An Integration Job Engine for Everyone

Enhancing Apache Camel with
Data Mapping and Job Management

Michael Gysel & Lukas Kölbener

SEMESTER THESIS

University of Applied Sciences of Eastern Switzerland (HSR FHO)

Department of Computer Science

in Rapperswil

Supervised by Prof. Dr. Olaf Zimmermann

May 2015

Contents

Abstract

Apache Camel is a comprehensive integration framework that leverages Enterprise Integration Patterns. However, data mapping and job management capabilities are lacking in Apache Camel at present. In this semester thesis we conceptualised and implemented *Camel Jobs*, an integration job engine based on Camel that allows system integrators to build integration jobs as Camel routes. The routes are automatically enhanced with error handling, monitoring, and trigger interfaces for HTTP, JMX, and JMS. Furthermore we integrated the Groovy-based data mapper Nomin in *Camel Jobs*. Nomin mappings can be written by non-developers with ease, but still provide the power of the complete Java language.

To replace SunGard's long-established integration server for the Apex Collateral product, we designed and implemented two SunGard specific layers that reside on top of *Camel Jobs*. *Apex Connectivity* enhances *Camel Jobs* with job templates, support for the Apex integration interfaces and batch management. A layer specific for each Apex Collateral customer is used to build integration jobs and data mappings. Having been deployed into an Apache Tomcat web server or operated as a standalone Java process, the new Apex integration solution can be monitored in two ways: in custom management applications integrated via a RESTful HTTP interface or in the Web-centric systems management console hawt.io.

Camel Jobs has been released on GitHub[1] under the Apache 2.0 open source license.

[1] github.com/gysel/camel-jobs

1. Management Summary

Context

The commercial banking software Apex Collateral, developed by SunGard, needs to be in integrated into the IT environments of its customers — international banks and insurance companies.

SunGard has entrusted us, Michael Gysel and Lukas Kölbener, to replace their current integration solution in close collaboration with the Apex Collateral development team.

A Java-based integration solution has to be built serving two different user groups. Developers need to write complex integration jobs with the same product as business experts define mappings to transform data from a customer specific to the Apex Collateral data schema and vice versa.

Licensing a costly commercial data integration tool or spending a lot of time and money in developing an own integration software is not an option for SunGard as data integration is not the focus of the Apex Collateral product team. SunGard therefore aims to complete its Apex Collateral suite with a lightweight integration product which can be maintained with ease and combines available and well established open source software. SunGard's prestudy found no suitable existing solution for its needs, but identified Apache Camel as a promising basis framework.

Approach

We chose the iterative approach of Scrum to face the diverse requirements and to build a solution with the most business value. Not knowing what exactly we were going to build, the agile approach helped us to react quickly to new findings and to stay close to SunGard's needs through regular reviews.

In a complete analysis of the current Apex integration solution we listed its strengths and weaknesses. A comparison with the Enterprise Integration Patterns by Gregor Hope and Bobby Woolf[3] helped us engineer the requirements. Well defined personas guided the requirement analysis and prioritisation throughout the project to satisfy the multiple stakeholders interested in the solution.

Instead of focusing on writing a lot of code, we built multiple lightweight components connecting mature open source tools and frameworks. An extensive evaluation was necessary to find a data mapping solution to integrate with Apache Camel which was suitable for both, developers and integration experts.

Result: Apex Connectivity Server

A layered architecture with three main layers allowed us to make every component shown in Figure 1.1 as reusable as possible.

Camel Jobs is an integration job engine built on Apache Camel. This module contains job management, job triggers (HTTP, JMS, and JMX interfaces), error handling, storing of failed records, and metrics. Additionally it supports Nomin mappings, a Groovy-based mapping tool which business experts can use with ease while it empowers developers with the all the possibilities of the Java language. Support for hawt.io, a powerful web console for monitoring Camel-based applications, makes Camel Jobs a reusable integration job engine for everyone. We published it on GitHub.

Apex Connectivity builds Apex Collateral product specific integration logic on top of Camel Jobs. This includes support for the Apex integration interfaces, batch management, job templates, and default data mappings which can be reused by multiple customers.

Customer X is the main module deployed to an Apex Collateral customer environment either as a standalone process or within an Apache Tomcat application server. It contains all customer specific job definitions and data mappings. Based on Apex Connectivity, it can use or override features provided by the other modules.

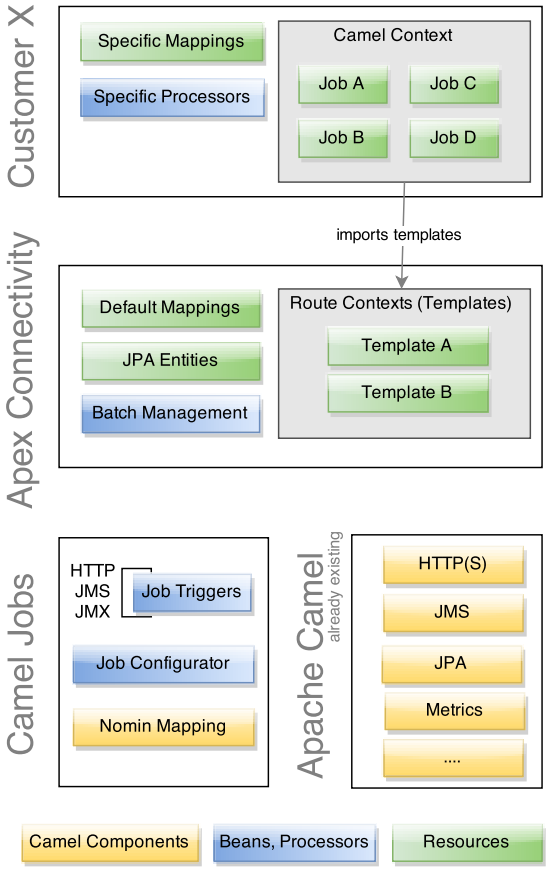


Figure 1.1: Apex Connectivity Server Architecture

Together, the three layers build an instance of the new *Apex Connectivity Server*. Two customer reference projects, Connectivity-Spirio-Bank and Connectivity-Vista-Bank, contain reference jobs which show the ability of the Apex Connectivity Server and prove that it satisfies the defined requirements.

Outlook

SunGard plans to use the Apex Connectivity Server in an integration project in the next months and Camel Jobs has been published open source as a development release. Many further development ideas like a better hawt.io integration or more powerful job configurations have been documented.

2. Project Context

Industry partner for this semester thesis is the multinational technology company SunGard. The software produced will be integrated into an existing architectural landscape. To ensure that all technological and economical decisions are aligned with the existing product called Apex Collateral, SunGard provided the specifications documented in this chapter.

2.1 Business Background

Apex Collateral is a commercial banking product developed and marketed by SunGard, a large multinational computer software company.

> "Apex Collateral is SunGard's innovative solution for collateral management, optimisation and trading on a single platform. It helps collateral traders, risk professionals, operations staff, and senior management manage and optimise their collateral on an enterprise-wide basis." [17]

Several customers from Asia, Europe and North America run Apex Collateral on their infrastructure. To integration the Java application with other related applications, some of those customers rely on the data integration provided by SunGard using a tailored Enterprise Application Integration (EAI) product.

The Apex Collateral integration solution, called EAI Server, is limited in its capabilities and restricts integration experts to only use a single, standardised work flow and a limited set of protocols. This forces the specialists working with this product to implement code that is complex and hard to read. High integration cost and a lack of flexibility are the resulting consequences.

SunGard intents to replace its integration server with a new solution based on established open source components. This product will be called **Apex Connectivity Server** and will completely replace the existing EAI Server.

2.2 Technology Environment

Apex Collateral is a proprietary product and SunGard owns all its source code. Technological aspects are — whenever possible — implemented leveraging existing open source software. Apex Collateral therefore is based on a set of popular open technologies.

- Spring Framework[53] for Dependency Injection
- Hibernate[33] for Object-Relational Mapping (ORM)
- Quartz[48] for task scheduling
- Apache Camel[10] for message translation and routing
- Apache POI[14] to create Microsoft Office documents
- Jasper Reports[36] to create PDF documents
- ActiveMQ[7] as a Java Message Service (JMS) broker

In this project, these technologies should be favoured to other products whenever possible.

2.3 Prestudy by SunGard

SunGard has conducted a prestudy and assessed a short list of four popular data integration tools prior to the start of this thesis in order to narrow down the project definition.

1. *Apache ServiceMix* [15] bundles ActiveMQ, Camel, CXF, and Karaf into an OSGi powered platform. It is entirely released under the Apache license.
2. *Red Hat Fuse* [50] is an Enterprise Service Bus (ESB) built by RedHat based on open source technologies. Fuse itself is a commercial product and may not be used in productive environments without obtaining a license. It has a limited Eclipse-based user interface to visualise Camel routes.
3. *Talend Data Integration* [55] is a commercial data integration platform and has an Eclipse-based user interface.
4. *Mule ESB* [45] is a commercial ESB tooling with comprehensive graphical management. It is based on proprietary components.

The long list contained more tools that were not assessed in detail mostly as they did not comply with the technology environment. WSO2 Carbon[58] for example was not evaluated as it is not based on Apache Camel[10] but on Apache Synapse[16].

Further options would include to use an internal SunGard platform or implement a custom solution. The exhaustive comparison of all options is attached in Appendix D.

The key takeaways are the following:

- Commercial integration products charge around 2'000$ to 25'000$ per deployment and therefore cut a significant part off the margin.
- Some vendors offer free editions. These version cannot be used as most of the existing customers would not accept them.
- There is no open source component supporting graphical data mapping. Only the commercial versions of Talend and Mule ESB do have such functionality.
- Apache Camel, CXF, ActiveMQ, and Karaf are used by all evaluated products except Mule ESB.

SunGard therefore decided to attempt to build it's own integration product and instructed the project team to evaluate and enhance existing open source libraries in order to construct an integration product using open technologies.

2.4 Apache Camel

The prestudy has shown that Apache Camel is a popular library for comparable integration software. SunGard suggested to use Apache Camel as the basis for the new solution unless the analysis finds severe reasons not to do so.

Claus Ibsen and Jonathan Anstey introduce Camel in their book *Camel in Action*[4] as following:

> "James Strachan, Rob Davies, Guillaume Nodet, and Hiram Chirino, within the open source communities of Apache ActiveMQ and Apache ServiceMix, brought the idea of Camel to life. Apache Camel is essentially an implementation of the EIP book[3], and in the summer of 2007 version 1.0 was released.
>
> Apache Camel is an integration framework whose main goal is to make integration easier. It implements many of the EIP patterns and allows you to focus on solving business problems, freeing you from the burden of plumbing. Using connectivity components has never been easier, because you do not have to implement JMS message listeners or FTP clients, deal with converting data between protocols, or mess with the raw details of HTTP requests. All of this is taken care of by Camel, which makes mediation and routing as easy as writing a few lines of Java code or XML in a Spring XML file.
>
> Apache Camel has since become very popular and today has an ever-growing community."[4, p. xxi]

2.5 Project Definition

The goals and deliverables of this thesis are specified in the project definition which was signed at the beginning of this project. A translated excerpt of the original document written in German can be found in Appendix C.

After covering the project context, we introduce the broad field of Enterprise Application Integration within SunGard in the next chapter. An analysis of the current integration solution will lead us to the requirements for the new Apex Connectivity Server which are followed by the Design and Implementation Chapter.

3. Analysis

The analysis introduces the wide area of EAI and describes the existing solution used by the Apex Collateral product.

3.1 Enterprise Application Integration

EAI is about integrating applications with different protocols and formats as well as handling remote procedure calls. In Apex Collateral, EAI is solely referred to data integration in the form of Enterprise Integration Pattern (EIP)[3, p. 147] Document Messages. Remote function calls (Command Message[3, p. 145]) are not supported by the Apex Collateral interfaces.

As EAI is a complex field, solutions vary significantly in their complexity and approach. Kay Wähner[24] has categorised integration products in three categories as illustrated in Figure 3.1.

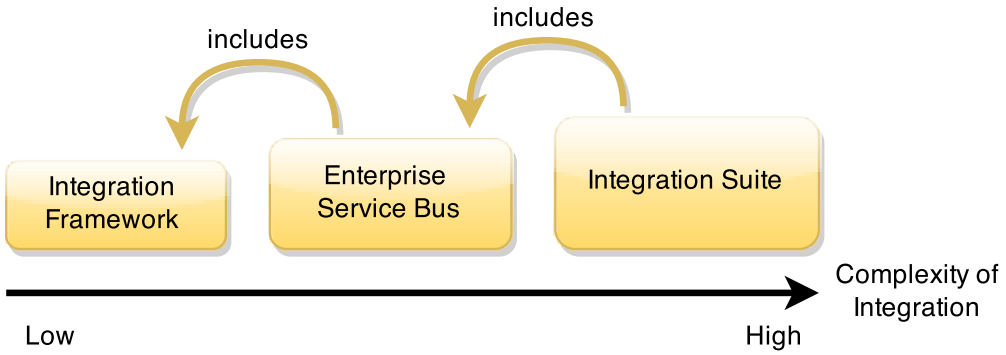


Figure 3.1: Illustration of integration framework, ESB and integration suite, taken from InfoQ[24]

We added native solutions as we have seen many integration needs solved without special products. The following category descriptions are inspired by Kay Wähner's article on InfoQ.com[24].

Native solution Data integration done without any integration tool or framework. The solution builds on a native programming language and selected libraries for data parsing and transformation.

Integration Framework An integration framework is a component developed to integrate applications with different protocols and technologies. It supports the developer with automatic parsing and conversion as well as concepts such as endpoints, producer and consumer and the Enterprise Integration Patterns[3]. Apache Camel[10] is a popular example of an integration framework.

Enterprise Service Bus An ESB has the same core functionality as an integration framework but is enhanced with remote procedure calls and tools for deployment, administration, monitoring, and data transformation. The engineer using an ESB does not need to be a skilled programmer. Many vendors offer commercial support for their products.

Integration Suite Pure integration functionality as provided by an ESB is combined with business process management, business activity monitoring, master data management and more high level integration topics.

An ESB and an integration suite both support remote procedure calls and complex tooling for job orchestration. As Apex Collateral only needs to integrate data, the required solution is categorised as an integration framework.

3.1.1 Industry Examples of Data Integration Solutions

We interviewed architects of different applications and companies to identify what kind of data integration requirements they have and which approach they use to meet their requirements. Table 3.1 shows an overview of these interviews.

#	Company	System	Integration needs	Integration approach
1	Swisscom AG	Credit Risk Product	Integration of heterogeneous data with two different core banking system	Independent Web service implementation of interfaces, no EAI tools
2	Swisscom AG	Credit Risk Product	Periodical integration of industry statistics for calculations	File upload and parsing with native Java and an Microsoft Excel Application Program Interface (API)
3	Swisscom AG	Credit Risk Product	Import of customer specific calculation parameters	A well defined workflow with a comma-separated values (CSV) file uploaded to an File Transfer Protocol (FTP) Server, processed and transformed to eXtensible Markup Language (XML) by a Shellscript and forwarded to a webservice which writes the data into a database
4	SCS AG	Public Transport Ticket Distribution System	Import of tariff data from different organisations in several formats	Independent parser and translator for every organisation written in native Java
5	SCS AG	Public Transport Customer Information System	Scheduled or realtime import of heterogeneous data from multiple organisations. Complex data routing and transformations as well as multiple output channels are required	Native Java solution with open source APIs for accessing different endpoints

Table 3.1: Result of data integration interviews

Surprisingly, the solutions described in the interviews do not contain any usage of integration products. The interviewed architects gave revealing arguments for their approaches:

SCS AG architect referring to #5 "Java is a powerful and well known technology for mapping and transforming data. An integration framework like Apache Camel would impose a way of modelling and implementing work flows upon the developer.

This might have a positive impact for structuring, but we do not see real advantages that would compensate the complexity introduced by an extra layer."

Swisscom architect referring to #1 "The data mapping and routing for our interfaces to the core systems have a complexity which would justify an integration framework. Nevertheless, the implementation of those interfaces has been done once for each core system and is not changing frequently. So we do not see a need to be more flexible in this area."

The Swisscom AG architect has identified a potential use case for an integration framework in #3:

Swisscom architect referring to #3 "Our solution that integrates the customer specific calculation parameters has raised many issues. Customers have problems to provide their data in the requested CSV format. The shell script transformer is easy to deploy but difficult to maintain. An integration framework could simplify the implementation and provide higher flexibility for our customers. As we implement only one workflow, the solution should be deployable standalone or in an Apache Tomcat web server. Extra infrastructure for this integration workflow is not acceptable."

The given statements raise the question why SunGard looks for a new integration framework whilst the interviewed architects seem to favour a native Java solution for their data integrations.

We have identified the following conditions that indicate a need for an EAI solution with an integration framework:

- Multiple heterogeneous interfaces are to be implemented to successfully integrate an application with its peripheral systems.
- The application has to be integrated for multiple customer environments which require different implementations of the interfaces.
- The data mapping and transformation requires deep understanding of the business domain. The integration work is therefore not done by software engineers but business analysts with varying knowledge of programming.

3.2 Existing Apex Collateral Integration

This section describes the current Apex Collateral integration product called EAI Server and defines the scope of this thesis in more detail.

3.2.1 Overview

Figure 3.2 shows the system context diagram of the current Apex EAI solution.

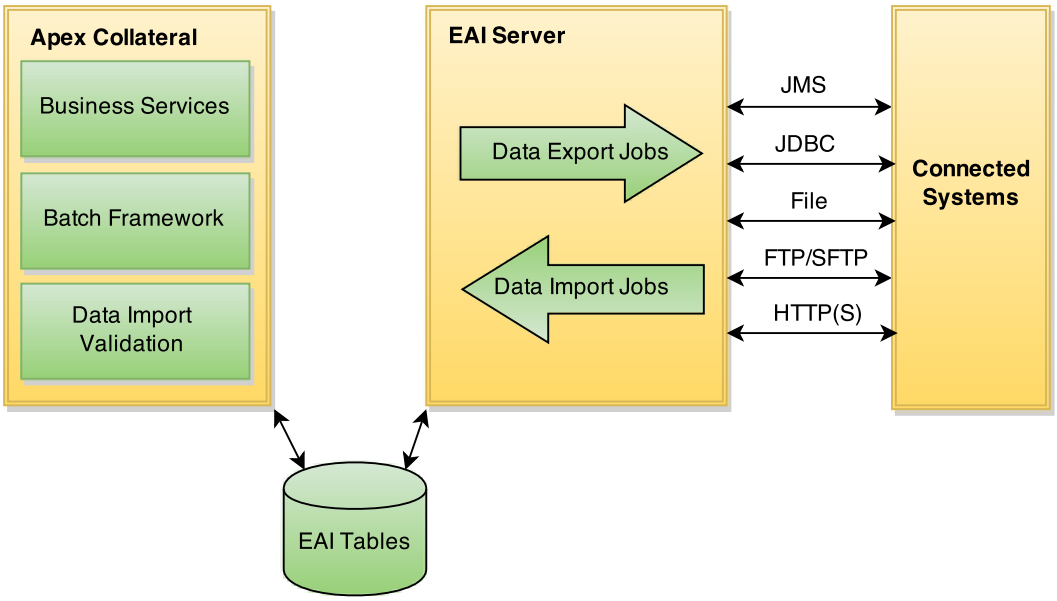


Figure 3.2: System Context Diagram

The existing environment consists of four components:

- The **Apex Collateral** core application connects to the EAI tables and ensures data consistency. All records not conforming with the expected data format are rejected.
- The **EAI Tables** which are part of an Apex Collateral release and define the foundation of the integration process. This component represents a Shared Database as described by Martin Fowler[3, p. 47].
- The **EAI Server** connects the EAI Tables with customer specific systems and resources.
- The **connected systems** are given by the customer environment. These interfaces vary for every Apex Collateral installation.

On the Apex Collateral side, the core application imports and exports data from the EAI Tables. On the customer side, the EAI Tables are accessed by a component providing data transformation and routing. Larger customers tend to use their existing EAI solution like a sophisticated ESB. Smaller customers often entrust SunGard with the task to integrate Apex Collateral into their environment. In these cases, SunGard uses its own solution called *EAI Server*.

Martin Fowler recommends not to use a Shared Database and use Remote Procedure Invocation or Messaging instead[3, p. 49]. This is design structure however is given in the scope of this thesis and will not be changed as part of the solution design.

