebel's services directly. Siebel probably could access trust-registry's database more easily using External Business Components, but using that integration technique would mean very profound changes to the core Siebel functionality. Therefore the data sharing approach was abandoned.

Presentation layer integration would impose some technical limitations on the trust-registry application. All the means of doing presentation layer integration are very tied to Microsoft technologies (COM, ActiveX, Internet Explorer -browser), and not easily – if at all – usable from a Ruby on Rails -application. In addition this integration style would have at least as severe challenges as using Siebel data from external application with availability and security, therefore the presentation layer integration style was also abandoned.

Data replication seemed to be the logical alternative. With data replication the next question to ponder was whether the integration should be batch-based or online. The current implementation is a batch-based integration using EIM and it works well. However, some factors led us to think about implementing online integration instead. Firstly, as one of the goals of this work is to provide an integration framework that can later be reused in other applications as well, online integration would seem to be more widely reusable than batch-style integration. Secondly, using online integration, if successful, would eliminate the manual steps required in the current EIM-based solution. Thirdly, the risk of conflicting updates of data is greatly reduced (while not completely eliminated) with online integration. Finally, the user experience is somewhat improved with more up-to-date data. A further motivation was also that this work became more challenging and motivating.

The challenge with online integration is that Siebel cannot be trusted to be always on, whereas the trust-registry application is required to be. Another important consideration is the security aspect. Due to the firewall it is advisable to have Siebel as the party initiating connections to the outside world, and not vice versa. Also, the connection between the applications must be reasonably secure. Based on these requirements most of the techniques available in Siebel for real-time integration had to be abandoned. Of the Siebel object interfaces only Java Data Bean would be reasonably usable in the Ruby on Rails application, but that would require 24/7 availability from Siebel, and the connection would have to be initiated from the trust-registry application. The various types of Web Service interfaces partly suffer from the same limitations, although using only outbound calls from Siebel to trust-registry would be feasible, in which case the trust-registry application would have to cache all updates made while Siebel does not call it, and return them on the next call. That would potentially require implementing quite complex algorithms for resolving conflicts. However, the use of JMS and a message queue or an EAI platform seemed to provide quite an elegant solution. Firstly, the JMS implementation can be deployed on a 24/7 platform, thus making sure it is available whenever either system needs it. If the JMS implementation is deployed outside the Siebel firewall, all the connections between it and Siebel can be initiated from Siebel. JMS connections can be secured with SSL, which should provide strong enough encryption and parties can also be authenticated using SSL certificates. The use of JMS also solves the problem of different availability of the two parties: If Siebel is not available when the trust-registry application sends a message to it, the JMS implementation will store the message and it will be delivered when Siebel becomes available and connects to JMS queue. Therefore we decided to implement the integration based on JMS messages.

6 Analysis

The designed architecture was evaluated to make sure that it fulfils the customer requirements. General information about architecture evaluation and assessment was gathered from books [1] [2] and [3].

The actual evaluation was done by applying ATAM.ATAM is specified in [7]

6.1 Architectural approaches and patterns

Analysis of architectural styles and patterns is based on Enterprise Integration Patterns books by Hohpe and Woolf. [5]

The principal integration style is *Asynchronous messaging*. It does not require both systems to be up and ready at the same time. High frequency of messages will also reduce many inconsistency problems, which are usually biggest problem with file transfer based batch integrations. Messaging also allows applications to be de-coupled from each other.

After selecting the messaging style, we had to decide what type of message channel type to use. There are two main channel types: *publish-subscribe* and *point-to-point*. Publish-and-subscribe channels are used in one-to-many style integrations, while point-to-point channel is used in one-to-one style integrations. Our case was purely one-to-one integration, thus point-to-point channel was selected.

To make sure that messages are delivered even the messaging system fails, *Guaranteed delivery* pattern was applied. This means that the messaging system uses a data-store to persist messages. When the sender sends a message, the send operation does not complete until the message is stored in the data-store. Also, the message is not deleted from the data-store until it is successfully received the other endpoint.

