Web Service Interface for Legacy Virtual Product Lifecycle Management System

Sathyavakeeswaran Krishnan
Abstract

Web Service Interface for Legacy Virtual Product Lifecycle Management System

Sathyavakeeswaran Krishnan

Enterprises have deployed Legacy Systems to carry out the mission critical business processes within its current IT infrastructure. Legacy Systems being primary assets of the organization, the key challenge of such systems is reusing and leveraging the valuable business logic that resides within these applications. Despite the significant business value provided by the legacy systems, to conform to the changing business needs demands transition of these legacy systems towards modern architecture. Service Oriented Architecture (SOA) an architectural approach helps enterprises in reuse of existing assets as well as addressing increasing business demands. An effective way of migration of legacy systems to SOA is through exposing their functionality as services. SOA approach constitutes various advantages that employ standard, platform independent, well-defined service interfaces, which is suitable for realizing enterprise application integration with other systems. There has been a growing interest in the legacy system to SOA modernization among the academic and industrial research community. The methods proposed by this community primarily focus on how the critical business logic of the legacy system is migrated i.e. how to convert the functionality of existing systems into services to gain the benefits of SOA. This research focuses on investigating the feasibility of providing web service access to the data stored in legacy vPLM system. It also involves developing a prototype taking application integration best practices, Scania IT SOA guidelines and metrics such as reusability, performance and security into account.
ACKNOWLEDGEMENTS

My sincere thanks to my thesis supervisor Magnus Persson at Scania for his patience in supporting continuously and guiding me with this project.

I would like to express my sincere gratitude to my supervisor Justin Pearson at Uppsala University for his constant support in reviewing my master thesis and offering valuable suggestions and indispensable recommendations.

Special thanks to Rodrigo Cid, Camilla Arnersten, Madhu Cheluvaraju, Sreenivas Velayudhan and Olle Sundblad for their support during the thesis. Also thanks to all employees at the IWJP department, we interviewed during our work and for all interesting discussions around the coffee table.

Sincere thanks to my mom and brother for their love, endless support and encouragement throughout the thesis work. Finally, special thanks to my friends who have always been supporting and loving.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>ACKNOWLEDGEMENTS</th>
<th>..................................................................................</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>..................................................................................</td>
<td>4</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>..................................................................................</td>
<td>5</td>
</tr>
<tr>
<td>CHAPTER 1</td>
<td>..................................................................................</td>
<td>6</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>..................................................................................</td>
<td>6</td>
</tr>
<tr>
<td>1.1 Background</td>
<td>..................................................................................</td>
<td>6</td>
</tr>
<tr>
<td>1.2 Problem motivation</td>
<td>..........................................................................</td>
<td>7</td>
</tr>
<tr>
<td>1.3 Central goal of the thesis</td>
<td>..........................................................................</td>
<td>8</td>
</tr>
<tr>
<td>1.4 Overview</td>
<td>..................................................................................</td>
<td>8</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>..................................................................................</td>
<td>9</td>
</tr>
<tr>
<td>THEORETICAL FRAMEWORK</td>
<td>.......................................................................</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Understanding SOA</td>
<td>..................................................................................</td>
<td>9</td>
</tr>
<tr>
<td>2.1.1 Basic SOA concepts</td>
<td>...........................................................................</td>
<td>10</td>
</tr>
<tr>
<td>2.1.2 Principles of SOA</td>
<td>...........................................................................</td>
<td>12</td>
</tr>
<tr>
<td>2.1.3 Benefits of SOA</td>
<td>..................................................................................</td>
<td>13</td>
</tr>
<tr>
<td>2.1.4 SOA realization</td>
<td>...........................................................................</td>
<td>14</td>
</tr>
<tr>
<td>2.1.5 SOAP vs. REST</td>
<td>...........................................................................</td>
<td>16</td>
</tr>
<tr>
<td>2.1.6 Non-functional attributes of SOA</td>
<td>...................................................................</td>
<td>17</td>
</tr>
<tr>
<td>2.2 Understanding ENOVIA V5 system components</td>
<td>.....................................................................</td>
<td>18</td>
</tr>
<tr>
<td>2.2.1 CATIA V5</td>
<td>...........................................................................</td>
<td>19</td>
</tr>
<tr>
<td>2.2.2 ENOVIA V5 VPLM</td>
<td>...........................................................................</td>
<td>20</td>
</tr>
<tr>
<td>2.2.3 ENOVIA V5 LCA</td>
<td>...........................................................................</td>
<td>20</td>
</tr>
<tr>
<td>2.2.4 ENOVIA V5 VPM Navigator</td>
<td>...........................................................................</td>
<td>21</td>
</tr>
<tr>
<td>2.2.5 ENOVIA VPM People and Organization</td>
<td>.....................................................................</td>
<td>23</td>
</tr>
<tr>
<td>2.3 Legacy system challenges</td>
<td>...........................................................................</td>
<td>23</td>
</tr>
<tr>
<td>2.4 SOA challenges</td>
<td>...........................................................................</td>
<td>24</td>
</tr>
<tr>
<td>2.5 Challenges in migration of ENOVIA Systems to SOA</td>
<td>.....................................................................</td>
<td>24</td>
</tr>
<tr>
<td>2.6 Migration of Legacy Systems to SOA</td>
<td>...........................................................................</td>
<td>25</td>
</tr>
</tbody>
</table>
2.6.1 Review of various methods of Migration .............................................. 25