Transactions are used while importing data from the EAI Tables into the core Apex Collateral database. One import or export consisting of one or many records is executed in a transaction. When one job fails, the transaction is rolled back and the error code of these particular records are set to *failed* in a new transaction. High volume jobs can be configured to perform intermediate commits in order to prevent high memory consumption.

The machine running the EAI Server is secured on an operating system level and only entitled users may connect to it. Other than this no special precautionary measures are taken. There is no authentication or authorisation concept implemented in the EAI Server.

3.2.2 Scope of this Thesis

The mission of this semester thesis is to conceptualise and implement a solution that is more flexible and powerful than the EAI Server. The other integration relevant components like the EAI Tables and the Apex Collateral core are not scope of this thesis.

3.2.3 Stakeholders

There are several stakeholders having interests in the Apex Collateral EAI process:

- **SunGard Professional Services (PS) Team**: Employees implementing and configuring the EAI Server for new or existing customers. PS members have a well developed domain and business knowledge but often lack sophisticated programming skills. Integration projects tend to have tight time schedules and limited resources.
- **SunGard Developer Team**: Engineers implementing the Apex Collateral system as well as the EAI Server. Developers are highly sophisticated software engineers but lack in-depth domain knowledge. These specialists are called if a requirement like a complex data mapping can not be met by the existing EAI Server and needs to be implemented with a module written in Java. It has recently happened that developers helped out in the PS Team to mitigate resource shortages.
- **Apex Collateral Customer**: The customer has a special interest in the EAI Server as it may result in constraints on how to provide and receive business data. Additionally a customer might have special non-functional requirements like security constraints or monitoring demands on the EAI Server and its interfaces.

3.2.4 The EAI Server

For every client integrating Apex Collateral in their environment, a set of integration jobs are defined. A job can either run in realtime[1] mode or is triggered by another system. When running a job in realtime mode, it polls the data source for new records in a configurable time interval and processes those records immediately as opposed to the batch mode in which the job needs to be explicitly triggered by a job trigger and processes the data as a logical set of records.

Structure of a job

Figure 3.3 outlines the main components of an integration job within the EAI Server:

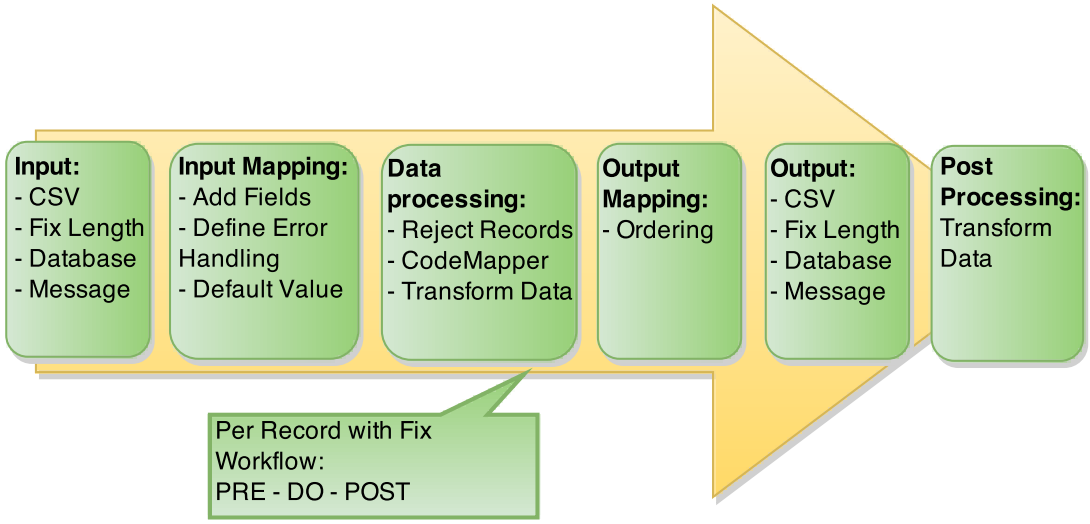


Figure 3.3: Integration Job in the EAI Server

Input/Output Modules In and out modules are adapters which provide a key-value interface for different formats and protocols like CSV files, databases or message queues.

Data Mapping In and out attributes can conveniently be mapped to each other with a CSV mapping file. Out attributes with default values can be introduced without having a corresponding input attribute.

Data Processors are called once for every input record and follow a fixed flow: *In - Process - Out*. Multiple processors can be chained after each other. They are mostly used to modify the data during an integration job.

[1] Ad hoc is used instead of realtime later in this document as realtime is technically not precise.

Post Processors are used to finalise data processing after all data has been stored in the defined output sink.

For error handling, a job creates a file which consists of all records that produced an error or had been selectively rejected. Exceptions are forwarded to the EAI Tables or can be investigated in the EAI Server log files.

3.2.5 EAI Server Deficiencies

The current solution contains flaws so that some requirements are not satisfied.

Static Workflow

Data processors have a fixed workflow (*In - Process - Out*) and are stateless. This design implies that mappings dependent on multiple records are not supported. Workarounds overcoming this restriction tend to break or ignore the underlying architecture. Global intermediate data stores, integrating logic in out modules and direct analysis on the raw input data are some of the consequences.

The limitation of having only one out module led to the design of composed jobs like the one described in Figure 3.4.

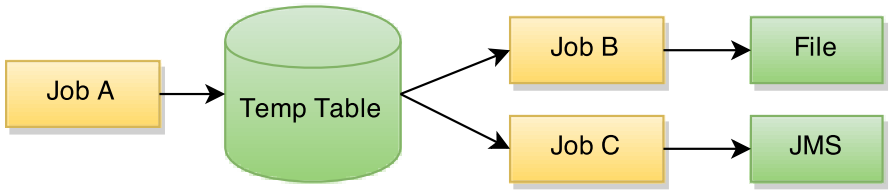


Figure 3.4: Three jobs and a temporary table are required to write the data to two different targets

This requirement matches the EIP *Recipient List* [3, p. 249].

> **How do we route a message to a list of dynamically specified recipients?** Define a channel for each recipient. Then use a Recipient List to inspect an incoming message, determine the list of desired recipients, and forward the message to all channels associated with the recipients in the list. [3, p. 249]

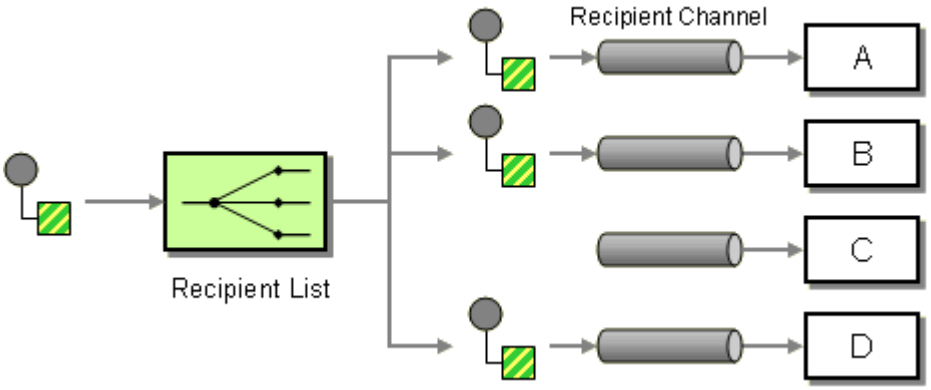


Figure 3.5: Illustration of the pattern "Recipient List"

Testability

Input and output data is stored and forwarded in object arrays visible for all components of a job. Processors and modules are therefore highly coupled and very hard to test in isolation. The current integration jobs are not covered with automatic test suites, which makes it difficult to ensure correct behavior during release upgrades or changes in downstream interfaces.

Complex Mappings

The mapping mechanism works well for what it is designed for. The PS team can configure in and out attributes with their corresponding data formats. It is additionally possible to define default values and code mappings. The latter are pre-defined mappings from one value to another. As an example, a value *3* of the attribute *trade type* is always replaced by the a value *secure_trade*.

Nevertheless, more complex mappings are not possible and therefore need to be implemented using data processors written in Java. A complex mapping may consist of calculations or conditional mappings.

Two Examples showcase mappings that cannot be implemented using the current data mapper:

- *TRADE_DATE* of record *A* must be smaller than the corresponding value of record *B* for *A* to be integrated.
- The value of *PRICE* needs to be divided by 100.

Usability and Simplicity

The current solution is not well documented and not self describing enough to be used and understood by an average professional services member. Furthermore most integration projects have very tight schedules and limited resources. This results in code duplication and therefore bad maintainability of the integration jobs.

Core and Customer Specific Code

The architecture of the EAI Server clearly separates core and customer specific code. Nevertheless, due to the reasons described in the section above, this separation is often broken due to the pressure of an integration project. The result is copied core code in customer specific projects as well as customer specific code within the core code.

Fault Tolerance

The EAI Server does not have a robust error handling and is not able to handle malicious messages properly. Therefore a malicious message may take an interface completely down as the server tries to reprocess this message for an indefinite number of times. An integration server should recover from invalid data formats, temporarily unavailable data sources or unexpected message contents. A malicious message should be preserved for manual review and the server should continue to process data from its interfaces. Some sources or targets already have support for rejected messages. However this needs to be facilitated by all possible protocols and formats.

Missing Adapters

The current solution supports CSV, Fix Length files and Key/Value formats which can be accessed by direct file access, (S)FTP, Message Queues or Database Connections. Support for Web services, XML or JavaScript Object Notation (JSON) are missing. These missing adapters result in restrictions on the customer's data interfaces.

Deployment

The EAI Server is currently a standalone Java application. Deployments into a Java application server or an OSGi container are not possible.

The analysis of the EAI Server and especially identifying its flaws helped us to collect and define requirements for the new Apex Connectivity Server. These requirements are document in the following Chapter.

4. Requirements

These chapters describes the (non-)functional requirements of the target solutions and the target users working with the end product.

The requirements in this chapter are not prioritised. As part of the sprint planning meetings, these requirements are transformed and prioritised to user stories and tasks. The description in this chapter is meant to provide a high level overview and establish a common sense between all stakeholders of this project.

4.1 Personas

Even though an integration product is not a piece of software with users in a traditional sense, there is a set of different people exposed to user experience questions throughout the life cycle of the software.

Our personas are based on interviews with employees of SunGard that were part of integration projects and have hands on experience in implementing interfaces between software systems.

4.1.1 Characterisation Elements

Personas should cover the following areas.

Role describes the persons role within an integration project and the characteristics commonly identifying such a person.

Programming skills outlines how much experience the person has with writing code and which languages he is familiar with.

Business domain knowledge should summarise the subjects background on the business domain. This includes his familiarity with business terms, calculations and data standards.

Integration product experience describes whether this person is exposed to the integration product for the first time or whether he has been exposed to the product before.

Main goal identifies the main objective the person tries to achieve.

4.1.2 Frank Frontline

Frank is an integration expert. He is part of the team which is primarily focusing on rolling out an already built software package in customer specific environment.

He is 35 years old and lives out of his suitcase. Every month he is working in a different country of the world, following the trail of the successful sales team.

Using stackoverflow.com and similar sources he solves problems by writing his own code. But his main area of expertise is his detailed knowledge about the products he has already integrated in various customer environments.

His experience is based on working with multiple clients in very different environments. This allows him to efficiently translate customer requirements into product functionality and bridge the communication gap between the development team and the customers very specific needs.

Frank has a broad knowledge about the product and the domain model used for integration. Not because he has a deep understanding of its architecture, but he knows every flaw or problem that ever occurred within the integration projects.

The primary goal of his assignment is to finish the integration project on time.

Alex Allrounder

There are two type of Franks. The one described above has very basic programming skills. Alex Allrounder however is just like Frank, but started his career in a technical context rather than a business context and can therefore handle much more complex technical challenges. Alex is familiar with software engineering topics like object-oriented programming, unit testing, design patterns and has a basic understanding for software architecture.

4.1.3 Paul Perfectionist

Paul is a developer. Most of his time he works on new features whilst ensuring a high quality code base. He detects code duplication, missing test cases and inconsistent naming of components. As he has been maintaining parts of the integration product, he knows its concepts and architecture quite well.

From time to time it happens that the integration team lacks resources and a developer has to help out. Paul hardly travels to visit the customers but he talks to the integration expert and takes specific tasks. Complex integration tasks that require sophisticated modules are assigned to developers even more often.

Paul has a major in computer science, has been writing code since many years and knows several programming languages. He always chooses the right tool to solve problems. Since he has been working on the same product for several years, he has a good understanding of the underlying business domain. Nevertheless he sometimes has to validate his solutions with a business analyst or an integration expert when facing detailed domain questions.

His expertise is primarily the product he is working on and he is not very familiar with alternate terminologies or different approaches on solving the same problems. He therefore struggles sometimes when being exposed to customers directly.

Paul's main objective is to keep a clean architecture resulting in good maintainability within all software components.

4.1.4 Stephen Steward

Stephen is an operations manager. His goal is to keep the software and all interfaces up and running.

Unlike Frank and Paul he is not employed by the software producing company but he is part of the customer team. Every morning he ensures that all batch jobs finished successfully and otherwise troubleshoots them. He works in a close collaboration with the vendor support team to file bug reports and coordinate software updates.

He possess some knowledge about the business domain but his main strength is his detailed familiarity with all infrastructure aspects. To him, the integration product is an application like many others in his environment. His primary goal is to identify errors, performance problems or resource shortages as quickly as possible. In case of an error he sometimes needs to answer business related questions from product managers about data errors or losses. Stephen therefore demands good technical and business related monitoring interfaces provided by the applications he observes.

He is not a programmer but uses shell scripts to automate tasks on a regular basis. He is only familiar with the integration product that is currently in place but knows exactly how it works.

4.1.5 Doug Doityourself

Doug is an integration expert with a similar background and programming skills like Frank Frontline. Unlike Frank is he is not part of the product team but an employee of the customer company.

Doug knows the customer interfaces and environments well and has integrated a large number of applications. He knows the connected applications with their requirements, capabilities, interfaces, and flexibilities. Doug has acquired a deep knowledge about the

product's data model and its interfaces as he has been working in this environment for several years.

Doug's goal is to integrate the product in his company environment with as little help by the product team as possible. Like this he avoids expensive and time-consuming change requests. He therefore requires well documented and easy to use interfaces. Doug loves standardised solutions as they make his life much easier. For interfaces defined by industry standards he expects predefined jobs which can be enabled and configured quickly.

4.1.6 Charles Counselor

Charles is an external consultant and is added to a project when there is an urgent need for a specialist and his position cannot be filled with current employees.

He is an expert on the business domain and has worked with many customers as well as most of the competitors. Charles supports the professional services team in integrating the product into different customer environments. He therefore needs to define integration jobs whilst he - compared to Frank Frontline - lacks knowledge about the EAI Tables model.

His primary goal is to satisfy his client, the product management and Frank with the work he's doing.

Charles Counselor is a person not yet existing in integration projects. Therefore his persona is not described in detail. The product management is planing to engage Charles to prevent resource shortages.

4.2 Functional Requirements

The Apex Connectivity Server faces the functional requirements presented in the following section.

Some requirements are documented as user stories whilst others are not as we found user stories only to be useful when there is an actual user role requesting this feature. Solely technical requirements like job triggers or protocols are therefore not documented as user stories but as bulleted lists, tables or using their textual representation.

4.2.1 Data Mapper

As Frank Frontline, I want to use an easy to use data mapper so that I can write or adjust mapping logic without writing Java code.

> As Paul Perfectionist, I want to use a powerful data mapper so that I can handle all mapping scenarios within the data mapper using configuration or a scripting language.

Frank Frontline would mostly use basic features of the data mapper. Peter Perfectionist will use a advanced features as well.

Basic Features

- Key mappings
  The source attribute 'Price' maps to the target attribute 'Total'
- Type conversions
  Convert the string '10.10.1973' to a java.util.Date object
- Calculations on values
  Divide the value of 'Price' by 100 during the mapping
- Defaulting of empty or inexistent values
  Insert the current date to the target attribute 'Created'
- Code mappings
  If the value is '3' then replace it with 'Security'

Advanced Features

- Conditional calculations
  If the value of 'Currency' is EUR, multiply the value 'Amount' with 1.20
- Multi-record conditions or instructions
  If the 'Date_Created' value of record A is before 2014, set 'Archive' value of record A and B to 'True'
- Conditional mappings
  If the 'Type' value is 'Partial', map the value 'Part_Price' to 'Price', otherwise map 'Total_Price' to 'Price'
- Extensible mappings
  The option to write mappings of any complexity with a programming language like Java
- Nested mappings
  Nested data strcutures like a key-value map within a key-value map can be mapped

4.2.2 Integration Workflow

> As Paul Perfectionist, I want to create integration jobs based on patterns so that I can reuse processing strategies.

The new Apex Connectivity Server needs to support the following EIP:

- Point to Point Channel
  Read and write from one source to one target
- Recipient List
  Write to multiple targets
- Aggregator
  Combine individual but related records
- Splitter
  Split one record into several records
- Message Filter
  Filter records based on static criteria
- Content-Based Router
  Route records to different targets based on the record content

Publish-Subscribe Channel has not been mentioned in the requirement workshops. However as it is such an important pattern in message based environments, it might also be required at some point.

4.2.3 Protocols and Data Formats

> As Frank Frontline, I want to read and write data from standardised protocols and formats so that I can integrate as many data sinks as possible.

Interviews with SunGard employees have shown that the protocols and data formats listed in Table 4.1 need to be supported for input and output interfaces.

Format \ Protocol	JMS	File	JDBC	FTP/SFTP	HTTP(S)
CSV	x	x		x	x
Fixed Length	x	x		x	x
XML	x	x	x	x	x
SWIFT	x	x	x	x	x
JSON	x	x		x	x
SOAP					x
Key/Value			x		

Table 4.1: Table of required (x) protocol and data formats

4.2.4 Job Templates

> As Frank Frontline, I want to reuse a job template when implementing a job that adheres to a predefined standard so that I need less time to implement it and avoid code duplication.

Some jobs are almost identical in all customer environments whereas other jobs vary significantly depending on the context. Job templates should be provided that allow reuse of workflow and mapping logic. A job template should provide a complete job definition and data mapping that can be imported into a customer specific configuration. Frank Frontline should then be able to optionally override the data mapping provided by the job template to adjust to customer specific interface definitions.

Job templates should be versioned to support interface changes as part of major release upgrades of Apex Collateral.

4.2.5 Job Execution

Integration jobs by the new Apex Connectivity Server need to be launched by two kind of triggers:

External triggers are used when a job needs to run whenever new data is available, which is when data must be delivered or requested promptly or when an integration job is part of a broader workflow controlled by the customer company. The following external triggers to start a job are required:

- A JMS message sent to a message queue by the Apex Connectivity Server.
- A RESTful HTTP call sent to a REST interface provided by the Apex Connectivity Server.

- A created or updated database record in the EAI Tables or downstream interfaces.
- A new file in a local or remote folder matching a specific file name pattern.

The Apex Connectivity Server should continuously be listening on those channels for new trigger objects. A runtime trigger has to provide feedback to the caller and return the following information:

- Return code indicating whether the job succeeded or failed.
- Volume statistics with number of imported, failed, rejected, and ignored records.
- Time duration needed to process the job.

Depending on the trigger, this information may be returned through a different channel. For example, a job triggered by a JMS message may return the result through a different channel using the request-reply pattern shown in Figure 4.1

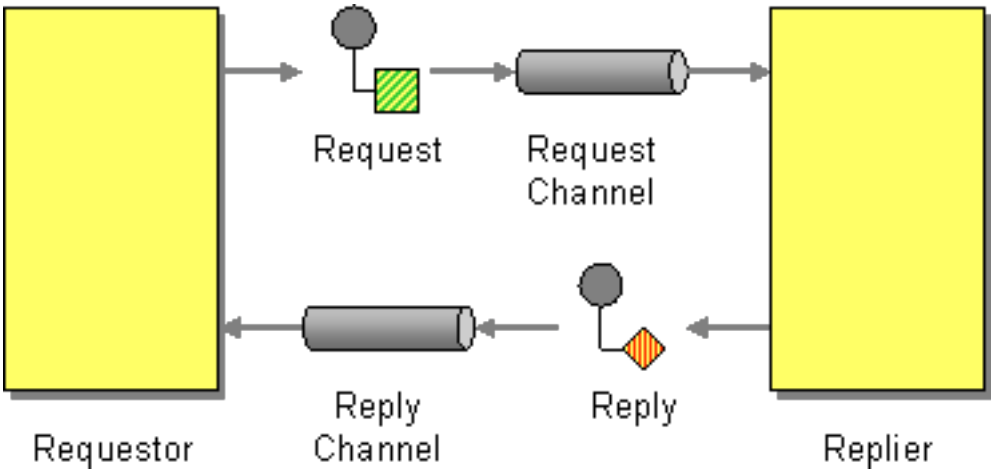


Figure 4.1: Request Reply pattern, taken from the EIP book [3, p. 154]

An ad hoc trigger represents an Event-Driven Consumer[3, p. 498].