Message translator pattern was applied in both ends of the messaging system to enable communication between two different system using different data formats. In our case, we needed to translate between Siebel's internal format to an XML format, which was then parsed to internal format of trust-registry. Structures of data were similar, but its representation was different between the endpoints.

Contents of messages were formed according to the *Document message* pattern. Document message passes data and lets the receiver decide what to do with it. The data is single unit of data. Another option was to use Command message patterns, which tells the receiver what to do: the sender expects something to happen on basis of the sent message. Timing is not important in context of Document message.

Content-based router pattern was applied to enable delivery of all changes in queue. In both end-points, the receiving process uses JMS message headers to decide which object type is encapsulated in the message. On basis of this information, the message passed to the correct translator component.

Both end-points also obey the *Event-driven consumer* pattern: incoming messages are consumed as soon as they become available. Other options would be *Polling* *consumer* or *Competing consumers*. There was no need for Competing consumers, as we expect that one consumer can handle all the update messages. We also want to guarantee that messages are handled strictly in the same order as they are sent.

6.2 Trade-off analysis

As a result of trade-off analysis, we discovered the following sensitivity points and possible trade-offs for each quality attribute. Also some risks were identified.

Performance

Perf-1	As performance has not been an issue so far as the amount of data is rather small, there is no need for real-time data synchronisation between Siebel and the trust-registry. There are certain requirements for the web-service, but they are out of the scope of this project.
Sensitivity points	The solution enables almost real-time based integration. Delay between database update and corresponding JMS message signalling the update to the other endpoint, is configurable. Minimum delay is 1 second. When amount of updated data increases, the minimum delay also increases.

Availability

Avail-1	Siebel system is not available 24/7, which limits availability of the integration component. At the moment desired availability level is 95%
Sensitivity points	The communication between Siebel and Trust-registry is based on persistent JMS queues, also known as *store and forward* model. Messages are stored either in memory or in a persistent storage, such as database. If either of the endpoints is down, the queue will hold all the messages until the endpoint recovers. JMS communication is asynchronous and thus the other endpoint is not aware whether or not the other endpoint is down. If the network connection between the endpoints is down, the JMS system will still hold all the messages in the queue until the network comes available again.
Trade-off	The solution needs 3rd party implementation of JMS, as it is not part the standard Java runtime environment. This increases complexity of the whole system and need of administrative work.
Risk	If Siebel is not able to contact to JMS Queues, the JMS Transport service will be shutdown and must be started again manually.

Security

Sec-1	Siebel is hosted by a different 3rd party than the trust-registry server. Connection between them is not secure as they are not located in the same LAN. The data transferred between Siebel and trust-registry must be secured with SSH or SSL (or some other technology

	providing at least similar security), as it contains personal information of FRC personnel. That kind of data is classified to be sensitive by the government. In the existing solution SSH based data protection has been applied.
Sensitivity points	JMS implementation is based on the Java Socket technology. In default mode, sockets are insecure, but they can be easily modified to apply SSL technology to secure the transport layer. The communication can be also tunnelled through a SSH tunnel.
Trade-off	Managing TCP/IP ports reserved by JMS is implementation dependent. If the JMS provider is changed, the firewall must be reconfigured.

Modifiability

Mod-1	FRC continuously develops new self-service systems for their field-personnel and thus the proposed solution must be easily modified and further developed.
Sensitivity points	The solution is based on standard messaging technology with Java-APIs. Siebel-side is based on standard Siebel features.
Trade-off	If Ruby on Rails will be FRC's preferred technology, the still need some Java-skilled people to modify and manage the integration component.