CHAPTER 3 ................................................................................................. 28
RELATED WORK .......................................................................................... 28
3.1 Literature review .................................................................................. 28

CHAPTER 4 ................................................................................................. 31
RESEARCH METHOD .................................................................................. 31
4.1 SMART-MP Technique ......................................................................... 31
4.2 SMART activities .................................................................................. 32
  4.2.1 Establishing Migration context ......................................................... 33
  4.2.2 Migration Feasibility Decision Point .............................................. 34
  4.2.3 Defining Candidate Services .......................................................... 34
  4.2.4 Describe Existing Capability ............................................................ 34
  4.2.5 Describe Target SOA Environment ................................................. 36
  4.2.6 Analyze the Gap ............................................................................... 37
  4.2.7 Develop Migration strategy .............................................................. 37

CHAPTER 5 ................................................................................................. 38
FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS AND
  CONSTRAINTS ............................................................................................. 38
5.1 Requirements specification ................................................................. 38
5.2 Experimental Lab setup ....................................................................... 40
5.3 Non-functional requirements ............................................................... 41
5.4 Business vision and objectives ............................................................ 42

CHAPTER 6 ................................................................................................. 44
IMPLEMENTATION ....................................................................................... 44
6.1 Implementation Approaches ................................................................. 44
  6.1.1 Reuse approach ............................................................................... 44
  6.1.2 Migration Approach ....................................................................... 50