Scheduled triggers are supported by a scheduling infrastructure and start jobs when specific points in time are reached. There are different ways to define a point in time:

- A single date and time instance.
- A date and time instance with a reoccurring object, defining in which intervals or on which specifics (like a weekday) points in time are reached.
- A timer, firing after a predefined amount of time has passed. Optionally the timer can restart n times.

4.2.6 Batch Processing

The EAI Tables and Apex Collateral core impose a batch processing concept which needs to be followed.

Jobs in batch mode are used to process data files that are provided on a regular basis. Data volumes in these jobs are usually large. Within the EAI Tables, records processed in batch mode are linked to a set using a value in the column `EAI_BATCH_ID` referring to a record in the table `EAI_BATCH`. Jobs in batch mode should support intermediate database commits to reduce memory usage and optimise performance as well as parallel processing to handle large volumes.

Jobs in ad hoc mode [1] are triggered by new files or database records and import usually small sets of data. The records in this mode are identifiable by the value `NULL` in `EAI_BATCH_ID`.

4.2.7 Delta Builder

In the old EAI Server, some jobs are implemented using a delta builder to skip the import of all rows that have not been changed compared to the last import. The new delta builder should be able to read data from Apex Collateral core tables and compare it with the data being imported to determine whether the current record differs from the present data. All records which contain data already available in the system should be skipped during the import.

4.3 Non-Functional Requirements

The following non-functional requirements should be satisfied by the new Apex Connectivity Server.

4.3.1 Usability

Referring to the personas 4.1 and stakeholders 3.2.3, some end users of the Apex Connectivity Server do not necessarily possess sophisticated programming skills and are more focused on business knowledge and project schedule. The new solution therefore needs to be easy to understand and use even for non-developers. Integration jobs are to be defined by configuration and should only need Java code extensions for uncommon and complex use cases. To ensure the right level of abstraction, usability tests with members of the professional services team need to be done when evaluating possible solutions.

[1] SunGard refers to this as realtime mode. Realtime however is guaranteeing strict timing contstraints which is not the case here. We therefore decided to call them ad hoc jobs.

- Frank Frontline 4.1.2 should be able to implement a data mapping for a job in an hour.
- Alex Allrounder 4.1.2 should be able to implement a sophisticated workflow in a day.

4.3.2 Simplicity

Visibility and simplicity of Apex Connectivity features must be guaranteed. Some features of the current solution have been ignored and reimplemented because users have not been aware of them.

Within the usability test described in Section 4.3.1, all of the existing features provided by the new solution should be used if suited. Workarounds for requirements covered by the solution show an incomplete application of this requirement.

4.3.3 Information Security

Outgoing connections need to support HTTPS and Secure File Transfer Protocol (SFTP) as well as encrypted and authenticated JMS messaging.

Authentication and authorisation to schedule jobs is delegated to the underlying operating system of the server and does not need to be implemented in Apex Connectivity.

4.3.4 Separation of Concerns

Developers with Paul Perfectionist as a representative (see 4.1.3) and Professional Service members like Frank Frontline (see 4.1.2) are the stakeholders working with the Apex Connectivity Server on a regular basis. As they have different technical backgrounds, they also handle very different tasks. Developers might enhance the Apex Connectivity code with new features, fix bugs or write extensions for customer specific integration projects when complex mappings or routings are required. PS members define integration jobs or configure job templates for every single customer to be integrated.
This separation has to be incorporated in the software architecture of Apex Connectivity. Core functionalities and customer specific mappings or extensions need to be clearly separated. Changes to Apex Connectivity code therefore needs to be prohibited for integration experts or at least validated by a developer responsible for the server. Components not intended to be changed by an integration expert should therefore not be visible for him.

4.3.5 Open Closed Principle

> Software entities should be open for extension, but closed for modification.
>
> Bertrand Meyer [5, p. 57-61]

Apex Connectivity needs to be open for extensions while it stays closed for unnecessary workarounds as it was common in the old solution (see 3.2.5).
Open for extension means that at least developers have options to extend workflows and data mappings by using native Java code when available features do not support customer specific requirements. Special attention has to be paid if a third party software for data mapping is used.
Closed for workarounds requires a prevention procedure that even within the pressure of a integration projects hacks are introduced into integration jobs. Hacks are defined as implementations which clearly violate the products software architecture or harm its functionalities like testability or information security.

It is difficult to measure if this requirement is satisfied and a real test requires a real integration project which is out of scope for this thesis. We therefore recommend a review process where complex solutions which do not only consist of simple configurations and mappings are reviewed by a person in charge of the Apex Connectivity Server.

4.3.6 Testability

Connecting different system implies a tight coupling to the interfaces of those systems. Both sides, the Apex Collateral core and the connected systems on the customer's side might change behaviour, refuse connections or provide corrupt data. To handle such an error prone environment, integration jobs need support by automatic test suites to ensure their correct behaviour. This is especially necessary for release upgrades of the Apex Collateral product. Test suites should be able to cover correct work flow, mapping logic as well as failure scenarios.

The following layers of tests are required:

1. Unit tests to verify code in isolation.
2. Integration tests to verify the interaction of components.
3. Acceptance tests to verify behaviour of complete integration jobs using test scenarios.

4.3.7 Deployment

Different deployment models are possible for the new integration solution. It should be runnable as a standalone Java application but also support at least one other deployment

model. Currently discussed options are deployments into a Java application server or an OSGi container. Whatever container is chosen, it must not interfere with the requirement of simple installations.

Customer environments are diverse. It should therefore also be possible to support several deployment options.

4.3.8 Performance and Throughput

Some of the interfaces need to process volumes with up to several million records. Those jobs should be run in a reasonable time frame and without running into memory limits. Therefore stream oriented and parallel processing needs to be supported. SunGard expects that the new integration server supports the following operation numbers:

- 50 records per import as the lower limit.
- 3'000'000 records per import as the upper limit.
- 10 to 50 different interface jobs in a typical installation.
- All of those imports should run in batch mode, roughly 30% of the imports should also support an ad hoc mode that polls for new records and imports them automatically.
- 15 seconds (lower limit) to 15 minutes (upper limit) for an import with 1'000'000 records.

4.3.9 Monitoring

The results code of job executions and occurred errors need to be accessible through an API and, if possible, through a user interface. The status of such jobs is being monitored on a daily basis by an Operations manager like Stephen Steward (see 4.1.4).

The following information should be available to an observer:

- List of jobs
- Last execution times per job
- Number of imported, skipped and rejected rows per run
- Errors and log messages of imports

4.3.10 Availability and Fault Tolerance

To ensure good availability, Apex Connectivity must implement a well planned fault tolerance. These following Patterns by Robert Hanmer [2] help keeping the server running even in a case of an error:

- Integration jobs implement a *Unit of Mitigation.* As a result, an error or exception in an integration job does not affect other jobs.

- Malicious records are rejected and handled properly. The system performs a *Roll-Forward* to resume normal execution. The handling of failed records with the use of a *Quarantine* should be possible and configurable on job level.
- Errors, failed records or complete failures of the server should be reported to a *Fault Observer*. See the Section 4.3.9 Monitoring for more detail.

To guarantee correct implementation, test cases for the above scenarios should run accordingly.

4.3.11 Maintainability

Apex Collateral is a product which is strategically relevant to SunGard and plays an important role to financial corporations all around the globe. The components used to build Apex Connectivity must therefore ensure good maintainability.
SunGard generally prefers third party libraries and products over custom implementations to meet the defined requirements. Third party solutions need to have either a broad community support or a long term product commitment by its vendor and an approved Open Source license. Accepted licenses are the Apache license, MIT license, LGPL and the BSD license. Probibited is the GPL. Other licenses need to be validated with the SunGard legal team. Community versions of commercial products are not admitted as customers generally do not accept them.
Custom implementations should be introduced only if requirements cannot be met by suitable third party solutions. If still introduced, design decisions and code documentation have to been reviewed by SunGard developers to ensure maintainability beyond the scope of this semester thesis.

4.3.12 Operations

Relevant job configurations should not be part of the Java jar file so that they can be modified without rebuilding the delivery. This requirement includes:

- Changes in configuration of integration jobs need to be loaded and activated without application restart. Configuration includes job triggers as well as workflow and mapping configurations.
- Software installations and upgrades should be easy to perform in less than 10 minutes by an experienced technical user not familiar with the product.

4.3.13 Integrity Tests, Audits, Logs

Integrity tests may be required due to certain customer requirements. They should include verification of a XML schema, a SWIFT Message as well as custom implementations of checksums.

Auditing of data is not required as relevant data is audited in the Apex Collateral product using Hibernate Envers[32].

Apex Connectivity needs to support logs with configurable levels of granularity. All logs should be based on the SLF4J API[56] in order to be independent of any logging library.

After defining all (non-)functional requirements, the next Chapter outlines design and implementation steps which were taken to satisfy these requirements.

5. Design and Implementation

To satisfy all requirements defined in Chapter 4, the Apex Connectivity Server was built. This chapter documents design and implementation aspects.

5.1 Design

Figure 5.1 introduces a high level architecture overview of the Apex Connectivity Server. It places all individual functional items into a layered structure and therefore differentiates Camel extensions from generic Apex Collateral specific functionality and pieces that will only be used by a single customer.

The individual layers and components of the architecture are described in the following sections about the implementation.

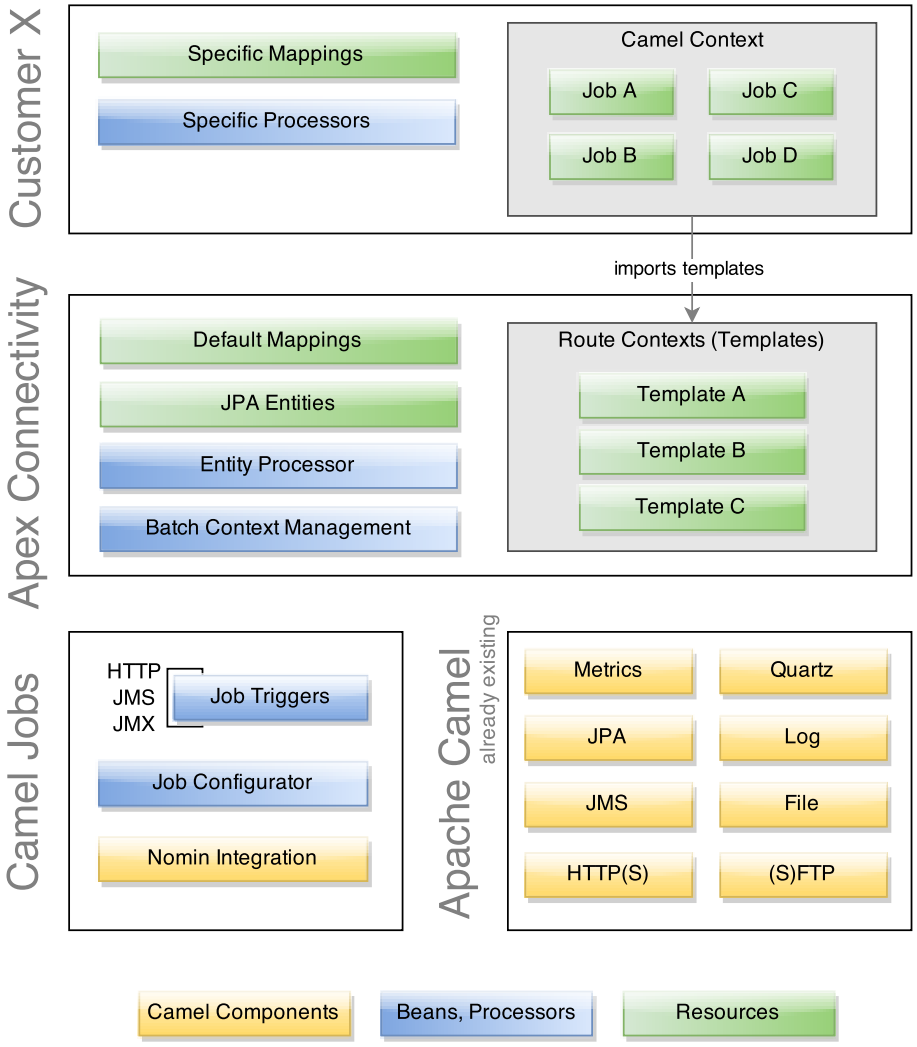


Figure 5.1: Architecture Overview of the Apex Connectivity Server

5.1.1 Technology

Apex Connectivity is based on Java and Apache Camel as these technologies were imposed by SunGard for this project. We additionally decided (see B.3) to use the Spring framework and Apache Maven as basis technologies to build Apex Connectivity. Appendix B.3 and B.5 discuss the evaluation of Spring and Maven.

Early in the project, the lack of a data mapping tool was identified as important missing feature of Apache Camel. Following an evaluation of several data mapping tools, Nomin[46] was chosen as the data mapper. It is a Groovy based mapping library and provides simple yet powerful mappings. Any technical project member as well as experienced developers are able to write mappings according to their skills. The detailed evaluation is documented in appendix B.1.

5.1.2 Infrastructure

The Apex Connectivity Server is embedded into an existing infrastructure of servers and components. The combination of these is very flexible. Figure 5.2 introduces a typical infrastructure.

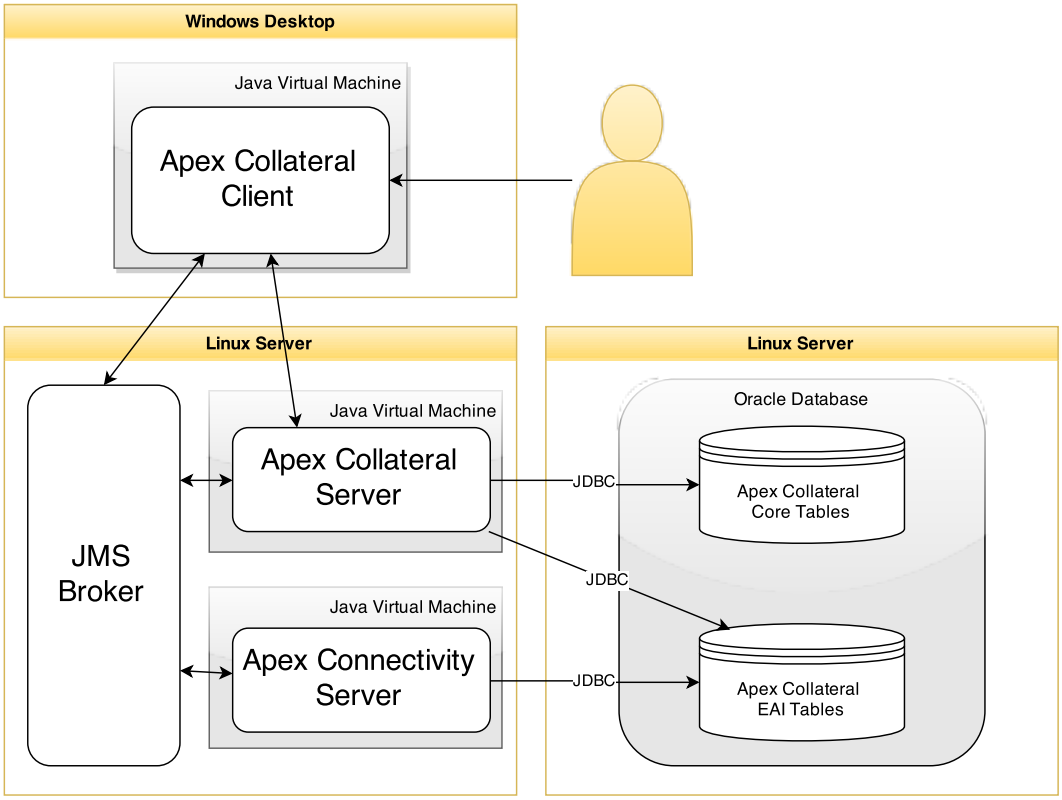


Figure 5.2: Example infrastructure of Apex Collateral and Apex Connectivity Server

The infrastructure surrounding the Apex Connectivity Server was not scope of the semester thesis and has not been dealt with.

We created a simplified deployment of Apex Collateral in order to develop the Apex Connectivity Server against real interfaces. Figure 5.3 illustrates the setup on a Hochschule für Technik Rapperwswil (HSR) Linux server and a developer desktop.

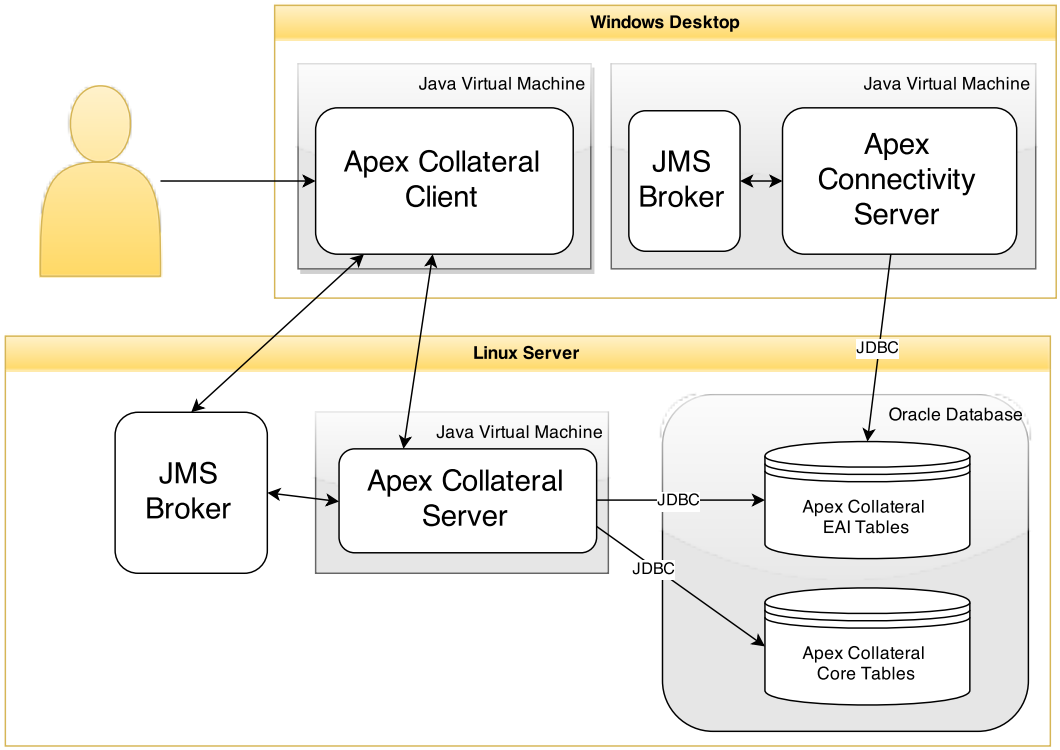


Figure 5.3: Development infrastructure of Apex Collateral and Apex Connectivity Server

Several differences can be observed when comparing Figure 5.3 with Figure 5.2:

1. Apex Connectivity is running with an embedded JMS broker. The JMS broker does not necessarily need to be the same as the one used by the Apex Collateral Server. We separated it to keep this dependency on the same machine like the Apex Connectivitiy Server.
2. Apex Connectivity is running on a different machine than Apex Collateral. This reduced our development cycle as we did not need to deploy it to the project server to test new versions.
3. The Oracle database is running on the same server as Apex Collateral. This setup would be unusual for a real customer infrastructure.

The Apex Collateral Connectivity Server is a single tier application and can therefore not be deployed in a distributed manner. All server components are operated in a Spring[53] application context. This is abbreviated in all deployment diagrams.

5.2 Project Structure

Figure 5.4 outlines the Apache Maven projects and their dependencies. Vista and Spirio bank are two fictitious bank names to illustrate the dependencies and positioning of customer specific source code.