Manageability

Man-1	The servers are located in different LANs separated by firewalls. The synchronisation sequence must be initiated from the Siebel side, as it is located behind a stronger firewall, which does not allow inbound connections.
Sensitivity points	In the current configuration, only the Siebel-endpoint establishes connections. The other endpoint opens listener ports, but does not establish any outgoing connections. This configuration makes it possible to allow the firewall to allow these outgoing connections and thus enable the integration. The problem with this approach is that Siebel does not behave well, if there is a network problem when it tries to connect to the JMS queues located in the trust-registry server. To avoid this kind of problems, a dedicated message forwarder application was designed. It is a standalone Java application run on the Siebel server. Its only task is to forward messages from the local (i.e. located on the Siebel server) queues to the remote queue (located on the trust-registry server) whenever the network connection allows communication between the servers. Siebel connects to queues of the local JMS provider. If the forwarder application is able to connect to the queue in the trust-registry server, it will read the local queue and forward messages to the actual queue. Otherwise the messages will stay in the local queue until the connection can be made. JMS provides transaction management over multiple queues and thus the whole operation can

	be done inside a persistent transaction to guarantee that no message is lost. Similar sequence takes place when the trust-registry server sends a JMS message to the Siebel server. The message is sent to the JMS queue in the trust-registry server, which is listened by a stand-alone Java-application on the Siebel server. If the network connection is down, the message remains in the JMS queue until the Java application is able to read it. When Java application reads the message, it will be forwarded to the local JMS queue, which is listened by Siebel.
Trade-off	As both of the servers have their own *toSiebel* and *fromSiebel* queues, complexity of the system (and thus need of administration) increases.

Man-2	The synchronisation sequence can be started manually as the log of the run must be checked by an administrator to solve possible conflicts. There is no need for automation, good documentation is enough.
Sensitivity points	The current configuration can be run either automatically or manually.
Trade-off	None

Scalability

Scal-1	The amount of synchronised data will diminish in the future, as some local branches will be merged together. However, if the same integration component will be used by other applications, the amount of data will be larger.
Sensitivity points	The integration technology will scale very well. Average JMS system can easily handle 100 messages per second, where one message contains one data-change.
Trade-off	If size of the synchronised data item is greater than JMS messages maximum payload, the proposed solution will not work anymore. Both endpoints will need some extensive re-factoring.

Reusability

Reus-1	The same integration technique will be used by other projects. This will be the first project, where Siebel is integrated with a web-based application. If experiences if this integration are encouraging, the same component will be applied also by other projects.
Sensitivity points	The same integration technology can be applied to other applications. At the moment it does not have any trust-registry specific components. Only the Siebel-side configuration is bound to the Contact business service. The integration logic is fully separated from the application logic, the only interface between them being the database.

Trade-off	Full separation of the integration logic from the application logic makes real-time integrations impossible. Communication between logics is based on database and polling, which is never optimal. If the application logic needs to be notified after successful data synchronisation, the Java-endpoint needs to be modified. And in case of Ruby applications, implementation of such a call-back interface is difficult due to different technologies. JRuby allows Ruby applications to be run inside a Java Virtual Machine and to integration Java and Ruby applications seamlessly. It should be considered, if notification is needed.

Extensibility

Ext-1	All the web-based systems of FRC will be developed with Ruby on Rails frameworks in the future and thus the integration component should fit to a Ruby based framework.
Sensitivity points	The application end-point is implemented with Java-technology, but it can be integrated with any database based application as the integration logic is fully separated from the actual application logic. Only the database is used for communication.
Trade-offs	Some changes to database structure and application logic might be needed when the technology is applied to a new application. The database table is used for keeping track of which data elements have been synchronised

Integrity

Integ-1	As described before, the data can be modified at the same time through the web interface as well as through Siebel. If these modifications are in a conflict, the result of the synchronisation must not be broken. I.e. data integrity of the master data must not be compromised.
Sensitivity points	Conflicts are resolved by Siebel's "upsert"-service. It will decide whether or not the requested operation should be insert or update. In the Ruby side, conflicts are not solved: if transferred entity contains ID that already exists, it will be overwritten. Otherwise the entity will be added as a new element.
Risks	If the same element is updated at the same time in Siebel and in Ruby application, there is a possibility for conflict. After that, the databases are not consistent.