CHAPTER 7 ................................................................................................. 58
RESULTS AND EVALUATIONS ................................................................. 58
7.1 Reuse Approach .................................................................................... 58
7.2 Migration Approach ........................................................................................................ 61
7.3 Comparison Reuse vs. Migration Approach ................................................................. 64
CHAPTER 8 .......................................................................................................................... 67
CONCLUSION ..................................................................................................................... 67
8.1 Conclusion and Future Work ......................................................................................... 67
APPENDIX A ....................................................................................................................... 69
REFERENCES .................................................................................................................... 74
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>SOA Collaborations – Publish-Find-Bind-Execute Paradigm</td>
<td>11</td>
</tr>
<tr>
<td>2-2</td>
<td>Web Service Collaboration</td>
<td>12</td>
</tr>
<tr>
<td>2-3</td>
<td>Web Services Protocol Stack</td>
<td>15</td>
</tr>
<tr>
<td>2-4</td>
<td>CATIA V5 user interface</td>
<td>20</td>
</tr>
<tr>
<td>2-5</td>
<td>ENOVIA V5 LCA</td>
<td>21</td>
</tr>
<tr>
<td>2-6</td>
<td>ENOVIA V5 VPM Navigator</td>
<td>22</td>
</tr>
<tr>
<td>4-1</td>
<td>SMART Family of Techniques</td>
<td>32</td>
</tr>
<tr>
<td>4-2</td>
<td>SMART Process</td>
<td>33</td>
</tr>
<tr>
<td>4-3</td>
<td>AS-IS High-level ENOVIA Architecture</td>
<td>36</td>
</tr>
<tr>
<td>5-1</td>
<td>ENOVIA Part Detail Preview</td>
<td>39</td>
</tr>
<tr>
<td>6-1</td>
<td>Generated Project Structure for ENOVIA using Apache Axis</td>
<td>49</td>
</tr>
<tr>
<td>6-2</td>
<td>CAA Web Service WSDL Document</td>
<td>50</td>
</tr>
<tr>
<td>6-3</td>
<td>ENOVIA Use cases</td>
<td>53</td>
</tr>
<tr>
<td>7-1</td>
<td>Reuse approach expected result</td>
<td>59</td>
</tr>
<tr>
<td>7-2</td>
<td>Reuse approach actual result</td>
<td>60</td>
</tr>
<tr>
<td>7-3</td>
<td>REST API output response from POSTMAN for unique partnumber</td>
<td>62</td>
</tr>
<tr>
<td>7-4</td>
<td>REST API output response from POSTMAN for part number and unique version</td>
<td>63</td>
</tr>
<tr>
<td>7-5</td>
<td>REST API output response from POSTMAN for part number, unique version and drawing type</td>
<td>64</td>
</tr>
<tr>
<td>A-1</td>
<td>ENOVIA Service Interface</td>
<td>70</td>
</tr>
<tr>
<td>A-2</td>
<td>Logical Architecture</td>
<td>72</td>
</tr>
<tr>
<td>A-3</td>
<td>TO-BE Architecture</td>
<td>73</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 2-1: Non-functional requirements .............................................................. 17
Table 2-2: Non-functional requirements metrics .................................................. 18

Table 6-1: Migration Evaluation Criteria for ENOVIA Systems ............................ 51
Table 6-2: HTTP Methods for RESTful Services .................................................. 55
Table 6-3: ENOVIA REST Service API ............................................................... 55

Table 7-1: Comparison Reuse vs. Migration Approach ........................................ 64
Chapter 1

Introduction

This chapter introduces the general background needed for research of this thesis and describes the goals of the study.

1.1 Background

Enterprises rely on software systems that have been developed a long time ago, which are difficult to replace since they carry out the mission critical business processes (Marks & Bell, 2006). Currently, there is a need to analyze the legacy application capabilities in meeting the changing market demands. Additionally, IT has become an integral part of enterprise data management to support the business processes of an organization ranging from data storage to provisioning data when required by the business operations.

A key challenge faced by many large enterprises is the maintenance of their legacy applications and more specifically migrating those applications to modern and flexible platforms (Khadka, Saeidi, Jansen, & Hage, 2013). Modernization efforts of these legacy enterprise systems help in leveraging the core value of legacy systems, in addition, to quickly adapting to the frequency of change in business requirements and supporting agile business processes (Verma & Shah, 2011). Moreover, the need for seamless integration of such legacy systems with modern systems would require newer enterprise integration techniques. Enterprise application integration can be defined as seamless integration or interfacing two or more enterprise systems to function as in real time (Stevetuf, 2008). One plausible solution could be a migration of these systems to Service Oriented Architecture in which the legacy features can be re-used (Papazoglou, Traverso, Dustdar, & Leymann, 2007).

SOA also bridges the gap between the business processes and technology resulting in improved reuse of existing applications and interoperability of disparate applications (Choi, Nazareth, & Jain, 2013).

The gaining popularity of SOA has compelled organizations to handpick SOA as the architectural solution to provide a robust computing platform. Furthermore,
constraints such as the ever-changing business needs, shorter time to market also make SOA a strong candidate for organizations to migrate from legacy systems.