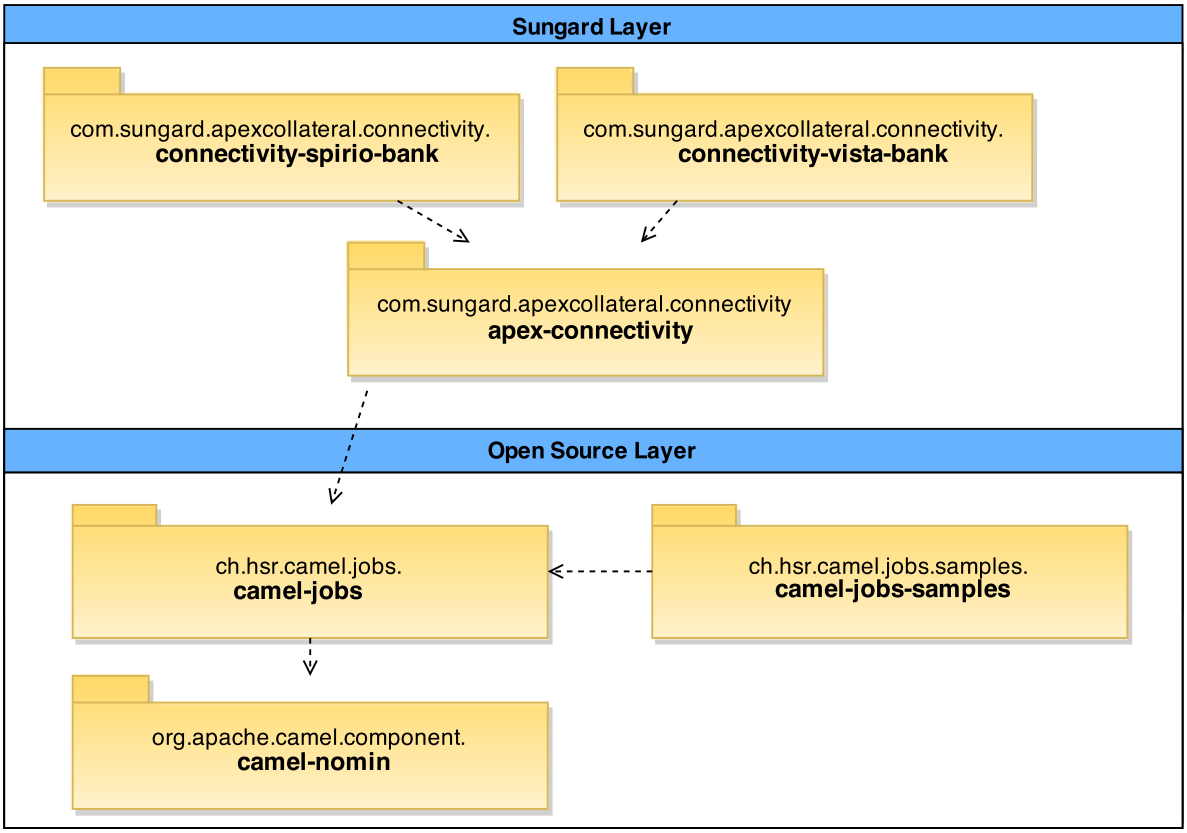


Figure 5.4: Project structure and dependencies

The source code is split into several different projects to ensure separation of open source and proprietary parts as well as to establish reusability of components such as the Camel[10] integration of Nomin[46].

5.2.1 Open Source Components

All open source components are publicly available on GitHub[1].

- *camel-nomin* contains the Camel endpoint for Nomin and allows the developer to use a Nomin mapping in a Camel URI.
- *camel-jobs* enhances Camel with features for management and monitoring of integration jobs.
- *camel-jobs-samples* showcases the capabilities of the camel-jobs module.

5.2.2 SunGard Proprietary Components

- *apex-connectivity* contains all SunGard-specific code such as the interface structure of the EAI Tables.
- *connectivity-spirio-bank* showcases the capabilities of the solution for a fictive apex customer called Spirio Bank.
- *connectivity-vista-bank-jee* demonstrates the ability to deploy the Apex Connectivity Server into Apache Tomcat.

5.2.3 Shared Spring Context

All the components and Camel definitions are integrated using a Spring application context. The configuration is based on a list of Spring XML files located in the different the Maven projects. Figure 5.5 lists all projects and context configuration files.

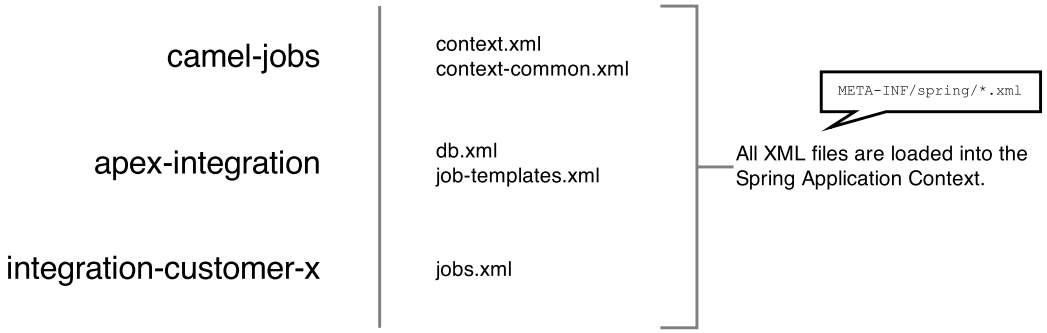


Figure 5.5: Spring Context

[1] github.com/gysel/camel-jobs

5.3 An Integration Job Engine using Apache Camel

This section describes how Camel is used in Camel Jobs to support management and monitoring of integration jobs.

5.3.1 Integration Jobs modeled using Camel Routes

Data integration with Camel is based on Camel routes. A route is always triggered by a single endpoint and then sends the exchanges to one or multiple other endpoints. Listing 5.1 shows a simple route reading from a JMS queue and forwarding the data to two Java beans before logging the result.

Listing 5.1: Simple Camel route written in the Spring DSL

```
<route>
    <from uri="jms:queue:prices"/>
    <bean ref="priceCalculator"/>
    <bean ref="priceMapper"/>
    <log message="price has been calculated and mapped, result: ${body}" />
</route>
```

In Camel Jobs, routes are used for multiple purposes and should not be mistaken as the same thing as an integration job. Listing 5.2 shows an example where the route *scheduler-fx-import* periodically executes the route *job-fx-import*. While the first is merely a trigger as described in Chapter 4, Section 4.2.5, the second is the actual execution route containing the integration logic. Both routes together form an integration job.

Listing 5.2: Two Camel routes building an integration job

```
<route id="scheduler-fx-import">
    <from uri="timer://timer?fixedRate=true&period=60000" />
    <log message="Trigger periodical fx-import" />
    <to uri="vm:trigger-job-fx-import" />
</route>

<route id="job-fx-import">
    <from uri="vm:trigger-job-fx-import" />
    <to   uri="ahc:http://openexchangerates.org/api/latest.json?app_id={{openexchangerates.
      api.key}}" />
    <log message="Got ${body.base} rates as of ${body.date}." />
     [...]
</route>
```

5.3.2 Route Definition DSL

Camel Routes can be defined using one of the following Domain Specific Language (DSL)s:

- Java DSL - A Java-based DSL using the fluent builder style.
- Spring XML - A XML-based DSL in Spring XML files
- Blueprint XML - A XML-based DSL in OSGi Blueprint XML files
- Groovy DSL - A Groovy-based DSL using Groovy programming language
- Scala DSL - A Scala-based DSL using Scala programming language

Furthermore it is also an option to design a new DSL specifically for integration jobs.

The chosen DSL should be based on a language that is known to most of the users. This reduces the list of options to either a Spring XML or a Java DSL.

The following architectural decision is documented using the "Y-Template"[59].

> In the context of *choosing a suitable DSL* facing *the requirement of concise and powerful route definitions*, we decided for *Spring XML-based route definitions* and neglected *Java-based route definitions* to achieve *better readability and comprehensibility for non-developers* accepting *the increased verbosity of XML and the slightly reduced feature set.*

5.3.3 Split the Camel Context to Support a Layered Architecture

Camel routes and configurations like exception handlers are defined within a Camel context, which is built by a *<CamelContext>*-Tag in a Spring context file. The Camel context needs to be shared between the multiple projects as shown in Figure 5.6.

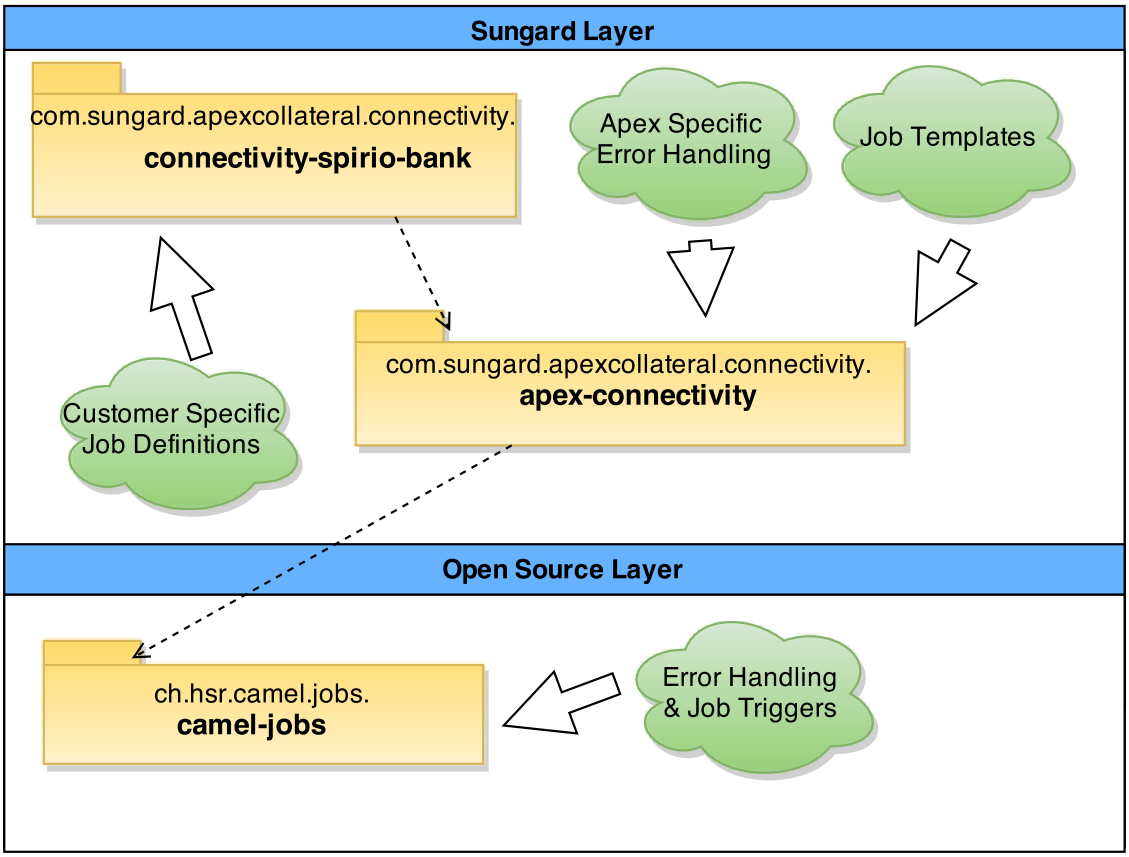


Figure 5.6: Camel Context Definitions in different projects

The features visualised with clouds show which parts of an integration job is defined in which project. Every part of a job is constructed with Camel routes either using the Java DSL as in Camel Jobs or the Spring DSL as in the SunGard specific projects. In order to construct integration jobs like this, all components need access to the same instance of the Camel context.

Camel Limitation

Unfortunately, the architecture described is not supported by the Camel Spring DSL as it is not possible to split *<CamelContext>* definitions over multiple files or projects[1]. Doing so creates multiple Camel contexts running in parallel.

[1]See Claus Ibsen's comment on our question regarding this limitation: http://stackoverflow.com/questions/29900123/split-camel-context-definition-into-multiple-files-jars/29983577#29983577

Bypassing Context Limitation using the AdviceWith Feature

Camel provides a powerful feature called adviceWith[8], which enables changing Camel routes during runtime using the Java DSL. While this feature was designed for test purposes, it is used within Camel Jobs to enhance already defined routes. Like this, job definitions can be written in a customer specific project and then enhanced with error handling and other configurations within Camel Jobs.

Changing Camel routes during runtime might not pass a security audit as it would allow an attacker to inject malicious steps into a running route.

Listing 5.3: adviceWith example

```
1 route.adviceWith(camelContext, new RouteBuilder() {
2     @Override
3     public void configure() throws Exception {
4         interceptFrom().setHeader("ExecutionId", simple("${routeId}-${date:now:yyyyMMdd-
      HHmmss}"));
5     }
6 });
```

Listing 5.3 shows an example how a Camel route can be enhanced with a header containing an ExecutionId.

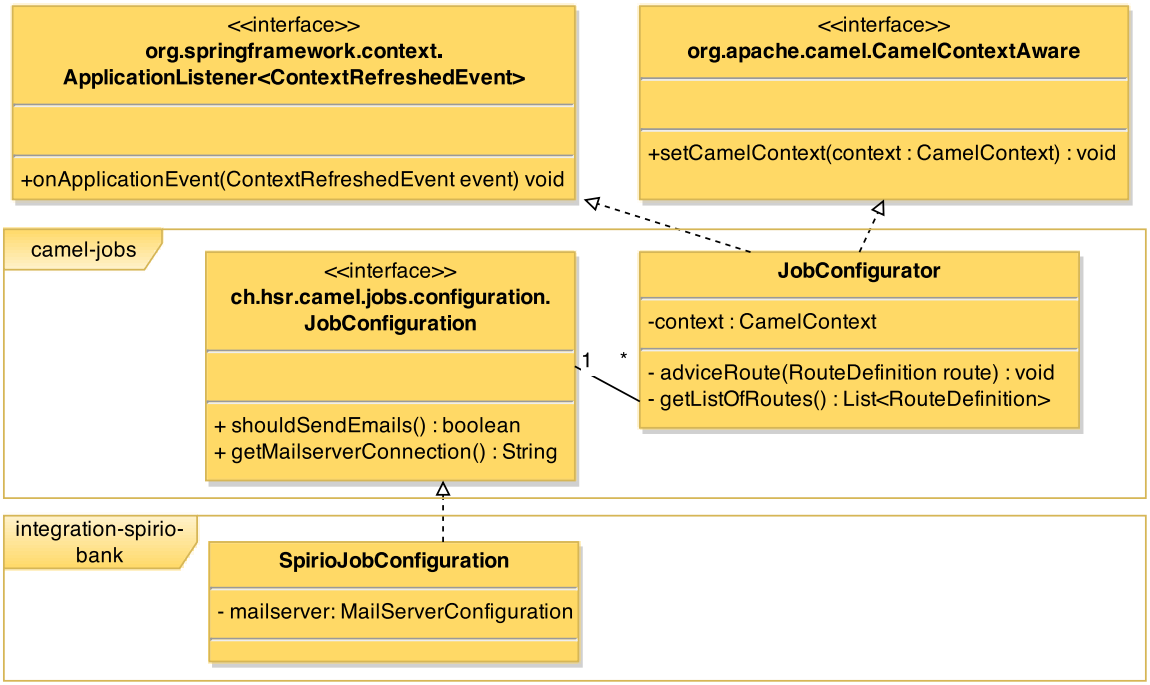


Figure 5.7: JobConfigurator.java class diagram

Figure 5.7 shows the class diagram for the job configuration. *ApplicationListener* and *CamelContextAware* are already existing interfaces provided by the Spring framework and Camel. Implementing these interfaces, a JobConfigurator object, being called after context loading, enhances all existing jobs with an ExecutionId and handling for errors and failed records. The JobConfigurator can be customised for every customer project using a JobConfiguration object.

Using the Java DSL and the Camel AdviceWith feature within the JobConfigurator, we were able to bypass the Camel Context limitation introduced by the Spring DSL.

5.4 Data Mapping

We did an extensive evaluation of data mapping tools that is documented in Appendix B.1. The assessment did not reveal any satisfying graphical or configuration-based mapping tool. We therefore decided to use Nomin[46], a Groovy-based library.

The core of Nomin is written in Java and Groovy and is very simple to use as demonstrated in Listing 5.5. The referenced mapping file `map2car.groovy` is shown in Listing 5.7.

Listing 5.4: A simple Nomin mapping in Java

```
1 Map<String, Object> in = new HashMap<>();
2 in.put("MARKE", "Tesla");
3 in.put("MODELL", "Model S");
4 in.put("MOTORENTYP", "Elektrisch");
5
6 NominMapper nomin = new Nomin("map2car.groovy");
7 Car car = nomin.map(in, Car.class);
```

5.4.1 Nomin Integration for Camel

To integrate Nomin into Camel routes without having Frank Frontline to write Java code, we developed a Nomin extension for Camel. The Camel URI format is:

Listing 5.5: The Nomin URI format for Camel

```
nomin:destination?options
```

The `destination` represents the fully qualified name of the class the message body will be converted to.

Name	Default value	Description
mapping	mapping.groovy	The file to read the mapping definition from.

Table 5.1: Nomin component options

The example route in Listing 5.6 converts a map into a car object using 5.7 as the mapping logic.

Listing 5.6: A route using the Nomin component

```
<from uri="vm:map2car" />
<to uri="nomin:org.apache.camel.component.Car?mapping=map2car.groovy" />
<log message="Message body is a car now: ${body}" />
```

A Nomin mapping starts with a list of imports followed by the mapping type instruction. `mappingFor` declares the side a and side b. This mapping can then be used to convert records from from a to b as well as vice versa.

Listing 5.7: map2car.groovy - The Nomin mapping file

```
import org.apache.camel.component.Car
import org.apache.camel.component.Car.EngineType

mappingFor a: java.util.Map, b: Car

// number of wheels is always 4. value is not present in a
b.nrOfWheels = 4
// brand on b maps to MARKE on a
b.brand = a['MARKE']
b.model = a['MODELL']
b.engineType = a['MOTORENTYP']
// example for a text to Java Enum mapping
// first value is from side a and b is the second value.
simple (["Diesel",EngineType.DIESEL], ["Benzin",EngineType.PETROL], ["Elektrisch",
    EngineType.ELECTRIC])
```

This example can be found in the JUnit test suite in the class `NominComponentTest` the the camel-nomin project.

5.4.2 Code Tables

The old SunGard EAI Server supports code tables to convert for example a numerical value into the corresponding three letter currency code. This can be implemented in Apex Connectivity using simple Nomin conversion as shown in Listing 5.8.

At the request by a Frank Fontline representative during the usability review documented in B.1.3, we added a lookup feature which allows code mapping from a external property file as shown in Listing 5.9 and 5.10.

Listing 5.8: Code mapping using Nomin conversions

```
    a.currency_code = b.currency
    simple ([1, 'CHF'], [2, 'USD'])
```

Listing 5.9: A code lookup in a Nomin mapping

```
1  import static com.sungard.apexcollateral.integration.mapping.CodeTable.map;
2
3  mappingFor a: java.util.Map, b: FxRate
4
5  // mapped fields
6  b.quotedCurrency = {map("currencycodes.properties", a.code)}
```

Listing 5.10: currencycodes.properties containing a list of codes

```
1  1=CHF
2  2=USD
3  3=EUR
```

5.5 Job Triggers

The jobs can be triggered using a RESTful HTTP, a JMS message or a Java Management Extensions (JMX) method invocation.

5.5.1 RESTful HTTP

The RESTful HTTP API runs on either an integrated Jetty server or using a servlet in a servlet container.

The following operations are supported:

- `GET /jobs` returns a list of all jobs.
- `GET /jobs/{jobName}` returns details about one job.
- `POST /jobs/{jobName}` starts a job and returns the status as well as collected metrics (see 5.7).

The placeholder `{jobName}` has to be replaced with the actual name of the job. JSON and XML are supported formats and can be requested using the HTTP Header `Accept`.

An example using the Linux bash to start the job *job-fx-import* in the connectivity-spirio-bank reference project:

```
curl -X POST -H "Accept: application/xml" /
      http://localhost:8080/jobs/job-fx-import
```

5.5.2 JMS

An embedded JMS broker supports starting jobs by sending message to a queue named `JobTrigger` providing the header `JobName` containing the name of the job to start.

When a ReplyTo is set on the message, the job execution details will be sent to the specified destination.

5.5.3 JMX

A JMX MBean called `ch.hsr.sa.eai.JobManagement` is available to start jobs. The exposed operation is called `startJob` and takes the job name as the only parameter. This operations returns 0 as the status code for successful executions and 1 for failed runs.

5.6 Job Templates

The requirement of job templates as described in 4.2.4 can be implemented using the Camel route context feature as shown in Listing 5.11.