7 Proof-of-concept

The proof-of-concept was designed based on the findings described in previous sections. In order to be able to completely de-couple the integration components from the actual trust-registry application, we decided to implement a separate process, that polls the changes in the trust-registry's database and sends them to Siebel via a message queue implementation and another process that listens to another message queue for updates coming from Siebel and writes them to trust-registry database. There are several possible implementation techniques for this, but to keep things as simple and straightforward as possible, we chose to use an Enterprise Application Integration (EAI) platform. There are several EAI platforms available in the market, and we chose to use Sun Microsystems' Java Composite Application Platform Suite (JavaCAPS), since we already had some knowledge of it and it was easily available for this PoC. JavaCAPS has its own message queue implementation, but for the sake of the exercise we decided to use another message queue implementation, the Sun Java System Message Queue, that comes bundled with the Sun Java System Application Server. This way we were able to demonstrate that practically any JMS-compliant message queue implementation can be used for communicating with Siebel.

On the Siebel side the possibilities were a bit more limited because of the capabilities of Siebel and our knowledge of them. To receive data from the queue and update it to the Siebel Database we decided to implement a JMS Subsystem that listens to the queue and upon receiving a message starts a workflow to process it. This workflow transforms the message from XML to Siebel's internal representation and updates a business object, which will internally handle the persistence to a database. To send the changes made in Siebel to queue one has to implement a trigger that will detect the changes and start another workflow that will create a message in the XML format and write it to the queue. The actual writing is done using the JMS Business Service in Siebel, which in turn uses the JMS Subsystem to communicate with the queue. We did not implement the local message forwarder in the PoC.

The PoC environment consisted of two servers, one running the Siebel applications and another running the Siebel database, which in this case was Microsoft SQL Server. Both machines were running Microsoft Windows Server 2003 operating system. The Siebel version used was 7.8. We decided to use the database server machine to represent the server running the trust-registry application. Therefore we installed a MySQL database into it. We also installed JavaCAPS and Sun Application Server (for message queues) on the server. On the Siebel server we installed the necessary JMS libraries to connect to the message queues. Figure 2 shows the PoC environment.

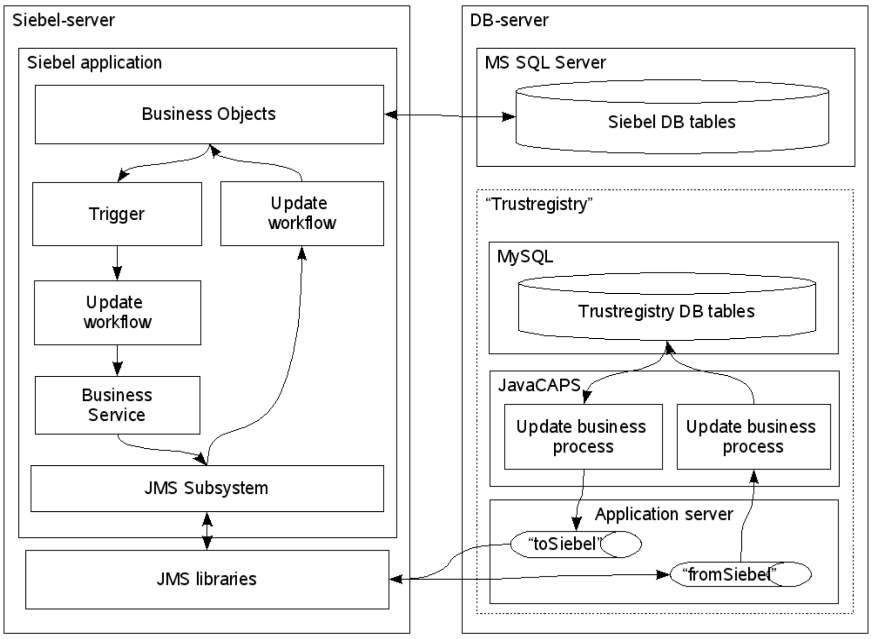

Fig. 2 PoC-environment

The processes that read data from trust-registry-database and write it to Siebel and vice versa were implemented as business processes with the eInsight Business Process Management -tool within JavaCAPS.