1.2 Problem motivation

The business processes within Scania are currently supported by a vast array of applications. Over the years Scania business has grown, thereby extending the operation internationally which puts heavy demands on the Scania IT services. New applications were to be integrated with the existing ones i.e. process of integrating multiple applications that are independently developed and implemented across heterogeneous platforms to operate as one. This has reflected in the supporting IT systems i.e. existing core systems have become very complex and the usage of these applications have increased considerably.

In addition, there is a difference in the way the business expresses its demands and the demands of the IT-people for turning this into IT support. These challenges initiate a request for a structured, agile application portfolio which not only makes an efficient use of the IT infrastructure but also flexible enough to keep up with continuous changes in organization’s business processes.

Scania is using Dassault Systèmes CATIA V5 platform for the mainstream product design with ENOVIA V5 vPLM as its Product Data Management vault. CATIA V5 application is used to manage the 3D geometries whereas ENOVIA V5 application helps in data management such as storing and retrieving the simulated data to and from the system.

Scania heterogeneous IT environment and enterprise landscape are the principle driver for picking Web services as the middleware innovation for the institutionalized service contract. Numerous studies claim that (Razavian & Lago, 5AD) (Kamoun, 2007) (Baghdadi & Al-Bulushi, 3AD), SOA introduces a concept in which services are significant elements of business processes. Since it has the capability of integrating different services in different business scenarios, the goal of this project was to implement a web service interface for the existing legacy system that would provide a closer relationship between business and IT.
1.3 Central goal of the thesis

In this research, the central aim is to develop a web service prototype with clearly defined integration best practices while considering Scania IT standards. This research aims to answer the research questions following the principles:

- What are the major challenges influencing the successful migration of legacy systems to SOA?
- What are the practical implications from a technical perspective of implementing SOA within the enterprise landscape of Scania?

1.4 Overview

In chapter two, a brief outline of the SOA, Legacy System (Scania ENOVIA system) and challenges of migration to SOA are explained. Readers who are already familiar with these SOA concepts could skip the entire section 2.1 in Chapter 2.

In chapter three, related work are discussed along with their advantages and challenges in legacy to SOA migration.

In chapter four, the research method and approaches used to structure the execution is discussed.

Chapter five defines the functional, non-functional requirements and constraints of this project.

Chapter six explains the motivation of chosen approach, highlights the business benefits and implementation details of this project.

Chapter seven discusses how the application is tested through different use-cases and compares the results of various implementation approaches.

Finally, chapter eight contains the conclusion and the summary of the project. It also presents possible future work on the platform.
Chapter 2

Theoretical Framework

This chapter provides an insight into the key concepts of SOA and its principles. Furthermore, potential benefits of SOA adoption, the realization of SOA through web services are discussed and elaborated. Finally, plausible challenges of migrating Enovia system to SOA modernization are explained.

2.1 Understanding SOA

Most enterprises today has a hybrid IT environment i.e. systems and applications with diverse technology, platforms and architectures. The key challenge is to integrate these systems from multiple vendors across different platforms in a seamless manner. Moreover, business processes should quickly respond to changing business requirements. Thus, an IT framework must empower the business to adopt respectively.

One of the fundamental guarantees of the service-oriented model is encouraging reuse of business resources in legacy frameworks. SOA can be defined as a method to expose discrete business functions as adaptable services to support business processes (Endrei et al., 2004).

From a business point of view according to IBM:
“A Service-oriented architecture provides the flexibility to treat the elements of business processes and their underlying IT infrastructure as secure, standardized, component (services) that can be reused and combined to address changing business priorities.” (Norbert Bieberstein, Sanjay Bose, Marc Fiammante, Keith Jones, 2013)

From the above definition, it is clear that the SOA activity needs to satisfy various targets to meet the business objectives and one of those destinations is to chronicle reuse of services. Fundamental building blocks of SOA are services (SEI | Carnegie Mellon University, n.d.).

A service is a coarse-grained, discoverable, and self-contained software entity that interacts with applications and other services through a loosely coupled, often
asynchronous and message-based communication model (Brown, Johnston, & Kelly, 2002).