Listing 5.11: A job template defined with <routeContext>

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:camel="http://camel.
      apache.org/schema/spring"
    xmlns:broker="http://activemq.apache.org/schema/core"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.
      springframework.org/schema/beans/spring-beans-3.0.xsd
     http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
      spring.xsd">

     <routeContext id="exposure-import-template" xmlns="http://camel.apache.org/schema/
      spring">

       <route id="trigger-exposure-realtime-import">
          <from uri="file://in?move=processed&amp;include=exposures.csv" />
          <setHeader headerName="jobName">
             <simple>job-exposure-import</simple>
          </setHeader>
          <to uri="jobManager" />
       </route>

       <route id="job-exposure-import">
          <from uri="vm:trigger-job-exposure-import" />
          <log message="importing exposures, batch: ${header.BatchId}" />
          <transacted />
          <!-- unmarshal an process -->
       </route>

    </routeContext>

     <routeContext id="just-another-job-template" xmlns="http://camel.apache.org/schema/
      spring">
       <route>
       </route>
    </routeContext>
</beans>
```

Job templates are defined in the apex-connectivity project and imported by multiple customer specfic projects as shown in Listing 5.12.

Listing 5.12: Import a template in a Camel context

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <routeContextRef ref="exposure-import-template"/>
</camelContext>
```

If the integration job uses a Nomin data mapping, the mapping can be overriden by the customer specific project by providing an identical named mapping file in the classpath.

5.7 Metrics

Camel offers a metrics component to collect data directly from routes as seen in 5.13. Supported types are counter, histogram, meter and timers.

Listing 5.13: A metric of type counter

```
1 <to uri="metrics:counter:a-metric-name" />
```

We added to ability to link certain metrics to jobs and access them as part of the job trigger API (see 5.5)). We defined the following naming conventions of counter metrics in order by expose them via the API.

Metric Name	Description
job-{name}.successful	The record or message has been successfully processed.
job-{name}.failed	An exception has occurred during process of the record or message.
job-{name}.rejected	The message or record contains an unexpected content or format and should be examined.
job-{name}.ignored	The message will not be processed as it was rejected by a message filter.

Table 5.2: Metric naming conventions

Counter metrics are not bound to a route execution and therefore will by default never be reset to zero. We implemented a component to erase the counter values in a regular interval. The desired interval, e.g. once a day at midnight, can be configured using the entry `metrics.reset.cron` in the application properties file which takes a Quartz [48] cron expression.

5.8 Batch Processing

Batch processing is a SunGard specific concept imposed by the EAI Tables and documented as requirement in Chapter 4, Section 4.2.6.

Every interface table is mapped as a JPA entity and inherits from a common parent `EaiEntity`. This class contains a reference to `EaiBatch` representing the record which combines a set of data into one atomic import or export set. Those classes are visualised in Figure 5.8.

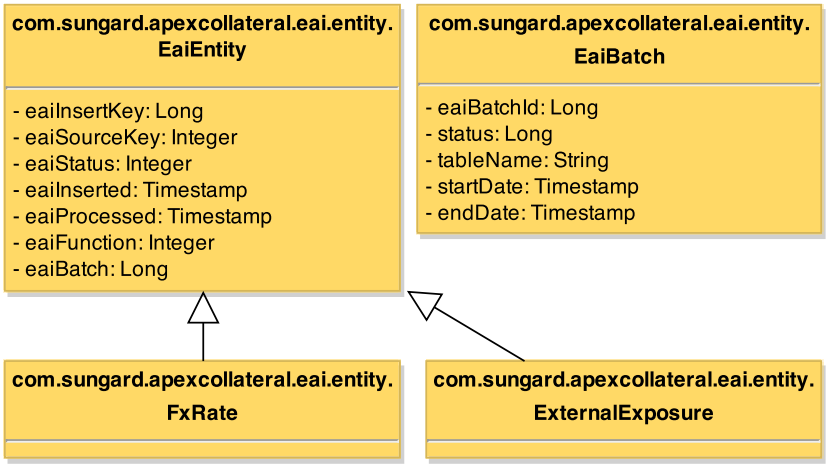


Figure 5.8: Batch job classes implemented in Apex Connectivity

The foreign key `EaiEntity.eaiBatch` is a weak reference and not a mapped `@ManyToOne` relationship. This allows simpler handling in parallel contexts.

Listing 5.14 outlines a job using a batch context. The following pieces are required to use batches.

1. An import of the `create-batch-context` route (line 10).
2. The header `BatchTableName` (lines 14-16).
3. The header `BatchId` (lines 17-19).
4. A message to `direct:create-batch-context` to create the batch record in the database (line 21).
5. A call to the bean `entityProcessor` once for every record which sets the batch id amongst other common fields (line 35).

Listing 5.14: Code example of a batch import

```
1  <beans xmlns="http://www.springframework.org/schema/beans"
2    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xmlns:camel="http://camel.apache.org/schema/spring"
4    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.
       springframework.org/schema/beans/spring-beans-3.0.xsd
5     http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
       spring-2.14.0.xsd">
6
7     <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
8
9       <routeContextRef ref="create-batch-context"/>
10      <routeContextRef ref="exposure-import-template"/>
11
12      <route id="job-exposure-batch-import">
```

```
13          <from uri="file://in?move=processed&amp;include=exposures-batch.csv" />
14          <setHeader headerName="BatchTableName">
15             <simple>EAI_TRADE_EXPOSURE</simple>
16          </setHeader>
17          <setHeader headerName="BatchId">
18             <simple>${bean:idGenerator.nextId}</simple>
19          </setHeader>
20          <!-- wireTap does not modify the original message body! -->
21          <wireTap uri="direct:create-batch-context" />
22          <log message="New batch record with id ${header.BatchId} created" />
23          <to uri="direct:exposure-processing" />
24       </route>
25
26       <route id="exposure-processing">
27          <from uri="direct:exposure-processing" />
28          <log message="importing exposures" />
29          <transacted  />
30          <unmarshal ref="delimitedFormat"/>
31          <split strategyRef="listAggregationStrategy">
32             <simple>${body}</simple>
33             <convertBodyTo type="java.util.Map" />
34             <to uri="nomin:com.sungard.apexcollateral.eai.entity.ExternalExposure?mapping=
       map2exposure.groovy" />
35             <to uri="entityProcessor" />
36          </split>
37          <log message="parsed ${body.size} exposures" />
38          <to uri="jpa:ExternalExposure?usePersist=true" />
39          <log message="finished importing exposures" />
40       </route>
41
42    </camelContext>
43 </beans>
```

5.9 Performance

Apache Camel provides several building blocks to implement performing and scalable routes to satisfy the requirement documented in Chapter 4, Section 4.3.8. The core of scalable routes is formed by components supporting the *parallel execution* flag (e.g. the Splitter) and the Staged Event-Driven Architecture (SEDA) component which is an implementation of the SEDA framework suggested by David Welsh in his PhD[6].

The performance of Java applications is mainly dependent on the following key parameters:

1. Memory configuration (e.g. heap size, PermGen size)
2. Computational speed (e.g. number of CPUs and their clock rate)
3. Round-trip time between application components (e.g. application server and database)
4. Availability of pooled resources (e.g. database connection pool)

Assuming that all of the mentioned parameters are set to an optimal value, a second set of criteria has to be considered.

1. Split work in smaller transactions
2. Process data in parallel transactions

Both splitting the work and processing it in parallel is configurable as demonstrated in Listing 5.15. The optimal values of those parameters need to be obtained using experimental testing.

Listing 5.15: Configuration file with performance related parameters

```
# The maximum number of database connections
# Should be at least pool.size.max + 10
db.maximumPoolSize=30
# Number of records to be processed in a transaction
# Used by ListUtil to split work in smaller junks
batch.recordsPerTransaction=200
# Thread pool settings
pool.size.max=20
pool.size=10
```

The thread pool related properties also need a thread pool profile configuration in the Camel context as shown in Listing 5.16.

Listing 5.16: Thread pool profile definition

```
<threadPoolProfile id="defaultThreadPoolProfile" defaultProfile="true"
    poolSize="{{pool.size}}" maxPoolSize="{{pool.size.max}}" maxQueueSize="0"
    allowCoreThreadTimeOut="false" rejectedPolicy="CallerRuns" />
```

The Listing 5.17 showcases an import using two routes. The first route runs only once. The second route is activated multiple times in parallel using a direct consumer and processes a subset of records.

Listing 5.17: Code example of a batch import

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:camel="http://camel.apache.org/schema/spring"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.
      springframework.org/schema/beans/spring-beans-3.0.xsd
     http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
      spring-2.14.0.xsd">

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">

      <routeContextRef ref="create-batch-context"/>
      <routeContextRef ref="exposure-import-template"/>

      <threadPoolProfile id="defaultThreadPoolProfile" defaultProfile="true"
         poolSize="{{pool.size}}" maxPoolSize="{{pool.size.max}}" maxQueueSize="0"
         allowCoreThreadTimeOut="false" rejectedPolicy="CallerRuns" />

      <route id="job-exposure-import">
         <from uri="vm:trigger-job-exposure-import" />
         <log message="importing exposures, batch: ${header.BatchId}" />
         <!-- parsing and mapping -->
         <log message="parsed ${body.size} exposures" />
         <split  parallelProcessing="true">
            <method bean="listUtil" method="partition" />
            <to uri="direct:exposure-persisting?timeout=0" />
         </split>
         <log message="finished importing exposures" />
      </route>

      <route id="exposure-persisting">
         <from uri="direct:exposure-persisting" />
         <transacted />
         <log message="importing ${body.size()} exposures" />
         <to uri="jpa:ExternalExposure?usePersist=true" />
         <log message="imported ${body.size()} exposures" />
      </route>


    </camelContext>
</beans>
```

This approach retains a synchronous approach. The first route only finishes once all records have been processed. This guarantees that the job manager correctly recognises the end time of the job. A better performance can be attained when using SEDA as shown in Listing 5.18. However this means that this route is based on an asynchronous approach and that the first route finishes even when the import has not ended yet. Depending on the business context, this might be possible and the person developing the job has to decide which approach is suitable to solve his business need.

Listing 5.18: Code example of a SEDA batch import

```
<route id="job-exposure-import">
    <from uri="vm:trigger-job-exposure-import" />
    <log message="importing exposures, batch: ${header.BatchId}" />
    <!-- parsing and mapping -->
    <log message="parsed ${body.size} exposures" />
    <split>
        <method bean="listUtil" method="partition" />
        <to uri="seda:exposure-persisting" />
    </split>
</route>

<route id="exposure-persisting">
    <from uri="seda:exposure-persisting?concurrentConsumers=15" />
    <transacted />
    <log message="importing ${body.size()} exposures" />
    <to uri="jpa:ExternalExposure?usePersist=true" />
    <log message="imported ${body.size()} exposures" />
</route>
```

5.9.1 Performance Tests

A performance test with two physically separated machines running the application server and the database server verified that used performance tuning efforts are sufficient. A data set of 10'000 records can be imported in 15 seconds. It is safe to assume that the execution time of imports scale in a linear way. 1 million records would therefore be imported in approximately 25 minutes. This is lower than the initially defined target of 3 million records in 15 minutes but SunGard accepted this reduced throughput. Furthermore is it questionable whether it is possible to achieve a better performance than the one measured using Hibernate and a remote database. Optimised JDBC code and removing the network layer might be required to further improve import performance.

5.10 Deployment

Apex Connectivity can be deployed standalone as a shaded Apache Maven jar encapsulating all dependent libraries. The other possibility is to package it into a `.war` file and deploy it into a Java servlet container like Apache Tomcat.

5.10.1 Standalone

Use Maven to create a `.jar` file with all dependencies embedded. This process makes use of the Maven shade plugin[12]. For an example usage of the Maven shade plugin

see the pom.xml file in the connectivity-spirio-bank project. The Listings 5.19 and 5.20 demonstrate how to package and start the Apex Connectivity Server.

Listing 5.19: Use maven to create an executable jar file

```
1 cd connectivity-spirio-bank
2 mvn package
```

When starting the Java process, use `-fa classpath*:META-INF/spring/*.xml` to instruct the main class `org.apache.camel.spring.Main` which Spring context files to load.

Listing 5.20: Start the standalone server

```
1 java -jar connectivity-spirio-1.0.0.jar -fa classpath*:META-INF/spring/*.xml
```

5.10.2 Application Server

The project `connectivity-vista-bank-jee` demonstrates how to create a structure in order to package it into a servlet container.

The following building blocks are required:

1. `pom.xml` specifies the packaging (`.war`) and additional dependencies: `camel-servlet` and `spring-web`.
2. `src/main/webapp/WEB-INF/web.xml` defines the Spring `ContextLoaderListener` loading `jee-context.xml`.
3. `src/main/webapp/WEB-INF/jee-context.xml` defines a Camel context and loads all the necessary resources.

The only difference compared to a standalone Camel context is the REST configuration using a servlet instead of Jetty to serve the HTTP requests. Listing 5.21 illustrates such a context.

Listing 5.21: A Java application server Camel context

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring"
    autoStartup="false">

    <!-- let Camel use @Component scanned route builders -->
    <contextScan/>

    <restConfiguration component="servlet" bindingMode="json"
        contextPath="ApexIntegrationJEE/rest" port="8080">
        <dataFormatProperty key="prettyPrint" value="true" />
    </restConfiguration>

    <route id="job-sayhello">
        <from uri="vm:trigger-job-sayhello" />
        <log message="Hello World!" />
    </route>

</camelContext>
```

Use the Maven *package* goal as shown in Listing 5.22 to create a `.war` file with all dependencies embedded. This process makes use of the Maven WAR plugin[13].

Listing 5.22: Use maven to create an war file

```
cd connectivity-vista-bank-jee
mvn package
```

The .war file generated in `/target` can be deployed to any Java application server from which Apache Tomcat was specifically tested.

5.11 Hawt.io Integration

Hawt.io[31] is a web console to manage Java applications. It uses a Jolokia[39] agent to communicate with the Java Virtual Machine (JVM) using JSON over HTTP.

The Apex Connectivity Server integrates Jolokia using the Spring agent. A simplified version of `context.xml` is shown in Listing 5.23 .

Listing 5.23: A Spring-based Jolokia agent

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jolokia="http://www.jolokia.org/jolokia-spring/schema/config"
    xsi:schemaLocation="http://www.jolokia.org/jolokia-spring/schema/config
    https://raw.githubusercontent.com/rhuss/jolokia/master/agent/jvm-spring/src/main/
      resources/org/jolokia/jvmagent/spring/jolokia-config.xsd
    http://www.springframework.org/schema/beans http://www.springframework.org/schema/
      beans/spring-beans-3.0.xsd">
    <!-- the jolokia-spring xsd is directly referenced from github
         as the official url returns a 404 -->

    <jolokia:agent lookupConfig="false" systemPropertiesMode="never">
        <jolokia:config autoStart="true" host="0.0.0.0" port="8778" />
    </jolokia:agent>