The process that polls the changes in the database is initiated by a scheduler component that can be configured to start a new business process instance at a given time or between given intervals. In our PoC we chose to initiate the process every 60 seconds, since we thought that this interval is sufficiently short provide almost real-time updates to Siebel, and yet it is so long, that the performance penalty on the system should be very little. Each time the business process instance is started, it reads all records from the database where the "exported on" column is null. This column already exists in trust-registry database, and it is used to signal whether new or changed data has already been exported to Siebel. The trust-registry application should set it to null when the row is updated. After reading the records they are transformed to XML message(s) and sent to Siebel. Here we had two possible implementation strategies: we could send several records in one message or each record in a separate message. Clearly, the first method is more efficient, but since the amount data and changes to it is fairly small, we thought that sending several small messages wouldn't be too inefficient in this context. Also, we were unsure of how Siebel handles messages with multiple records, so we decided to play safe, and only send one record per message. Secondly, as we chose to send only one record in each message, we could implement the business process so, that each instance handles all the records that need handling at that time and sends several messages, or so, that each instance only handles one record and sends one message, and the next instance handles the next record and so forth. The latter method is too inefficient for real-life implementation, but because it is simpler to implement and we were on a tight schedule we chose it. Thus the process is very simple and depicted in figure 3.

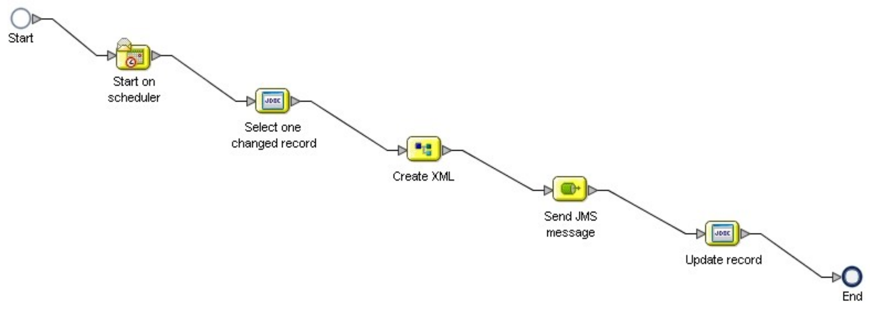

Fig. 3 The process to send data to Siebel

In the PoC we only implemented the process for updating the contacts, but similar processes for other database tables can be implemented as easily.

The process that reads data from Siebel and updates the database is also very simple. A process instance is started for each new message that appears in the queue. The process instance first checks the type of message and then queries the appropriate database table to see whether we are updating an existing record or inserting a new one. Then the update or insert is done and the process instance ends. In the update or insert the both the "imported_on" and "exported_on" fields are set to current time. If the "exported_on" field was left null, the other process would immediately send the record back to Siebel. This process is shown in figure 4.