As described earlier, SOA acquaints a methodology focused on providing services in a distributed manner (Endrei et al., 2004). These services will be consumed by either end clients applications or different services. Since services are the fundamental building blocks of SOA, it is important to present this concept.

2.1.1 Basic SOA concepts

The SOA key concepts are explained through the interaction of three main roles, which are service consumer, service provider and service registry (Endrei et al., 2004).

Service consumer

A service consumer is a software entity that calls a service provider to consume a service, in order to deliver a particular business process. Traditionally, service consumers can be an end-user application or another service.

Service provider

A service provider is a software entity that implements a service specification, which defines the interface and functionality offered by the service. It provides sufficient information in the service registry, which eases service consumers to find the appropriate service.

Service registry

Service registry acts as a bridge between the service consumer and the provider. It enables service providers to register the services with service description and helps the service consumers to look up the interfaces and services location defined by the providers.

SOA collaboration has three operations, which are publish, find and bind, invoke. Figure 2-1 (Michael Champion, Chris Ferris, Eric Newcomer, & David Orchard, 2002) explains the relationship between web services roles and its operations. Figure 2-2 (Endrei et al., 2004) illustrates the detailed web service collaboration.
Publish

The service provider describes and defines the services functionality and publishes in the service registry.

Find

The service consumer uses find operation to retrieve the service description from the registry.

Bind and Invoke

The service registry uses the service description to bind to the suitable service provider and invoke the requested service.

Figure 2-1: SOA Collaborations – Publish-Find-Bind-Execute Paradigm
2.1.2 Principles of SOA

With a particular end goal to construct services that are gainful for the enterprises, there are characteristics that every service should meet. These characteristics are the key elements of services, which are:

Coarse-grained

Coarse-grained refers to the distinctiveness of services to provide significant business process capability (Grace Lewis, Morris, & Smith, 2005). Each service tends to encapsulate one or several steps within of a business process, as against low-level business functions. Low-level business functions split functionality into fine-grained services, which are limited to small part of business logic. This might increase the number of service calls resulting in communication overhead (Dirk Krafzig, Karl Banke, 2004). Depending upon the level of usefulness every service provides, the granularity of the service fluctuates.
Loosely coupled

Coupling (TechTarget, 2011) “refers to the degree to which software components depend upon each other” (Dirk Krafzig, Karl Banke, 2004). Loose coupling is a design principle whereby interfaces of services ought to be designed as independent from the service implementation. Also, loose coupling allows the flexibility for the service provider to change the service implementation without affecting the service consumers as long as the service contract is unchanged.

Reusable and composability

SOA promotes reusability. Each service, encapsulating business logic should be properly divided such that it has the potential to be reused across the enterprise (Erl, 2007; Marks & Bell, 2006). The principle of composability refers to how services can be composed of other services to form a new composite service thereby promoting reuse (Erl, 2007).

2.1.3 Benefits of SOA

The main advantages of SOA that provide competitive edge to businesses are

Leverage existing assets

SOA gives a layer of abstraction to leverage existing assets by exposing them as services to be used by service consumers. This helps enterprises in getting value out of existing assets as opposed to needing to rebuild a system from scratch.

Easier to integrate and manage complexity

The service consumer interacts with the service provider through service specification and not the underlying implementation. Since the details of service implementation are hidden, integration turns out to be easier and manageable. This feature of SOA is called implementation transparency as it isolates the service implementation complexities.

Technology independence

SOA decouples technology service lifecycles from business process lifecycles using service contracts and service composition (Dirk Krafzig, Karl Banke, 2004). Technology independence aids in better integration and communication of different applications using
a common protocol and standardized interfaces in a heterogeneous IT landscapes. This heterogeneity nature of SOA enables different technologies to integrate and makes an enterprise less susceptible to technology changes.

**More responsive and faster time-to-market**

Leveraging existing assets reduces the entire software development lifecycle of requirement analysis, design, development and testing those results in fewer development efforts. This actuates fast development of new business services and enterprises can react rapidly to changes and diminish the time-to-market.