</beans>
```

Hawt.io can then be instructed to connect to the Apex Connectivity Server using HTTP on port 8778.

Apache Camel[10], ActiveMQ[7] and Quartz[48] are supported by Hawt.io out of the box. Integration jobs can be triggered using the JMX or the JMS API. An example of a Camel route visualisation is shown in Figure 5.9.

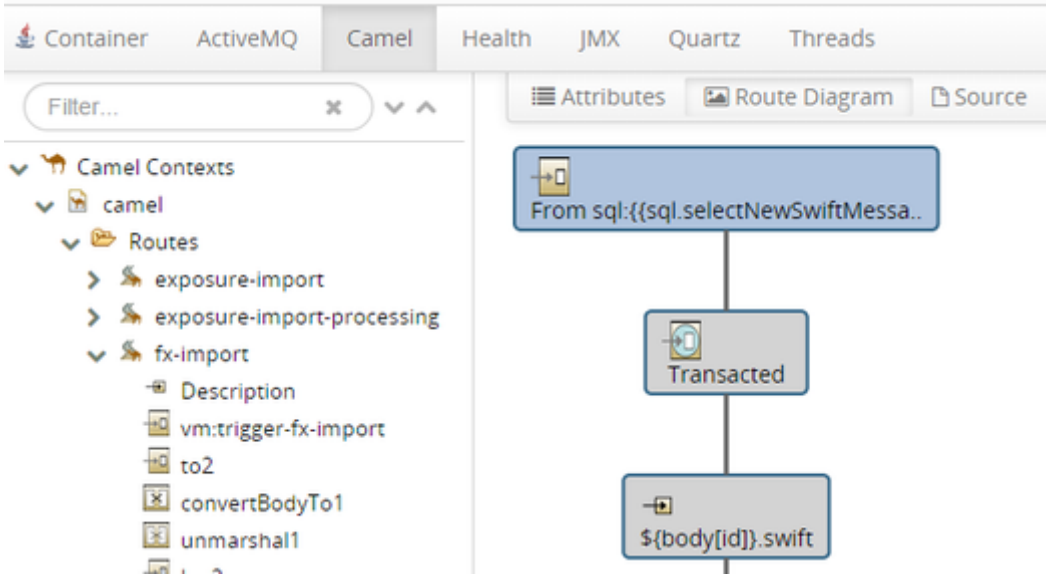


Figure 5.9: Hawt.io screenshot

Hawt.io is developed actively and has gained high popularity in the past years. Enhancing it with additional plugins to support Camel Jobs specifics like job results and errors, hawt.io could replace all other currently needed monitoring mechanisms. Our ideas for enhancing hawt.io are documented in Section 6.2.

After covering the important design and implementation aspects, the next chapter assesses the built solution described in this chapter against the defined requirements.

6. Conclusion

The detailed analysis in Chapter 3 of the integration solution currently used in Apex Collateral has brought forward many flaws which influenced the requirements for Apex Connectivity listed in Chapter 4. The empiric approach described in Appendix A, persona-based requirements and detailed evaluations documented in Appendix B allowed us to conceptually design and build Apex Connectivity — the new integration solution for the Apex Collateral product.

The following requirement assessment proves that we were able to built a integration job engine that satisfies most of SunGard's needs and does not implicate the flaws identified in the old solution. Two reference projects with three reference jobs were built to prove the ability of the Apex Connectivity Server based on Apache Camel.

We are very satisfied with the Apache Camel as basis framework as it is extremely powerful, well documented and easy to learn. The layered architecture allowed us to publish the core integration job engine named *Camel Jobs* as open source software on GitHub.

Nomin Mappings allow us to address both Paul Perfectonist and Frank Frontline with the very same tool. As integration jobs can be reused from job templates, integration work for a new Apex customer can be reduced significantly with the new Apex Connectivity Server.

Nevertheless, the Apex Connectivity Server has to prove itself within a real integration project. SunGard plans to use it for a first project within the next months.

6.1 Requirement Assessment

This section assesses the developed solution based on the defined requirements. The example projects connectivity-spirio-bank and connectivity-vista-bank serve as test scenarios for the assessment.

All requirements are rated with a rating from 1-3:

1 The requirement is fully satisfied.

2 The requirement is partly satisfied.

3 The requirement is not satisfied.

6.1.1 Functional Requirements

Table 6.1 assesses the provided solution with the defined functional requirements described in Chapter 4, Section 4.2.

Requirement	Rating	Assessment
4.2.1 Basic Data Mapping	1	Supported by Nomin and camel-nomin.
4.2.1 Advanced Data Mapping	1	Supported by Nomin except *Multi-Record Conditions*, which can be built using a Camel Aggregator.
4.2.2 Integration Workflow	1	Supported by Camel Enterprise Integration Patterns.
4.2.3 Protocols and Data Formats	2	Camel supports endpoints for all required protocols and data formats. SOAP, FTP, and SFTP have not been tested within the reference projects.
4.2.5 Job Triggers	1	JMS and REST triggers are implemented and tested in Camel Jobs. The connectivity-spirio-bank project contains examples for database, file, and scheduled triggers.
4.2.5 Job Result	1	A JobResult is generated for every execution. If the REST or JMS API are used as a trigger, the result is returned to the caller in XML or JSON.
4.2.6 Batch Processing	1	Batch processing is implemented and tested by the exposure-import-job in the connectivity-spirio-bank project.
4.2.4 Job Templates	1	Job templates are realised with the Camel RouteContext feature[21]. The apex-integration project contains a reference implementation (see *job-templates.xml*). Data mapping can be overridden by the customer specific project using the template by providing an identical named mapping file in the classpath.
4.2.4 Delta Builder	3	Delta Builders have not been implemented within the thesis project. This requirement is very SunGard specific and has therefore been prioritised lower than other features.

Table 6.1: Assessment of functional requirements

6.1.2 Non-Functional Requirements

Table 6.2 assesses the provided solution with the defined non-functional requirements described in Chapter 4, Section 4.3.

Requirement	Rating	Assessment
4.3.1 Usability	1	Representatives of Frank Frontline and Alex All-rounder have approved that the usability of job creation and data mapping is sufficient. See Appendix B.1.3 for minutes of meeting.
4.3.2 Simplicity	2	The outlined metrics have only been validated with a feature walk-through. A verification using an actual user test has not happened as it was not given a high priority in agreement with SunGard.
4.3.3 Information Security	2	An encrypted HTTPS connection is used in the reference project connectivity-spirio-bank. Support for encrypted JMS messaging and SFTP exists but has not been tested in the reference projects.
4.3.4 Separation of Concerns	2	Separation is ensured by the project structure. An integration expert works only with customer specific projects and does not access the internals of Camel Jobs or the configuration done by a developer in apex-integration. An exception is a job template as route and mapping definitions are placed within the apex-connectivity project
4.3.5 Open Closed Principle	1	Java code can be used within the Nomin mappings and as a processor within Camel routes which leads to great extensibility for integration jobs with special requirements.
4.3.6 Testability	1	These test classes in the reference projects are examples for the test layers: • Unit test: EntityTests.java (apex-connectivity) • Integration test: FxImportTest.java (connectivity-spirio) • Acceptance test: ExposureImportTemplateTest.java (apex-connectivity)

4.3.7 Deployment	1	A build of the connectivity-spirio-bank project generates a Java Archive (JAR) file that can be run standalone. A build of the connectivity-vista-bank project generates a Web application ARchive (WAR) file that can be deployed to a web server like Apache Tomcat.
4.3.8 Performance	2	As described in Chapter 4, Section 5.9, the performance of the provided solution does not satisfy all goals but is sufficient for SunGard. Memory usage can become a problem when importing millions of records in combination with the splitter feature. A streaming approach described in Section 6.2.4 would solve this issue.
4.3.9 Monitoring	1	The RESTful HTTP API provides the required information for monitoring. Exceptions are logged and can optionally trigger email notifications.
4.3.10 Availability & Fault Tolerance	1	The JobConfigurator class supports handling of failed records, exceptions, and the possibility to report to a Fault Observer by email.
4.3.11 Maintainability	1	All used libraries are licensed under either the Apache license, the MIT license, the Eclipse License, the BSD license or the LGPL. Therefore it is possible to use Apex Connectivity in a commercial environment without legal obstacles.
4.3.12 Operations	2	The simple and easy to use deployment process described in 5.10 guarantees the installation of Apex Connectivity within minutes. Hot deployment, meaning that mappings and job definitions can be changed at runtime, is not yet supported as it was prioritised low by SunGard. This feature is only used for testing during the development process.
4.3.13 Integrity Tests, Audits, Logs	2	The Camel Log endpoint uses the SLF4J API[56] for logging. Integrity tests have not been implemented in the reference jobs but can be built using Java code within a Camel processor endpoint.

Table 6.2: Assessment of non-functional requirements

6.1.3 Camel Jobs for Swisscom

In our interviews on data integration described in Chapter 3, Section 3.1.1, a Swisscom architect told us about a use case within their software product that could benefit highly from an integration job engine. In every product instance installed within a customer environment, a Shell script reads a CSV file provided on a FTP server, transforms the data to XML and sends it to a webservice provided by the product. The data mapping done by the Shell script differs for every customer.

We requested a detailed specification document from Swisscom and analysed if the requirements can be met by Camel Jobs. As this is clearly the case, Camel Jobs was recommended to Swisscom as a replacement for the current solution.

6.2 Future Work

This section describes ideas for further development of Camel Jobs.

6.2.1 Hawt.io integration

Hawt.io already supports most of the technologies used by Camel Jobs. With additional plugins for Camel Jobs, Stephen Steward could monitor and manage all aspects of integration jobs with hawt.io. The following enumeration lists ideas on improving hawt.io for Camel Jobs.

- List job executions with their results in hawt.io.
- Access and inspect rejected records in hawt.io.
- List occurred exceptions grouped by job id.
- Trigger a job execution directly from hawt.io.
- Add Nomin support to view, debug, and modify mappings at runtime.

Furthermore, it should be possible to build Camel Jobs with an embedded hawt.io instance to simplify deployment.

6.2.2 Automated Batch Processing

As described in Chapter 5, Section 5.8, every job supporting batch processing needs to implement the following steps:

1. Set a header field *BatchTableName*
2. Set a header field *BatchId*
3. Send a message to the *create-batch-context* route
4. Send a message to the *entityProcessor* for every entity

Steps 2. and 3. could be automated using the Camel adviceWith feature in the same way the JobConfigurator already uses it. A naming convention for batch jobs would empower a BatchConfigurator bean in the apex-connectivity project to automatically add the *BatchId* field and a message to *create-batch-context* for every batch job.

This requires Camel to handle multiple adviceWith calls correctly as it is planned for Camel release 3.0. Apache has not yet published a release date for Camel 3.0.

6.2.3 Store Job Result in a Database

All job executions are handled by the JobManager bean. A job execution produces a job result containing state, metrics and an execution id. Currently these information are logged and if possible returned to the caller. For a production environment it would be helpful to store every job result in a database for better monitoring and traceability.

6.2.4 Reduce Memory Usage

Camel currently reads all source data (for example a complete file) into memory for processing. It can additionally occur that data is contained multiple times in memory when using features like a splitter. To keep memory usage low it is necessary to provide multiple small source files instead of one big one. A streaming approach that reads and processes only parts of the source would solve this issue.

6.2.5 Abstraction Layer for Job Definitions

The Camel Spring DSL used to define integration jobs is easy to read and write for a developer of any kind who is familiar with XML. But even if the DSL is well documented and described with many examples, a non-developer like Frank Frontline might have difficulties to write simple integration jobs with it. An additional abstraction layer, like a simplified DSL on top of Camel, could help Frank Frontline to write integration jobs without assistance by Paul Perfectionist.

6.2.6 Camel AdviceWith

AdviceWith[8] might impose a security risk to applications. When an attacker gains access to the system, e.g. the management console, he might be able to modify the route and inject malicious steps or remove critical parts.

A feasible solution to this issue would be to enhance Camel so that routes can be locked preventing further modifications.

The presented assessment shows that Camel Jobs, an integration job engine for everyone, is a promising approach to solve scenarios where a software product has to be integrated in multiple customer environments. SunGard is going to use it as its new Apex Connectivity Server, Swisscom might use it for its credit risk product and it is published on GitHub for usage and further development in the open source community.

A. Project Management

This Appendix documents aspects of project management like methodology, roles, environment, quality management, and risk management. The project was time boxed and started on 16. February 2015 and ended on 29. May 2015 which implies 15 working weeks.

A.1 Project Management Methodology

Scrum has been chosen as the project management methodology for this semester thesis. Scrum specifies an iterative and incremental approach which encourages a high involvement of the customer. This characteristics suit the requirements of a semester thesis well, as it provides only a short project definition at project start and, in our case, requires close collaboration with SunGard.

A.1.1 Scrum Sprints

Figure A.1 illustrates Scrum iterations, so called sprints. The Sprint length in the semester thesis was agreed to be two weeks which resulted in seven sprints plus one week for the final preparation of this documentation.

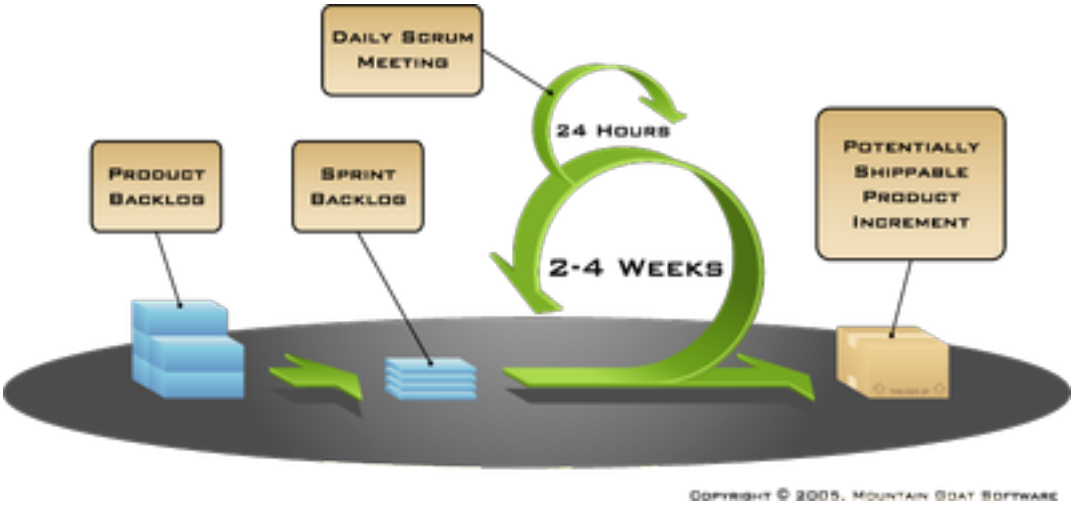


Figure A.1: Scrum Sprint Overview[51]

The *product backlog* contains all requirement yet to be satisfied and is maintained within this documentation throughout the project.

The *sprint backlog* contains all user stories partitioned into tasks for a given sprint. The sprint backlog is managed within an Atlassian JIRA[19] instance provided by SunGard. Figure A.2 shows a JIRA screenshot taken during sprint #7.

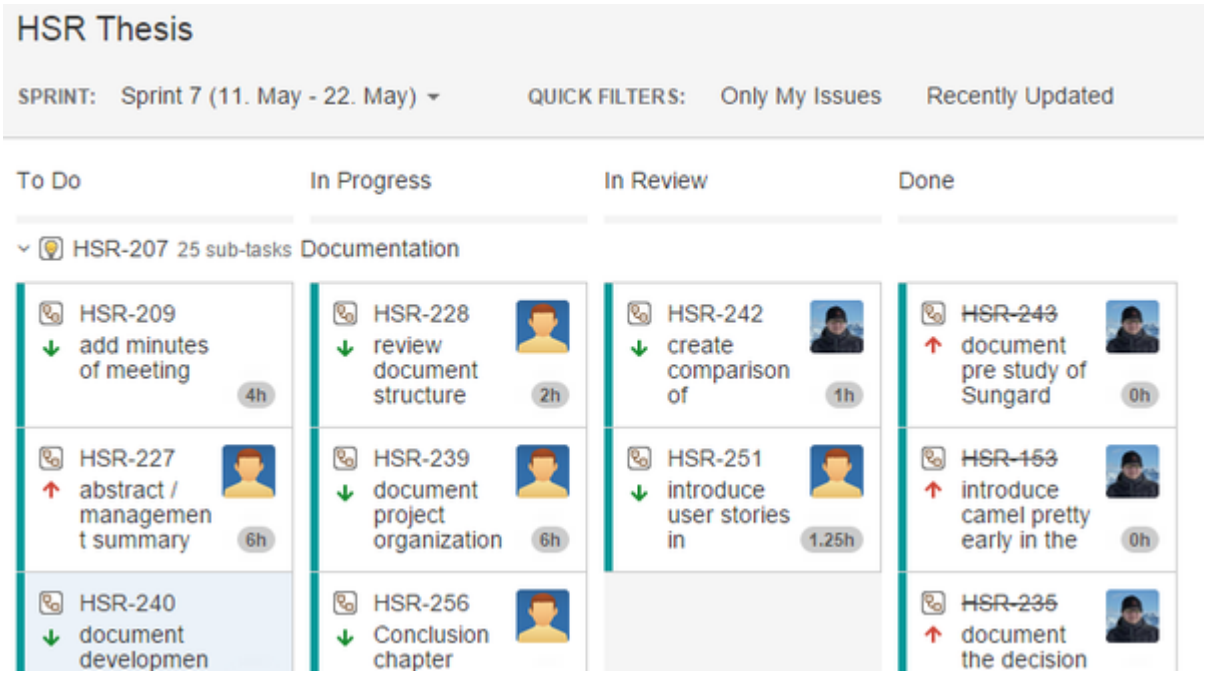


Figure A.2: Sprint Backlog in Atlassion JIRA[19]

A *potentially shippable product increment* is the result of every sprint. A simple integration job implementation with Apache Camel has therefore been implemented very early in sprint #3 and enhanced in the subsequent sprints. A Jenkins[38] Continuous Integration (CI) server as shown in Figure A.3 ensures that the code always builds correctly and all tests result positive.

Figure A.3: Jenkins Continuous Integration Overview

To satisfy all needs by all stakeholders, three regular meetings listed in Table A.1 have been organised and conducted in each sprint.

Day	Meetings	Description
#1	Sprint planning meeting	Team internal meeting to partition the planned backlog items into estimated user stories and tasks.
#2	Sprint status meeting	Meeting with HSR supervisor Olaf Zimmermann to review the project status according to HSR requirements.
#10	Sprint review meeting	Meeting with HSR supervisor and SunGard representatives to assess the work done and prioritise backlog items for the next sprint.

Table A.1: Sprint Meetings

A.1.2 Scrum Roles

Scrum defines three project roles — the product owner, the scrum master and the development team. As the semester thesis is an academical project, the scrum roles could not be mapped one to one. This Section introduces the involved persons and their roles in the project.

HSR Supervisor

Prof. Dr. Olaf Zimmermann is the HSR supervisor of this thesis and incorporates both, the role of the product owner and scrum master. *Product* in this context refers to the thesis itself and not the functional product. He ensures that all requirements by HSR for a semester thesis are met and decides in last instance about the scope of the thesis. As supervisor and coach of the development team, prof. Zimmermann also performs part of the scrum master role as he coaches the development team.

SunGard Representatives

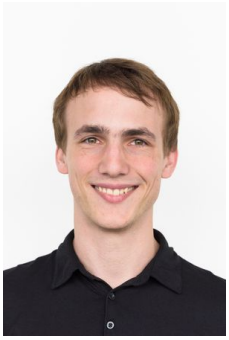
Marcel Roth's role is the Scrum product owner. He represents SunGard and focuses on generating as much business value as possible. To achieve this, he is responsible to define and prioritise backlog items as well as reviewing the product increment after each sprint. Marcel Roth incorporate both, Alex Allrounder and Paul Perfectionist, in one person as he worked as developer and professional service team member within the Apex Collateral team.

Three other SunGard employees, representing Frank Frontline, Alex Allrounder and Paul Perfeconist attended the biweekly sprint review meetings to define and prioritise requirements and assess the provided solutions.

Project Team

Michael Gysel and Lukas Kölbener formed the development team. They worked as an interdisciplinary team in which both were responsible for each part of the project. Both being Certified ScrumMasters®[22], they incorporated part of the scrum master role as they helped maintaining the product backlog and organised everything for correct sprint

operation.



Lukas Kölbener is an information technology student at HSR in his 8th semester. He works part time as Java developer for Super Computing Systems AG in Zurich, building ticket vending machines for the public transport industry.

Michael Gysel is an information technology student at HSR in his 8th semester. He works part time as Java Developer for SunGard in the Apex Collateral team. He could therefore provide many insights in SunGard's expectations towards the new solution.

A.2 Development Environment

Figure A.4 outlines all components of the development environment. Apex Connectivity is developed with Java 7 using Eclipse Luna. HSR has provided a Virtual Machine (VM) on which an instance of Apex Collateral including the EAI Tables is installed. On the same server a CI Jenkins server pulls for changes from the Git repositories on GitHub to build and test the different projects. Table A.2 indicates the software installed by us on the server. [28]

Software	Version
Apex Collateral	15.1.4
Oracle	11.2
Java	1.7
Jenkins	1.607

Table A.2: Installed software on sinv-56056.edu.hsr.ch (152.96.56.56)

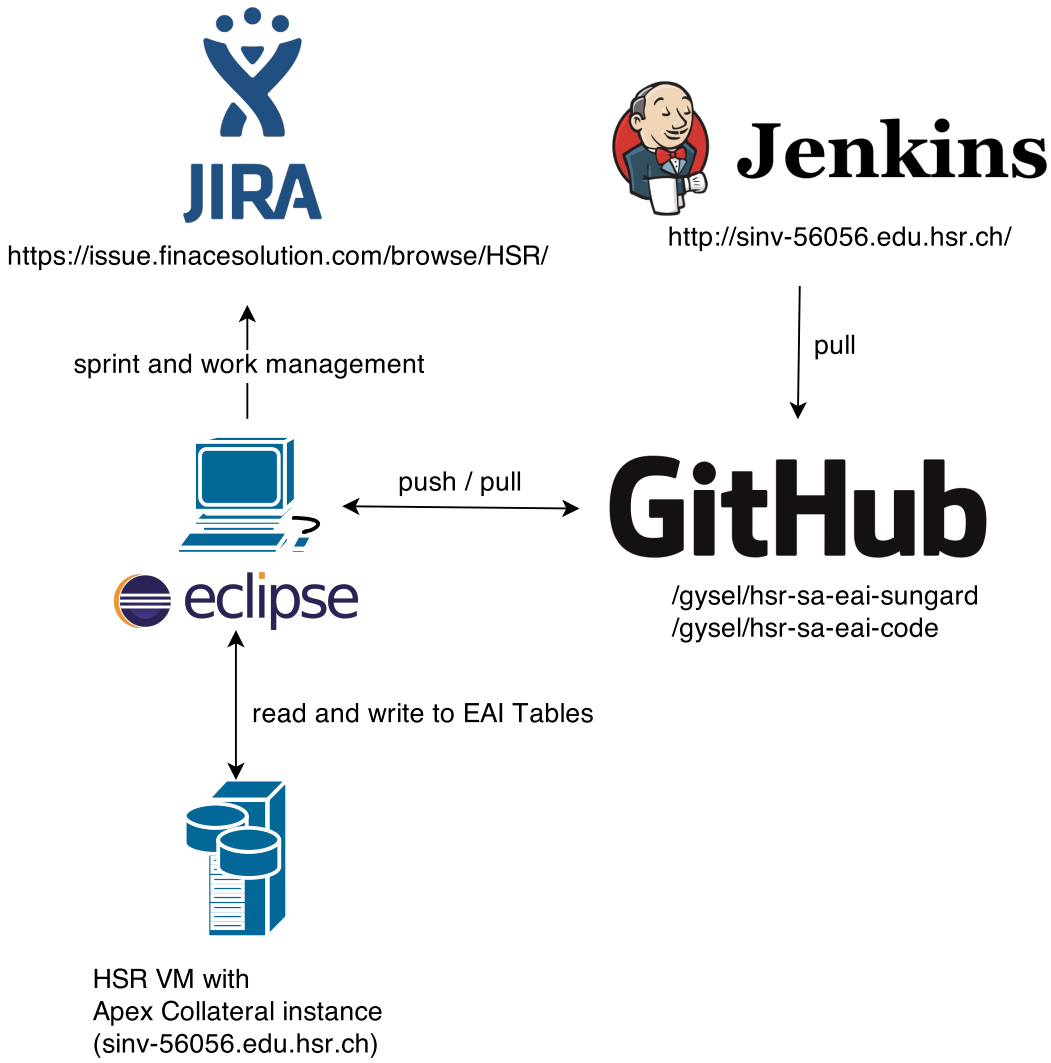


Figure A.4: Development Environment

A.3 Quality Management

To ensure a good quality of the produced software, we adopted several measures such as unit testing, code reviews and guidelines as well as CI.

A.3.1 Unit Testing

All written source code is ideally covered with JUnit[40]. Table A.3 lists the reached coverage. The Eclipse plugin EclEmma[27] was used to measure the code coverage.

Project	Coverage	Total Instructions	Remarks
camel-nomin	91%	89	
camel-jobs	77%	1474	
apex-connectivity	53%	506	Getters and setters of JPA entities are not covered.
connectivity-spirio	87%	174	
connectivity-vistabank-jee	-	-	Contains no Java code.

Table A.3: Code coverage by JUnit tests

Unit Tests are developed using the following test frameworks:

1. **JUnit**[40] as the central test framework.
2. **Mockito**[43] to mock dependencies and test units in isolation.
3. **Hamcrest**[30] to write expressive assertions.
4. **Camel**[10] **Test** to write Camel specific test cases.
5. **Spring**[53] to manage the test container for integration tests.

A.3.2 Reviews

Every coding task was moved from the JIRA status "*in progress*" to "*in review*" and then assigned to the other team member in order to verify the source code for correctness and good coding style.