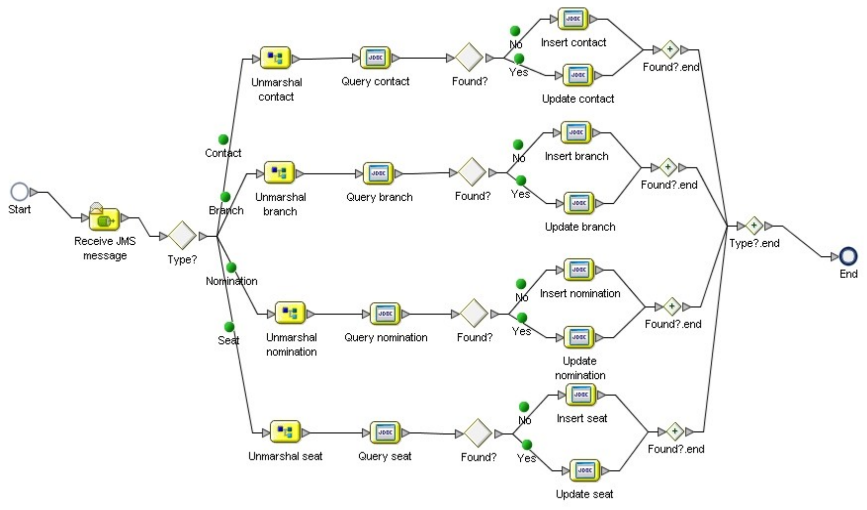

Fig. 4 The process to read data from Siebel

In the Siebel side, the design was constrained by Siebel's architecture as well as our limited knowledge of it. A significant amount of time was consumed while studying Siebel's documentation and learning Siebel's way to implement integrations and handle external interfaces.

The design is based on Siebel's EAI Java Business Service (JBS). The EAI Java Business Service is a service framework that allows custom business services to be implemented in Java and run from a Siebel application.

Even though Java Business Service can be used to implement very complex Java components and embed them inside Siebel, we needed it only to integrate Siebel with an existing JMS provider. For that, JBS provides a ready-made service component, namely EAI JMS Transport service. EAI JMS Transport service can be used to communicate with external JMS queues. Queue references are fetched through Java Naming and Directory Server, which was run along the application server.

EAI JMS Transport service can be integrated with Siebel's workflow service to invoke workflow processes and to allow workflow processes to invoke operations of EAI JMS Transport service.

For incoming messages, the JMS Transport service was configured to wait for messages from the *toSiebel* queue. When a message was received, it was automatically dispatched to workflow process named FRC-Import.

The FRC-Import workflow is shown in figure 5.

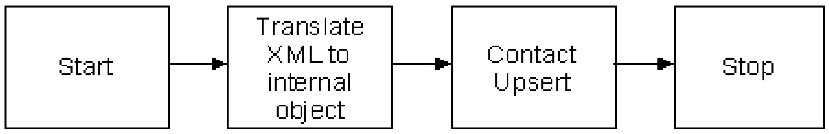

Fig. 5 Import workflow

Workflow's first task is to convert the contents of the received JMS message into an internal data representation format of Siebel. The resulting internal object is then forwarded to a dedicated Siebel business service to update or insert ("upsert") the received data into Siebel database. Siebel itself decides whether an update or insert operation is needed.

The process can be generalised by adding object type information to JMS message headers and using this information to guide the XML translator. Also the Contact Upsert step must be replaced with a generic Siebel Upsert service and guide its functionality with the same object type information.

For outgoing messages we first needed to define a Workflow policy. In Siebel terminology, a Workflow policy means a set of policies that can act as triggers to execute a workflow process. A policy consists of one or more policy conditions and a policy action. When the policy conditions are met, the policy action is executed.

A policy condition is a boolean expression. One policy can contain multiple conditions. All the conditions of the policy must be met before an action can occur. In our case, we defined a simple condition that is met when any changes to Siebel's Contact business object are done.

On basis of the defined conditions, related SQL database triggers were generated with Siebel's Generate Triggers service. These triggers will observe the actual underlying database and copy identifiers of all matching data to a dedicated work area.

Siebel itself will poll this work area can further pass all found data to the corresponding Workflow policy and start the associated policy action. The polling is done by Siebel Workflow monitor agent service, which also invokes the defined policy action.

In our case, the policy action was defined to invoke a Siebel workflow process manager service to run a dedicated FRC-Export workflow process. The workflow is described in figure 6.

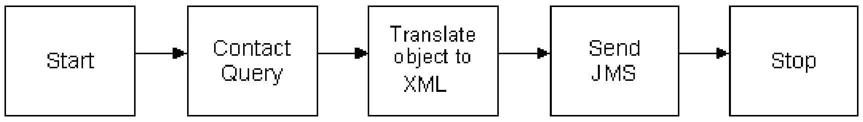

Fig. 6 Export workflow

Process receives a unique ID of the changed object as an incoming parameter. The first workflow step will query the changed object from Siebel database and further pass it to the next step, which will convert the internal object to an XML representation. The resulting XML document is then passed to Send JMS step, which uses the underlying EAI JMS Transport service to send the XML document to the *fromSiebel* queue. The FRC-Export workflow is invoked separately for each changed object, i.e. Siebel database row.