**Reduced cost and increased reuse**

Since SOA allows integrating existing systems, it preserves previous investments thereby offering a possibility to reuse existing business logic to compose new services. Reusing services also “significantly reduces redundancies and inconsistencies in business data” (Dirk Krafzig, Karl Banke, 2004).

**2.1.4 SOA realization**

The most common implementation of SOA is using a technology known as Web Services. According to W3C, the term web services can be defined as follows:

“A Web Service is a software application identified by a URI, whose interfaces and bindings are capable of been defined, described and discovered as XML artifacts. A Web Service supports direct interaction with other software agents using XML-based messages exchanged via internet-based protocols” (Farooq & Arshad, 2011) (Endrei et al., 2004).

Web Services are empowered by pervasive technologies XML and HTTP, wherein Extensible Markup Language (XML) is the most common format for sharing data between Web services and the Hypertext Transfer Protocol (HTTP) is a transport protocol, which is widely supported by browser and Web servers.

Web services = XML + transport protocol (such as HTTP) (Endrei et al., 2004)
Web services achieve interoperability through a set of XML-based open standards, such as WSDL, SOAP and UDDI. These standards provide a common approach to defining, identifying, locating, publishing and consuming web services. Figure 2-3 (CodeGear(TM), 2008) depicts the Web service protocols stack, which is detailed below.

- HTTP (HyperText Transfer Protocol) is the commonly used transport layer protocol in web services and is primarily responsible for delivering messages between network applications.
- Simple Object Access Protocol (SOAP) encodes messages in a common XML format across the wire and is the commonly used messaging protocol in web services (O’Reilly, n.d.).
- Web Service Description Language (WSDL) is the description protocol, which is XML based and is used in describing the functionality of a web service.
- Universal Description, Discovery and Integration (UDDI) specification is the discovery protocol that is used by service providers to list their exposed services and helps service consumers to locate effectively those services (Aydin, 2007)

2.1.5 SOAP vs. REST

The common standard format for exchanging Web Services data over HTTP is Simple Object Access Protocol (SOAP). WSDL is used in conjunction with SOAP to expose web services over the network.

Representational State Transfer (REST) a term coined by Roy Fielding (Fielding, 2000) is an architectural style for developing web services. In REST, client request server, the server processes the request and returns the appropriate response. REST focuses on accessing named resources through a consistent Uniform Resource Identifier (URI). REST uses standard HTTP as the transport protocol. In REST, actions such as creating, updating and deleting resources are performed using the HTTP verbs (e.g., GET, PUT, POST, and DELETE).

Scania is migrating its legacy applications at a fast pace to support REST architectural style since RESTful applications essentially fulfill the quality requirements of enterprise application integration such as performance and scalability. As discussed by Sudhanshu (Hate, 2009), REST seems to have more scalability, interoperability and performance compared to SOAP, whereas SOAP has more security and reliability. REST completely leverages protocols and standards that power the WWW (World Wide Web) and it is simpler than traditional SOAP-based web services. REST provides better support to the browser, easy integration of services and provides interfaces that simplify client application development.

However, SOAP is a traditional way of implementation among enterprises for application integration; REST is preferred when it demands to make changes in the complex application architectures. In addition, REST is predominant choice of implementing Application Programming Interfaces (APIs) to communicate over networks.
2.1.6 Non-functional attributes of SOA

The complex nature of today’s information systems is determined by not only the functional requirements but also the desired quality attributes of the system such as performance, availability, reliability, scalability and security which are Non-functional requirements (NFRs).

Non-functional requirements are requirements that specify criteria that can be used to judge the operation of a system, rather than specific behaviors. NFR can be contrasted with [functional requirements] that specify specific behavior or functions. In general, functional requirements define what a system is supposed to do whereas non-functional requirements define how a system is supposed to be.