A.3.3 Code Guidelines

We agreed to use the Camel Checkstyle[23] rules as our coding guidelines. The XML configuration file can be found in the Camel source code[1].

The following Checkstyle rules do not make sense in our development environment and we have therefore skipped them.

1. Tab indentation rule (we have used the Eclipse formatter)
2. Apache license header enforcement
3. Import grouping and order
4. Avoid start import (we have enhanced the whitelist with Hamcrest and Mockito)

[1] https://github.com/apache/camel/blob/master/buildingtools/camel-checkstyle.xml

A.3.4 Continuous Integration

A Jenkins installation connected to the GitHub repositories is running on the HSR project server (see Appendix A.2) . This continuous integration setup guarantees that failing test cases as well as compilation errors are noticed swiftly.

A.4 Project Plan - Sprints

As part of this thesis project a set of milestones as listed in Table A.4 were reached.

Milestone Name	Sprint	Description
Route DSL	Sprint 3	Decided not to build a new abstraction layer for Apache Camel
Data Mapper	Sprint 4	Nomin was approved as the Data Mapper
Feature Freeze	Sprint 7	Work on all product backlog items was finished
Project end	Sprint 8	Code and documentation were delivered to SunGard and HSR.

Table A.4: Milestones

We structured the thesis project into 7 Sprints to collect the requirements and build the functionality. The 8^{th} and last Sprint was exclusively used to finish the thesis document. Table A.5 lists all sprints and their functional scope.

Sprint	Start	End	Stories	Hours Worked
Sprint 1	16.02.	01.03.	Write Case Study	62
Sprint 2	02.03.	15.03.	Domain Analysis Compile and Document Requirements Project Environment First Prototype	64
Sprint 3	16.03.	29.03.	Evaluation Data Mapper Enhance Prototype	63
Sprint 4	30.03.	12.04.	Test Usability Job Scheduling	59
Sprint 5	13.04.	26.04.	Monitoring, Management and Operational Aspects	57
Sprint 6	27.04.	10.05.	Job Templates Performance Tests Batch Processing	82
Sprint 7	11.05.	24.05.	User Manual Deployment	101
Sprint 8	25.05.	29.05.	Finish and Review Thesis Document	60

Table A.5: Sprint List

Michael Gysel worked a total of 261 hours, Lukas Kölbener worked a total of 285 hours. Figure A.5 reveals the worked hours per epic.

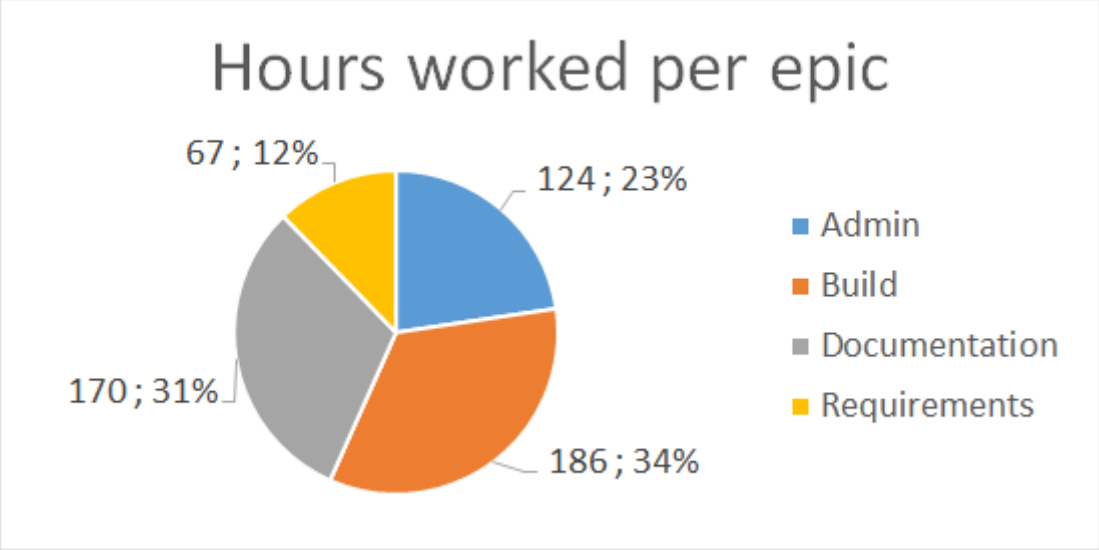


Figure A.5: Hours worked per epic

A.5 Risk Management

To assess the risk associated with this project, a comprehensive list of possible risks and their mitigations is described in the following Section.

The risk assessment shown in Table A.8 is based on a literature survey[18] taken in 2011. Two custom lists of project specific risks are documented in Table A.6 and Table A.7.

#	Risk	Impact	Evaluation & Mitigation
1	Stakeholders have opposing requirements.	It is impossible to prioritise requirements.	Discuss implementation approaches and priorities in meetings with all stakeholders present.
2	Wrong priorities are defined and unimportant features implemented first.	Important features are left out.	Validate priorities and functionality with all stakeholders on a regular basis.
3	The domain area is more complex than expected.	The implementation takes more time than expected and not all required features can be implemented.	The iterative approach helps to focus to maximize business value at all times.
4	A team member faces health issues and cannot continue to work on the project.	The project scope is impossible to fulfilled.	The project scope has to be renegotiated with SunGard and HSR.
5	Project infrastructure outage	JIRA, GitHub or Project Server goes down and the project is therefore delayed.	All components are supported by a company and professional support should therefore be available quickly.
6	Personal infrastructure failure	A personal laptop of one of the team members stops working.	HSR desktops are available and can be used as replacement hardware.

Table A.6: Project-Specific Management Risks

#	Risk	Impact	Evaluation & Mitigation
1	Functional problems with Camel	The project goal to build an integration job engine on top of Camel can not be reached.	Camel is likely able to cover most of the functional requirements as tools like Red Hat Fuse ESB[50], Apache ServiceMix[15] andTalend [55] are all based on top of Camel (see Appendix D).
2	Technical deficiencies of Camel	Analysing problems and the implementation of fixes take too much time.	Develop technical prototypes early in the project to verify the capabilities of Camel and related libraries.
3	Unstable development environment	Analysing and fixing problems takes too much time and causes a delay in the project plan.	Make use of established development tools and agree on a common version of Java, the IDE and plugins.
4	Architectural decisions introduce unnecessary complexity.	Development of functionality takes too much time and the know-how transfer to the SunGard team is difficult.	Assess every architectural decision regarding its impact towards the complexity. Where possible prefer simple solutions over rich but heavy handed libraries.
5	Developed source code is not covered with unit tests.	Future refactorings are hard to perform without appropriate test coverage. Unit tests serve as a functional documentation as they specify the intended behaviour of a piece of code.	All written software should be covered with automated unit tests using the testing capabilities of Camel and Spring[53].

Table A.7: Project-Specific Technical Risks

#	Risk	Impact	Evaluation & Mitigation
1	Misunderstanding of requirements	The final product consists of features that do not comply with the requirements of the customer.	User Stories are reviewed by the customer prior to the implementation and review meetings are held at the end of every sprint.
2	Lack of management commitment and support	The project lacks funding or resourcing.	Project management and stakeholders committed to attend review meetings whenever allowed by their schedules.
3	Lack of adequate user involvement	Requirements and solutions cannot be validated with users.	User base is small and available for interviews and reviews. Reviews to validate the progress will be performed on a regular basis.
4	Failure to gain user commitment	End users may not be able to provide required progress reviews or required contributions towards the requirement specification.	End user contributions and management reassurance will be obtained early in the project.
5	Failure to manage end user expectation	End users may not be able to use the product or refuse to do so.	The end users can influence the priorities of the user stories. This ensures that the most relevant features will be tackled first.
6	Changes to requirements	Already implemented functionality turns out to be unnecessary or based on wrong specification.	Likely to happen as the stakeholders have varying priorities and conceptions of the product to be developed. Has to be mitigated by reviewing user stories as part of the biweekly meetings. Absent stakeholders need to be informed of all decisions taken at such meetings.
7	Lack of an effective project management methodology	Project is delayed and generated business value is impacted.	The project will be managed using Scrum, a methodology that is already known to all involved parties.

Table A.8: Top Software Risks Evaluation

A.5.1 Risk Assessment

As all mitigation measurements described in the last Section were implemented and therefore none of the described risks put the project seriously at risk. The following risks impacted the project nevertheless:

Risk #4 of Table A.6 Both team members fell ill and were not able to participate in the project for several days including a sprint review meeting each. This shortage of resources however did not require renegotiation with SunGard and HSR.

Risk #1 of Table A.7 Camel does not support route definitions in multiple Spring context files, which makes it impossible to enhance to Camel routes in different projects. We were able to work around this limitation using the adviceWith feature as described in Chapter 5.

Risk #3 of Table A.8 Some important stakeholders within SunGard were often absent as they were travelling for their projects. Reviews therefore needed to be planned early and some technical reviews did not happen as it was impossible to find time when specific stakeholders were available.

All taken measures and used methodologies proved to be good choices to ensure good project management. The project did not suffer from management overhead nor were important aspect left out or forgotten during the project.

B. Evaluation of Libraries, Frameworks and Tools

This chapter outlines all evaluation decisions that influenced solution approaches. The architectural decisions in this chapter are documented using the "Y-Template"[59].

B.1 Data Mapper Evaluation

Mapping data from one format to another has been identified to be a critical feature. Therefore a careful evaluation of available data mapping tools has been conducted.

Through discussion with other engineers, research using Google's search engine, and with the help of other research results we found on the web (see [54] and [37]), a list of found data mappers has been consolidated. Some tools listed in Appendix B.1.2 have not been assessed in detail as they did not match at least one of the important criterion. The mapping features used for comparison are described in Chapter 4, Section 4.2.1.

Evaluation has shown that most tools belong to one of two categories. The first contains Java mapping tools to map from one Java object to another with mapping definitions in a separated configuration file. These tools can easily be integrated into Java processes and Camel routes but lack extensive mapping features and Frank Frontline friendly configuration syntax. The second half are more sophisticated integration suites or Extract Transform Load (ETL) tools — so called Doodleware[1]. They offer powerful graphical editors for mappings and support many of the advanced features listed in Chapter 4, Section 4.2.1. Unfortunately, these Doodleware systems are either standalone software or part of bigger proprietary integration suites and therefore troublesome to integrate with Camel routes.

All evaluated tools show one of these major disadvantages. In a second research cycle a different approach was found, which is to use a DSL for mappings. A library called Nomin which uses a Groovy DSL has been evaluated and proofed to satisfy all important criteria as shown in Appendix B.4.

[1] Visual development environment for integration jobs or transformation logic. The term was introduced by Jason Boro and further discussed by Gregor Hohpe[34].

B.1.1 Evaluation Details

This Section documents the detailed evaluation of the investigated data mappers.

Dozer and Data Mapper

Dozer[26] is an open source Java bean to bean mapper. It allows to map one entity to another, using a provided configuration. Data type conversion and nested objects are handled by Dozer. With the Data Mapper[25] JBoss provides some enhancements and an integration into their JBoss Developer Studio[49] to Dozer. Table B.1 outlines the result of the evaluation.

	Dozer	**Data Mapper**
Mapping features		
Calculations on values	Java, expressions	Java, expressions
Default values	Yes	Yes
Code mappings	No	No
Type conversions	Yes	Yes
Multi-record conditions	No	No
Conditional mappings	No	No
Multiple sources / targets	No	No
Maps (Key,Value), nested mappings	Yes	Yes
Usability		
User Interface	XML or Java	JBoss Developer Studio
Java extensibility	Yes	Yes
Integration		
Formats	XML, Java	XML, JSON, Java
Camel integration	Yes	Yes
Miscellaneous		
Complexity	Low	Low
Price	Free	Free
License	Apache 2.0 License	Apache 2.0 License
Technology stack	Java	Java, Dozer
Testability	Yes	Yes
Changes without compilation	Yes	Yes

Table B.1: Evaluation of Dozer and Data Mapper

Dozer is easy to use and has an active user base on their mailing list. However functional development has almost stopped and the support of default values, static values and code mappings is missing or cumbersome to use. *Data Mapper* provides a graphical user interface for data mapping and adds some new features like generating Java-based models

from XML or JSON schemas. The main gaps of Dozer are still present. We therefore concluded that Dozer does not fit the requirements.

MapForce by Altova

Table B.2 shows the capabilities of MapForce by Altova. MapForce is proprietary tool which runs standalone with a sophisticated user interface. It allows data transforming from and to many different formats.

	MapForce
Mapping features	
Calculations on values	Yes
Default values	Yes
Code mappings	Yes
Type conversions	Yes
Multi-record conditions	Yes
Conditional mappings	Yes
Multiple sources / targets	Yes
Maps (Key,Value), nested mappings	Yes
Usability	
User Interface	Graphical, job export as Java
Java extensibility	No, closed system
Integration	
Formats	XML, DB, EDI, File, WSDL, XLS, JSON **(missing: Java, JMS, REST)**
Camel integration	No (Implementation of a specific endpoint to integrate exported mappings is possible.)
Miscellaneous	
Complexity	High
Price	799$ for Enterprise Edition, 399$ for Professional Edition
License	Altova License
Technology stack	Proprietary
Testability	-
Changes without compilation	Yes but Camel integration requires jobs to be exported again.

Table B.2: Evaluation of MapForce

Featurewise MapForce is powerful enough to cover all requirements. However the missing integration with Camel, the lack of support for Java objects and JMS messages are major obstacles. MapForce is a properitary tool and costs 799$ per license. Floating licenses are available starting from 4'000$.

We recommend not to use MapForce as it is a fully fledged ETL tool and not intended to be used as a data mapping component in Camel routes.

Talend Open Studio

Table B.3 shows the capabilities of the Talend Open Studio for Data Integration. Like MapForce, Talend Open Studio is a proprietary software providing a sophisticated standalone user interface. Talend Open Studio integrates with many other EAI tools provided by Talend.

	Talend Open Studio
Mapping features	
Calculations on values	Yes
Default values	Yes
Code mappings	Yes
Type conversions	Yes
Multi-record conditions	Yes
Conditional mappings	Yes
Multiple sources / targets	Yes
Maps (Key,Value), nested mappings	Yes
Usability	
User Interface	Talend Open Studio (based on Eclipse)
Java extensibility	Yes (Java-based mapping enhancement)
Integration	
Formats	All
Camel integration	No. (Jobs can be exported as jar or OSGi bundle, but OSGi bundles require the Talend environment to run properly.)
Miscellaneous	
Complexity	Medium
Price	Free (Community Edition)
License	Apache v2, LGPL and GPLv2 License (depending on component)
Technology stack	Open Source Technologies
Testability	Only in Enterprise version
Changes without compilation	Yes but Talend requires jobs to be exported again.

Table B.3: Evaluation of Talend Open Studio

Talend Open Studio has a sophisticated user interface for creating mappings and routes. It supports all needed connectors for different formats, but lacks integration possibilities to Camel. The open source version is a Community Edition which implies problems described in Chapter 4, Section 4.3.11. Furthermore, exported jobs are difficult to integrate into solutions other than Talend products. For example it is not possible to use them in a Camel-based integration toolkit.

Nomin

Groovy is a script language that runs within a JVM. Mappings written in Groovy can therefore be changed without a need to recompile them. Furthermore they run seamlessly

in the JVM and do not need a separate infrastructure. Nomin[46] is a Java mapping framework based on a Groovy DSL and supports almost all of the required features as described in Table B.4.

A well-designed DSL can enable stakeholders like Frank Frontline 4.1.2 to write code themselves and can become a shared living specification of the data mapping system [1, p. 12 - 13].

	Nomin
Mapping features	
Calculations on values	Yes
Default values	Yes
Code mappings	Yes
Type conversions	Yes
Multi-record conditions	No
Conditional mappings	Yes
Multi sources / targets	Yes
Maps (Key,Value), nested mappings	Yes
Usability	
User Interface	Groovy scripts
Java extensibility	Yes (Java code can be used)
Integration	
Formats	Key/Value, Java objects [2]
Camel integration	No (Custom endpoint can be written with little effort.)
Miscellaneous	
Complexity	Medium
Price	Free
License	Apache v2
Technology stack	Java, Groovy
Testability	Yes
Changes without compilation	Yes

Table B.4: Evaluation of Nomin

B.1.2 Other Data Mappers

Apart from the tools analysed in the detailed comparison, the following tools were taken into consideration but no comprehensive analysis has been performed.

[2] All other formats can be supported through Camel endpoints.

- **Smooks**[52] is an extensible framework for processing XML and non-XML which can also be used for data transformations.
  The learning curve of Smooks is too steep to enable users like Frank Frontline to make changes to mappings. The Eclipse plugin has been decommissioned since August 2011.
- **Liquid Studio**[41] is a graphical XML mapping tool but does not support to map other formats.
- **Jamper**[35] is a Java-based application for creating XML transformation rules. It can not be used as a transformation engine nor map other formats than XML
- Oakland's **Data Transformer**[47] is a Java-based data transformation tool with a graphical user interface. There was no trial version available and the project has not been updated since 2011.
- **BeanIO**[20] is a Java bean binding framework to marshal and unmarshal Java beans from different formats. BeanIO does not support mapping from one bean to another.
- **Apache Camel Bindy**[9] is a Java bean binding framework for unstructured data. Based on annotations it does not allow mapping changes without recompilation and does not support mapping between beans.
- **Apache Common BeanUtils**[11] is a bean mapping and morphing library. With mapping logic written in Java it does not support mapping changes without recompilation.
- **MapStruct**[42] is a bean mapping and morphing library. Based on annotations it does not support mapping changes without recompilation.
- **EZMorph**[29] is a bean mapping and morphing library. With mapping logic written in Java it does not support mapping changes without recompilation.
- **Mule Studio**[45] is designed to be used with a Mule ESB and data mappings cannot be extracted from its environment.
- **Pentaho Kettle** is a powerful graphical ETL tool using a metadata-driven approach.
  Kettle requires its own runtime and can not be integrated into Camel routes.
- **ModelMapper**[44] is a bean mapping library. With mapping logic written in Java it does not support mapping changes without recompilation.

B.1.3 Detailed Nomin Evaluation

The sample implementation of a data mapping using Nomin shown in Listing B.1 has been validated with the stakeholders and proved to satisfy functionality and usability.

Listing B.1: Example of a data mapping with Nomin