Also this process can be generalised by replacing Contact Query element with generic Siebel Query element and adding type information to the JMS message headers. The header information will be used in the other end-point to dispatch the message to correct database handler.

7.1 Outcome of the proof-of-concept

As a result of PoC, we have a very basic integration implementation which can be used to communicate Contact updates and inserts between Siebel CRM and MySQL database used by the Ruby on Rails application.

When an update or insert operation is done in Siebel, the following sequence is executed:

1. Siebel workflow monitor notices the changed data and passes its ID to the export workflow.
2. The export workflow reads the changed entity (all attributes, not only the changed ones) and translates it to XML message.
3. The resulting XML message is put inside a JMS message and sent to remote JMS queue.
4. The EAI tool running in the same environment as the trust-registry application notices the arrival of the message and starts the receive workflow.
5. The receive workflow reads the message, unmarshals the XML and updates the local MySQL database.

Similarly, when the trust-registry's local database is updated, the following sequence is executed:

1. An export workflow that is started periodically queries the local database for entries that have the “exported_on” flag set to null.
2. If such a record is found, it's contents are marshaled into an XML message.
3. The resulting XML message is sent to a JMS queue.
4. Siebel’s import workflow notices the incoming JMS message.
5. The XML message is converted into Siebel’s internal object format.
6. The resulting object is passed to “upsert” service, which decides whether there is need for an update or an insert (i.e. are we updating an existing contact or inserting a new one).

At the moment, deletion is not handled. If a contact is removed from the master database (Siebel), a notification and ID of the removed contact will be passed to the export workflow. As the first actual step of the workflow tries to query corresponding contact entity, it will fail in this case as there is no more contact with the given ID. Support for removes can be implemented by designing a dedicated export workflow and a corresponding policy agent for the delete operation. In the trust-registry deletion is not handled either. Once the row is deleted from the database it can no longer be found with the query, and therefore the deletion is not propagated to Siebel. To overcome this shortcoming the trust-registry application should be modified to handle the XML creation and sending of the messages without external scheduled workflow. Another approach that would handle deletions in both systems would be using a deletion flag in the database instead of actually deleting records. After the changing of the flag is propagated to other system, the record may be physically deleted in both systems, and the physical deletion doesn't have to propagated.

Also, the designed message forwarder application and local queue model was not implemented during the PoC project. Mapping data tables and attributes between Ruby and Siebel databases was not thoroughly designed, as we did not have adequate documentation of the trust-registry database.

If either of the Siebel workflows encounters an error, the workflow will not recover. In the case of import workflow, the JMS receiver is stopped and it must be restarted manually. In the case of export workflow, the workflow monitor is stopped. Implementation of a decent error handler would need much more Siebel knowledge than was possible to obtain during this project.

8 Conclusions and Future work

In our work we set out to answer this question: ”What are the possible means to integrate Siebel CRM and FRC’s Ruby based trust-registry and which one of them should be used for the actual integration?” In order to answer this question, we set one main objective and several sub-objectives. The main objective was to define an integration strategy for integrating the Ruby on Rails based application with Siebel. In order to satisfy this objective, the research was phased into four sub-objectives.

Our first sub-objective was to define the important quality attributes for the integration. The main quality requirements for the integration were availability, security, reusability and integrity. The trust-registry application has to be available more or less 24/7, and it's availability must not be constrained by the fact that Siebel has lesser availability requirements. There is personal data transferred between the systems, so reasonable protection must be used. Also the Siebel system is more strongly protected than the trust-registry application, and the integration must not compromise this protection. There are similar systems to be implemented in the future, so the integrations strategy must be reusable in other contexts as well. The data integrity must be maintained at all times. We found out, that the current implementation that is based on Siebel's EIM-tool satisfies these requirements quite well. The only problem was, that since the EIM batch jobs that transfer information are run manually, they are not run very often, and there can be conflicting changes in both systems that have to be solved manually. Therefore we concluded, that in order to improve current practices, we would have to improve the integrity of the solution, whilst keeping the other important qualities at least at the same level.