In SOA, each service has a direct impact on system ability to achieve its requirements on functional side and primarily concerned with identifying Quality of Service (QoS) attributes on the non-functional side. O’Brien et al. (O’Brien, Merson, & Bass, 2007) described on how to determine QoS attributes that affect SOA with recommended solutions to satisfy quality characteristics. Furthermore, the adverse impact on the operational requirements of SOA due to lack of QoS attributes such as availability was discussed.

Service Level Agreement Monitor (SALMon) a monitoring, service level agreement, violation detection tool was suggested by Ameller and Franch as a monitoring and testing technique to obtain the runtime QoS (Ameller & Franch, 2008), wherein the quality characteristics primary emphasis is on the technical and non-technical characteristics of such services.

The Table 2-1 below represents the relationship between customer requirements, concerns and Non-functional requirement.

<table>
<thead>
<tr>
<th>Customer requirements</th>
<th>Concerns</th>
<th>Non-functional requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Ease of use</td>
<td>Usability</td>
</tr>
<tr>
<td></td>
<td>Unauthorized access</td>
<td>Security</td>
</tr>
<tr>
<td></td>
<td>Likelihood of failure</td>
<td>Reliability</td>
</tr>
<tr>
<td>Property</td>
<td>Metric</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>Throughput (number of operations performed per second)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response time to user input</td>
<td></td>
</tr>
<tr>
<td>Scalability</td>
<td>How the system responds to increased loads.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(for large number of users)</td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>Rate of occurrence of failure  (Systems which can provide regular</td>
<td></td>
</tr>
<tr>
<td></td>
<td>service where failure is significant)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean time to failure (a measure of the time between observer failures</td>
<td></td>
</tr>
<tr>
<td></td>
<td>of the system, which is more critical in Enovia systems where client</td>
<td></td>
</tr>
<tr>
<td></td>
<td>runs with long transactions). How easily client can resume connection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>with Enovia systems.</td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td>Probability of failure on demand</td>
<td></td>
</tr>
<tr>
<td>Usability</td>
<td>Errors made by the user in a given time frame.</td>
<td></td>
</tr>
<tr>
<td>Robustness</td>
<td>Time to start after system failure</td>
<td></td>
</tr>
</tbody>
</table>

The key measurable metrics for NFRs are given in Table 2-2 below

Table 2-2: Non-functional requirements metrics

2.2 Understanding ENOVIA V5 system components

This section introduces ENOVIA suite and its system components such as CATIA V5, ENOVIA V5 VPLM, ENOVIA V5 LCA, ENOVIA V5 VPM Navigator, but the
discussion of them is limited to those involved within the scope of this thesis work. Background knowledge of these concepts is provided to assist the reader in a better understanding of the later chapters.

Component Application Architecture (CAA) is the core architecture on which the application software like CATIA, ENOVIA and DELMIA are built. CAA Rapid Application Development Environment (RADE) V5 facilitates implementation of 3D PLM business processes to meet the industry standards. In addition, CAA RADE facilitates specific enterprise requirements, which necessitate the customizations of the CAD/CAM/CAE application in enhancing the product lifecycle capability.

CAA RADE provides the following benefits:

- Helps in fast development of robust applications
- To support service-oriented architecture (SOA) CAA RADE exposes language-independent Enovia web services APIs that assist the integration of Enovia systems with other third-party enterprise applications.

The architecture of CAA is beyond the scope of this thesis.

2.2.1 CATIA V5

CATIA V5 (Computer Aided Three-Dimensional Interactive Application) (Genuang, Takzim, Maokil, & Takzim, 2012) is a multiplatform CAD/CAM/CAE solutions developed by Dassault Systèmes. CATIA supports digital product definition and simulation i.e. a tool used to define, model, analyze and simulate the manufacturing of a product. CATIA aids in product engineering and manufacturing from initial concept to product in service, which is widely used in automobile and aerospace domains. To design a 3D model in CATIA, the user has to import Product Information into its Workspace. The bill of material, which provides details on all the required parts and product assembling, is obtained from a proprietary legacy system deployed in-house within Scania IT. A simple user interface of CATIA can be seen in Figure 2-4.
2.2.2 ENOVIA V5 VPLM

ENOVIA VPLM (Enterprise inNOvation VIA Virtual Product Lifecycle Management) (Slideshare, 2014) (Genuang et al., 2012) from Dassault Systèmes is an information management solution helping enterprises in managing the design and fabrication of parts including the industrial space modeling and configuration of digital models. ENOVIA acts as a virtual collaborative data management tool to design, simulate, validate products designs and processes throughout the product lifecycle.