```
1 import com.sungard.apexcollateral.integration.entity.*
2 import java.sql.Date
3 import java.sql.Timestamp
```

```
4  import java.math.BigDecimal
5  import java.util.Map
6  import java.text.SimpleDateFormat
7
8  mappingFor a: java.util.Map, b: ExternalExposure
9  // generic fields
10 b.eaiFunction = 4
11
12 // exposure fields
13 b.ownerId = a['principal']
14 b.counterpartyBranch = a['counterparty']
15 b.sourceName = a['system']
16 b.externalExposureType = a['product']
17 b.tradeExposureId = a['tradeRef']
18 b.tradeDate = a['dealDate']
19 dateFormat "yyyy-MM-dd"
20 b.valueDate = a['startDate']
21 dateFormat "yyyy-MM-dd"
22 b.maturityDate = a['endDate']
23 dateFormat "yyyy-MM-dd"
24 b.strikePrice = {a['strikePrice'] ?: 0}
25 b.buySellIndicator = a['tradeAction']
26 b.callPutIndicator = a['tradeActionOther']
27 b.notionalValue = a['notional']
28 b.notionalCurrency = a['currency']
29 b.exposureValue = a['mtmValuation']
30 b.valuationDate = a['mtmValuationDate']
31 dateFormat "yyyy-MM-dd"
32 b.exposureCurrency = a['mtmValuationLocalSysCcy']
33
34 // static fields
35 b.agreementId = "1234567890"
36 b.counterpartyId = "CS"
37 b.bookingEntity = "Collateral Management Book"
38 b.inFeed = 1
39 b.externalProductType = "FX"
40 b.mtmValueUsd = 0
```

We conducted review meetings on the example mapping with Nomin with different stakeholders from SunGard as documented in the subsequent Sections.

Meeting with Frank Frontline

Date: 31.3.2015

Attendees: Michael Gysel, a SunGard employee representing Frank Frontline.

Meeting minutes:

- Frank is capable to write mappings using the DSL.

- He needs file-based value lookups as some of them are based on large value tables. The value lookup integrated in Nomin is not sufficient. The existing solution provides lookups with values defined in text files. (e.g. `1=TYPE-A`)
- Frank is able to read the XML DSL to define the routes. He would likely be able to customise them as well.
- The ability to change mappings without recompiling the jar files is crucial to Frank. Using script files to map data allows Frank to change even advanced mappings. This is very useful to tailor data integration logic in a customer environment.

Meeting with Alex Allrounder

Date: 30.3.2015

Attendees: Michael Gysel, Lukas Kölbener, a SunGard employee representing Frank Frontline.

Meeting minutes:

- The DSL is exactly what Alex needs.
- XSLT is a requirement as well. however this would not be handled in Nomin but rather as an integrated camel component.
- The same row needs to be sent to several targets. This is possible using a multicast in Camel.
- A row needs to be duplicated and some small adjustments need to be done. The two rows need to be sent to the same target. This might be possible using a splitter or a custom camel component. Either of the solutions are acceptable to Alex.

Meeting with Sungard Architect

As soon as the functional match was verified, Nomin was suggested to the Apex Collateral product architect. His evaluation of Nomin from a technical point of view is documented in Table B.5.

Good	**Bad**
Reasonable, small codebase.	No activity (neither on GitHub nor on the mailing list).
Good functional documentation.	Mixture of Java and Groovy.
Most of the classes are small and do just one thing, good use of interfaces, keeping things simple.	As far as I was able to tell, not that much unit test coverage.
Some reasonable functional / integration tests that test mapping functionality.	
Mockito is used in some tests.	
Summary: All in all I'd say the project looks to be in a decent shape but could use someone actually taking care of it.	

Table B.5: Nomin assessment of a SunGard architect

B.1.4 Decision for Nomin

In the context of *evaluating a data mapping tool* facing *the requirement of configurable mappings*, we decided for *Nomin, a Groovy DSL-based tool,* and neglected *commercial ETL tools, as well as configuration-based data mapping tools,* to achieve *flexible but powerful mapping functionality* accepting *the need to learn a DSL for the users.*

B.2 Data Persistence

The database tables that need to be accessed can be categorised into two kinds:

- **EAI Tables** with a known set of columns forming the standardised way of importing and exporting data from and into Apex Collateral.
- **Customer specific tables** with a format that is unique to one client installation.

B.2.1 Interface Tables

In the context of *access to standardised interface tables* facing *the requirement of documented and reusable persistance access*, we decided for *JPA-based database access* and neglected *SQL-based database access* to achieve *code reusability* accepting *the need for additional application libraries.*

B.2.2 Customer Specific Integration Tables

In the context of *access to customer specific tables* facing *the requirement of loose coupling*, we decided for *SQL-based database access* and neglected *JPA-based database access* to achieve *less development effort* accepting *the need to write SQL statements.*

B.3 Application Context and Dependency Injection

In the context of *application frameworks* facing *the need for a Dependency Injection container*, we decided for *the Spring Framework* and neglected *lightweight containers* to achieve *conformity with the Apex Collateral product and better integration with Camel* accepting *the increased complexity.*

B.4 Deployment

In the context of *production deployments* facing *the need for an application container*, we decided for *a plain Java process* and neglected *OSGi* to achieve *a simplified deployment* accepting *that dynamic loading and disposal of application components is not possible.*

OSGi was not evaluated in detail as no functional or non-functional requirement indicated that dynamic loading and disposal of application components was required.

B.5 Build Automation

In the context of *the development environment* facing *the need for build automation*, we decided for *Apache Maven* and neglected *other tools* to achieve *less time spent on evaluation* accepting *that there might have been a better tool.*

C. Project Definition

The following project definition is a translated excerpt of the original project definition written in German.

C.0.1 Goals

The following goals are to be met:

1. Analyse the current problems, limitations and code smells of the existing EAI Server and integration modules as part of a case study.
2. Collect requirements from stakeholders and document them as user stories.
3. Define the scope of the new solution following an agile approach (collaborative prioritisation of user stories).
4. Implement and test a reference implementation of at least one existing job in a specified reference environment.

C.0.2 Deliverables

The following items have to be created and delivered:

1. Document describing the results of the analysis.
2. Compiled functional and non-functional requirements aligned with the stakeholders needs.
3. Design and implementation according to the defined user stories.
4. Design, implementation, and documentation of the reference implementation.
5. Suitable software documentation of all developed components (JavaDoc, UML, test cases, context overview).

C.0.3 Success Metrics

A successful implementation accommodates the following metrics:

1. Established open source solutions provide the basis of the new EAI Server (e.g. Apache Camel, Apache ActiveMQ, Apache CXF).
2. An additional layer or components enhance or complement the chosen open source technologies to simplify the definition of integration jobs.

C.0.4 Environment

Apex Collateral is based on a Java technology stack and uses several open source solutions like the Spring Framework[53], Hibernate[33], Quartz[48], ActiveMQ[7] amongst many others. It can be operated using an Oracle or Microsoft database server.

D. Results Prestudy by SunGard

The following tables summarise the results of the prestudy performed by SunGard. Please note that it is slightly shortened and desensitised. The prices are only represented as price ranges. None of the mentioned vendors provides quotes on their websites and SunGard had to request them.

		MuleSoft	
		Community Edition	Enterprise Edition
1	**Framework & Tooling**		
1.5	**Management & Monitoring**	Java Instrumentation (JMX)	Mule Management Console High Availability (Platin only*) Performance monitoring Operational control (Deploy) Java Instrumentation (JMX)
1.4	**Development & Tooling**	Anypoint Studio (Eclipse IDE)	Anypoint Studio (Eclipse IDE) DataMapper w/ DataSense Batch Processing Module Visual Debugging
1.3	**Application Layer**	Mule ESB (proprietary)	Mule ESB (proprietary)
1.2	**Services Layer**	Mule ESB (proprietary)	Mule ESB (proprietary)
1.1	**Kernel Layer**	Mule ESB (proprietary)	Mule ESB (proprietary)
2	**Others**		
2.1	**Open-Source**	YES	NO
2.2	**License-Model** **Partnership-Model**	?	VAR (Value-Added Reseller) OEM (>500'000 EUR/year)
2.3	**Price range**	FREE	10'000$ - 30'000$

		Apache ServiceMix	RedHat JBoss Fuse	Talend	
				Community Edition	Enterprise Edition
1	**Framework & Tooling**				
1.5	**Management & Monitoring**	-	JBoss Tools: Fuse Fabric8, Server Tooling, ...	Talend ESB SE	Talend Enterprise ESB Activity Monitoring Console, Talend Administration Center, High availability, load balancing, failover
1.4	**Development & Tooling**	-	JBoss Developer Studio (Eclipse IDE) JBoss Tools Integration Stack (Camel Tools, jBPM, Drools, ...)	Talend Open Studio	Talend Enterprise ESB Testing, debugging and tuning, Interactive data viewer
1.3	**Application Layer**				
1.2	**Services Layer**	Apache ActiveMQ Apache Camel Apache CXF	Apache ActiveMQ Apache Camel Apache CXF (RedHat Enterprise Ready)	Apache ActiveMQ Apache Camel Apache CXF	Apache ActiveMQ Apache Camel Apache CXF
1.1	**Kernel Layer**	Apache Karaf	Apache Karaf	Apache Karaf Apache Cellar	Apache Karaf Apache Cellar
2	**Others**				
2.1	**Open-Source**	YES	YES	YES	YES (core), NO (Enterprise features)
2.2	**License-Model**	Apache	ISV	?	VAR (Value-Added Reseller)
	Partnership-Model		1) Ready 2) Advanced		OEM
2.3	**Price range**	FREE	1'000$ - 10'000$	FREE	10'000$ - 20'000$

Glossary

Descriptions of Glossary items have mainly been taken from Wikipedia.org.

Apache Maven Apache Maven is a software project management and comprehension tool. 33, 36, 52

Apache Tomcat A lightweight open source implementation of the Java Servlet technology. 52

API Application Program Interface. 10, 29, 47, 94

Camel context The main Camel component used to build and execute routes. 40

CI Continuous Integration. 66, 69, 70

CSV comma-separated values. 10, 11, 14, 17, 61

Dependency Injection A design pattern that implements inversion of control for software libraries. The caller delegates the discovery of dependencies to the framework 5, 88

DSL Domain Specific Language. 38, 40, 41, 62, 72, 78, 83

EAI Enterprise Application Integration. 4, 8, 10–13, 81

EAI Tables A fixed set of database tables used to import and export data from and to Apex Collateral. 26

EIP Enterprise Integration Pattern, defined by Hohpe/Woolf[3]. 8, 15, 23

ESB Enterprise Service Bus. 5, 8, 9, 12, 84

ETL Extract Transform Load. 78, 81, 84

FTP File Transfer Protocol. 10, 58, 61

Groovy A scripting language for the JVM. 42, 78

HSR Hochschule für Technik Rapperwswil. 35, 67–69

HTTP Hypertext Transfer Protocol, an application protocol and the foundation of data communication for the World Wide Web [57] 24, 44, 53, 54, 94

HTTPS Secured HTTP 27, 59

JAR Java Archive. 60

JDBC Java DataBase Connectivity, a standardised API to connect to a relational database 24, 52

JMS Java Message Service, a Java message-oriented middleware application programming interface for sending messages between two or more clients. 5, 24, 25, 27, 35, 38, 44, 45, 55, 58, 59

JMX Java Management Extensions, a Java technology to manage and monitor JVM based applications. 44, 45, 55

JSON JavaScript Object Notation. 17, 44, 54, 58

JVM Java Virtual Machine. 54, 82, 83, 94

ORM Object-Relational Mapping. 5

OSGi A modular system and service platform for the Java language. 88

PS Professional Services. 13, 16, 27

REST REpresentational State Transfer, an architectural style for creating web services. Typically communicates over HTTP. 24, 53, 58

SEDA Staged Event-Driven Architecture, documented by Matthew David Welsh in 2002. 49, 51

SFTP Secure File Transfer Protocol. 27, 58, 59

SOAP Originally an acronym for Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of web services using XML. 58

Spring An open source application framework and inversion of control container for the Java platform. 33, 37, 39, 42, 88

SWIFT Message A message containing financial data, standardised by the Society for Worldwide Interbank Financial Telecommunication (SWIFT). 30

VM Virtual Machine. 69

WAR Web application ARchive. 60

XML eXtensible Markup Language. 10, 17, 30, 37, 44, 58, 61, 62, 71, 84

References

Literature

[1] Fergal Dearle. *Groovy for Domain-Specific Languages*. 2010-06-01 (cit. on p. 83).

[2] Robert Hanmer. *Patterns for Fault Tolerant Software*. 2007-11-28 (cit. on p. 29).

[3] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns*. 2003-10-10 (cit. on pp. 1, 6, 8, 9, 12, 13, 15, 25, 93).

[4] Claus Ibsen and Jonathan Anstey. *Camel in Action*. 2010-10-01 (cit. on p. 6).

[5] Bertrand Meyer. *Object-Oriented Software Construction*. 1998-11-02 (cit. on p. 28).

[6] Matthew David Welsh. "An Architecture for Highly Concurrent, Well-Conditioned Internet Services". PhD thesis. University of California at Berkeley, 2002 (cit. on p. 49).

Online Sources

[7] *ActiveMQ. An open source messaging server*. URL: http://activemq.apache.org/ (visited on 2015-03-18) (cit. on pp. 5, 55, 90).

[10] *Apache Camel. A rule-based routing and mediation engine which provides a Java object-based implementation of the Enterprise Integration Patterns*. URL: http://camel.apache.org/ (visited on 2015-04-13) (cit. on pp. 5, 9, 36, 55, 71).

[8] *Apache Camel AdviceWith Feature*. URL: http://camel.apache.org/advicewith.html (visited on 2015-05-10) (cit. on pp. 41, 63).

[9] *Apache Camel Bindy. Allow the parsing/binding of non-structured data (non-XML data) to and from Java Beans using annotations*. URL: http://camel.apache.org/bindy.html (visited on 2015-03-30) (cit. on p. 84).

[11] *Apache Commons BeanUtils. A bean mapping and morphing library*. URL: http://commons.apache.org/proper/commons-beanutils/ (visited on 2015-03-30) (cit. on p. 84).

[12] *Apache Maven Shade Plugin*. URL: https://maven.apache.org/plugins/maven-shade-plugin/ (visited on 2015-05-10) (cit. on p. 52).

[13] *Apache Maven WAR Plugin*. URL: https://maven.apache.org/plugins/maven-war-plugin/ (visited on 2015-05-10) (cit. on p. 54).

[14] *Apache POI. A Java API for Microsoft Documents*. URL: https://poi.apache.org/ (visited on 2015-05-11) (cit. on p. 5).

[15] *Apache ServiceMix. An open-source integration container*. URL: http://servicemix.apache.org/ (visited on 2015-05-23) (cit. on pp. 5, 75).

[16] *Apache Synapse. A lightweight Enterprise Service Bus*. URL: http://synapse.apache.org/ (visited on 2015-05-26) (cit. on p. 5).

[17] *Apex Collateral. A solution for collateral management, optimization and trading*. URL: http://financialsystems.sungard.com/solutions/collateral-management/apex-collateral (visited on 2015-02-23) (cit. on p. 4).

[18] Tharwon Arnuphaptrairong. *Top Ten Lists of Software Project Risks*. English. 2011-03. URL: http://www.uio.no/studier/emner/matnat/ifi/INF5181/h14/pensumliste/microsoft-word---iaeng-top-ten-lists-of-software-project-risk1---imecs2011_pp732-737.pdf (visited on 2015-03-09) (cit. on p. 74).

[19] *Atlassian Jira. A project tracking software*. URL: https://www.atlassian.com/software/jira (visited on 2015-05-19) (cit. on pp. 65, 66).

[20] *BeanIO. An open source Java framework for marshalling and unmarshalling Java beans from different formats*. URL: http://beanio.org/ (visited on 2015-03-30) (cit. on p. 84).

[21] *Camel RouteContext feature*. URL: http://camel.apache.org/how-do-i-import-routes-from-other-xml-files.html (visited on 2015-05-26) (cit. on p. 58).

[22] *Certified ScrumMaster (CSM)*. URL: https://www.scrumalliance.org/certifications/practitioners/certified-scrummaster-csm (visited on 2015-05-19) (cit. on p. 68).

[23] *Checkstyle. A tool to enforce Java coding style*. URL: http://checkstyle.sourceforge.net/ (visited on 2015-05-25) (cit. on p. 71).

[25] *Data Mapper. A Camel component to create, configure, and test data transformation*. URL: https://github.com/fabric8io/data-mapper/ (visited on 2015-03-24) (cit. on p. 79).

[26] *Dozer. A Java bean to bean mapper*. URL: http://dozer.sourceforge.net/ (visited on 2015-03-24) (cit. on p. 79).

[27] *EclEmma. An Eclipse plugin to measure Java code coverage*. URL: http://www.eclemma.org/ (visited on 2015-05-25) (cit. on p. 71).

[28] *Eclipse Luna. An integrated development environment for Java*. URL: https://www.eclipse.org/luna/ (visited on 2015-05-19) (cit. on p. 69).

[29] *EZMorph. A Java library to transform a Java object to another object*. URL: http://ezmorph.sourceforge.net/ (visited on 2015-03-30) (cit. on p. 84).

[30] *Hamcrest. A set of expressive matchers to be used in unit tests.* URL: http://hamcrest.org/ (visited on 2015-05-25) (cit. on p. 71).

[31] *Hawt.io. A modular web console to mange Java application.* URL: http://hawt.io/ (visited on 2015-05-28) (cit. on p. 54).

[32] *Hibernate Envers is a Hibernate module to preserve the history of changes of JPA entities.* URL: http://hibernate.org/orm/envers/ (visited on 2015-04-12) (cit. on p. 31).

[33] *Hibernate ORM. Domain model persistence for relational databases.* URL: http://hibernate.org/ (visited on 2015-03-18) (cit. on pp. 5, 90).

[34] Gregor Hohpe. *Hohpe Ramblings: Doodleware.* 2003-11-01. URL: http://www.eaipatterns.com/ramblings/02_doodleware.html (visited on 2015-04-07) (cit. on p. 78).

[35] *Jamper. A Java-based application for creating XML transformation rules.* URL: http://jamper.sourceforge.net/ (visited on 2015-03-30) (cit. on p. 84).

[36] *JasperReports. An open source reporting engine.* URL: http://community.jaspersoft.com/project/jasperreports-library (visited on 2015-05-11) (cit. on p. 5).

[37] *Javacodegeeks: List of Data Mapping Tools.* URL: http://www.javacodegeeks.com/2013/10/java-object-to-object-mapper.html (visited on 2015-03-16) (cit. on p. 78).

[38] *Jenkins. A Continuous Integration tool.* URL: https://jenkins-ci.org/ (visited on 2015-05-19) (cit. on p. 66).

[39] *Jolokia. An agent to expose JMX MBeans over HTTP.* URL: https://jolokia.org/ (visited on 2015-05-28) (cit. on p. 54).

[40] *JUnit. A Java framework to write repeatable tests.* URL: http://junit.org/ (visited on 2015-05-25) (cit. on p. 71).

[41] *Liquid Studio. A data mapping and xml mapping tool.* URL: http://www.liquid-technologies.com/xmldatamapper.aspx (visited on 2015-03-30) (cit. on p. 84).

[42] *MapStruct. A code generator which greatly simplifies the implementation of mappings between Java bean types-based on a convention over configuration approach.* URL: http://mapstruct.org/ (visited on 2015-03-30) (cit. on p. 84).

[43] *Mockito. A mocking framework for unit tests.* URL: http://mockito.org/ (visited on 2015-05-25) (cit. on p. 71).

[44] *Model Mapper. A Java bean to bean mapper.* URL: http://modelmapper.org/ (visited on 2015-03-24) (cit. on p. 84).

[45] *Mule Studio. A Java ESB that supports data mapping.* URL: https://www.mulesoft.com/studio (visited on 2015-03-30) (cit. on pp. 5, 84).

[46] *Nomin. A mapping engine for the Java platform transforming object trees according to declarative mapping rules.* URL: https://github.com/dobrynya/nomin (visited on 2015-03-30) (cit. on pp. 34, 36, 42, 83).

[47] *Oakland Data Transformer. A Java-based data transformation tool with a graphical user interface to define the mapping rules.* URL: http://www.oaklandsoftware.com/data-transformer (visited on 2015-03-30) (cit. on p. 84).

[48] *Quartz. A job scheduler.* URL: http://quartz-scheduler.org/ (visited on 2015-03-18) (cit. on pp. 5, 47, 55, 90).

[49] *Red Hat JBoss Developer Studio.* URL: http://www.jboss.org/products/devstudio/overview/ (visited on 2015-03-24) (cit. on p. 79).

[50] *Red Hat JBoss Fuse.* URL: http://www.redhat.com/en/technologies/jboss-middleware/fuse (visited on 2015-05-11) (cit. on pp. 5, 75).

[51] *Scrum images by Mountain Goat Software.* URL: http://www.mountaingoatsoftware.com/agile/scrum/images (visited on 2015-05-19) (cit. on p. 65).

[52] *Smooks. An extensible framework for processing XML and non XML.* URL: http://www.smooks.org/ (visited on 2015-03-30) (cit. on p. 84).

[53] *Spring Framework. Supports dependency injection, transaction management, web applications, data access, messaging, testing and more.* URL: http://projects.spring.io/spring-framework/ (visited on 2015-03-18) (cit. on pp. 5, 36, 71, 75, 90).

[54] *Stackoverflow: Data Mapping Tools.* URL: http://stackoverflow.com/questions/1432764/any-tool-for-java-object-to-object-mapping (visited on 2015-03-16) (cit. on p. 78).

[55] *Talend Data Integration. An Eclipse-based ETL tool.* URL: https://www.talend.com/products/data-integration (visited on 2015-03-30) (cit. on pp. 5, 75).

[56] *The Simple Logging Facade for Java (SLF4J) serves as a simple facade or abstraction for various logging frameworks allowing the end user to plug in the desired logging framework at deployment time.* URL: http://www.slf4j.org/ (visited on 2015-04-12) (cit. on pp. 31, 60).

[24] Kai Wähner. *Choosing the Right ESB for Your Integration Needs.* URL: http://www.infoq.com/articles/ESB-Integration (visited on 2015-02-20) (cit. on pp. 8, 9).

[57] *Wikipedia: Hypertext Transfer Protocol.* URL: http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol (visited on 2015-05-09) (cit. on p. 93).

[58] *WSO2 Carbon. An OSGi-based middleware product.* URL: http://checkstyle.sourceforge.net/ (visited on 2015-05-26) (cit. on p. 5).

[59] Olaf Zimmermann. *Making Architectural Knowledge Sustainable: The Y-Approach. SATURN Conference. Software Engineering Institute CMU.* 2012-05-07. URL: http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=31345 (visited on 2015-04-05) (cit. on pp. 39, 78).