Our second sub-objective was exploring and describing the integration techniques available in Siebel. We based our exploration on Siebel manuals, and although we didn't make a very thorough analysis of all the technologies, we think it is safe to say that we managed to create a good overview of Siebel's capabilities, which is also documented in this report. We also conducted a cursory literature study of general integration patterns to match Siebel's capabilities to more general concepts.

The third sub-objective was to analyse the technologies and patterns available in Siebel against the documented requirements. We did the analysis rather informally to reach our integration architecture, but after formulating the strategy we did an ATAM analysis to verify it. Our conclusion was that rather than trying to improve the EIM-based integration already in place, we should implement a different strategy. The chosen strategy is based on sending messages between the systems in almost real-time every time that some data is changed. This way the integrity can be improved and manual labour eliminated still honouring the availability and security requirements and constraints.

The fourth and final of our sub-objectives was to analyse the feasibility of chosen approach by implementing a PoC. We were unable to complete the PoC as we have planned it on the given timeframe. However, the most critical parts were done, so we can confidently make some statements about the chosen integration strategy and technology.

Our conclusion is that message-based integration between a Ruby on Rails application that has a relational database and Siebel CRM is a very viable strategy in an environment, where the amount of data and changes to it is moderate, the systems have different availability characteristics, data integrity is important and there are privacy considerations regarding the data. Ruby on Rails application can use JMS directly in several different ways, but a strategy of manipulating the data store directly has the advantage of not requiring changes to the existing application. The implementation can be based on coding the logic by hand in Java, but modern EAI-tools and the features they provide shorten the implementation cycle significantly.

Future work on this subject should concentrate on building a more complete PoC implementation and analysing the different quality attributes more thoroughly by means of measurements instead of theoretical analysis. A special attention should be paid to analysing how the actual implementation copes with different kinds of conflicts.

9 References

[1] Len Bass, Paul Clements and Rick Kazman: Software Architecture in Practice, Addison Wesley, 1998, 452 pages.

[2] Jan Bosch: Design and Use of Software Architectures, Addison Wesley, 2000, 368 pages.

[3] Jan Bosch and Peter Molin: Software Architecture Design: Evaluation and Transformation, Proceedings of the 1999 IEEE Engineering of Computer Based Systems Symposium (ECBS99), Mar 1999, pp. 4-10.

[4] Erich Gamma, Richard Helm, Ralph Jonhson and John Vlissides, Design Patterns - Elements of Reusable Object-Oriented Software, Addison Wesley, 1995, 416 pages.

[5] Gregor Hohpe and Bobby Woolf: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley 2005, 736 pages.

[6] Kasanen E, Lukka K, Siitonen A: The constructive approach in management accounting research, Journal of management accounting research, Fall 1993, volume 5, pp. 243-265.

[7] Rick Kazman, Mark Klein and Paul Clements: ATAM: Method for architecture evaluation, CMU/SEI, 2000, Technical report CMU/SEI-2000-TR-004.

[8] Nenad Medvidovic and Richard N. Taylor: Separating Fact from Fiction in Software Architecture, Proceedings of the Third International Software Architecture Workshop, 1998, pp. 105-108.

[9] Robert T. Monroe, Drew Kompanek, Ralph Melton and David Garlan: Stylized Architecture, Design Patterns, and Objects, IEEE Software, Jan 1997, pp. 43-52.

[10] Siebel Systems Inc.: Overview: Siebel Enterprise Application Integration, Version 7.8, October 2005, http://download.oracle.com/docs/cd/B31104_02/books/PDF/EAI1.pdf, referenced on 9.12.2007.

[11] Wikipedia article: Siebel Systems, http://en.wikipedia.org/wiki/Siebel_Systems, referenced on 9.12.2007.