2.2.3 ENOVIA V5 LCA

ENOVIA LCA (Enterprise inNOvation VIA LifeCycle Applications) is a set of information management solutions which is used to manage the Product Definition Data throughout the product lifecycle via an object oriented PLM (Product LifeCycle Management) database management application. It provides support for unified access from 3D product lifecycle information and the product, process and resource (PPR) information definition in ENOVIA V5 VPM and DELMIA V5 via a collaborative, web-based environment (Genuang et al., 2012).
ENOVOIA LCA solutions provide support for business processes from product specification through logistics planning. Enovia LCA is implemented within a comprehensive set of four ENOVIA foundations (LifeCycle Applications, Enterprise Architecture, RADE et PPR Hub) that covers the complete requirements of a scalable information system including extensive data management, workflow support and user-based life cycle applications. ENOVIA LCA is shown in Figure 2-5 (Systèmes, 2005).

![Figure 2-5: ENOVIA V5 LCA](image)

2.2.4 ENOVIA V5 VPM Navigator

ENOVIA V5 VPM Navigator (Enterprise inNOvation VIA Virtual Product Manager Navigator) is an object oriented PLM application similar to ENOVIA VPLM, which is fully integrated with CATIA V5. ENOVIA V5 VPM offers a component that supports ENOVIA plug and play in CATIA V5 application.
VPM Navigator along with VPM Relational Design and VPM Configured Product Design are the cornerstone applications for concurrent engineering in configured product design and review i.e. managing the creation and maturation of virtual product definition from the highest level of product hierarchy to the lowest feature usage. VPM Navigator allows one to access both product definition and product data stored in the ENOVIA V5 VPM with extensive capabilities for navigation, search and filtering using various methods such as queries or tree navigation and collaboration tools.

Furthermore, V5 VPM captures and manages engineering design intent by exposing specifications, manufacturing process and rules, operational parameters, simulation results, thereby accelerating the understanding of how change affects associated product components and processes. Enovia v5 VPM Navigator is shown in Figure 2-6 (Systèmes, 2007).

![Figure 2-6: ENOVIA V5 VPM Navigator](image-url)
2.2.5 ENOVIA VPM People and Organization

ENOVIA VPM People and Organization is a suite of software application, which provides an intuitive Web-based Administration of ENOVIA VPM. It enables functionality to search, create, modify and delete People and Organization objects such as Organization, Person, Project, Role and Context for system administrators\(^1\). Thus, it enables easy users and security setup management in an organization.

2.3 Legacy system challenges

This section summarizes the legacy system challenges faced by organizations:

- A legacy system is very expensive and directly proportional to the applications age as they become increasingly expensive to maintain. In addition, the internal knowledge becomes increasingly scarce as a platform ages.
- Most legacy applications are usually built around proprietary, single-vendor tools where certain part or all of the system may be implemented using an array of different programming language some of which might be even obsolete. Furthermore, the business processes and business rules are often tightly coupled with the software, which makes it difficult to extract the functionalities to be exposed as a service.
- Legacy systems may require appropriate infrastructure, which is incompatible with virtualized environments to provide the new functionality required by the enterprise.
- The entangled custom built nature of the legacy systems increases the overhead across build and deployment process, thereby making it difficult to get continuous feedbacks and maintenance time gets longer.

\(^1\) [http://www.3ds.com/products-services/enovia/products/enovia-v5/vplm/domain/ENOVIA_V5_VPM/product/POC/]
Despite critical business logic being held in these legacy systems, the integration efforts have always been a challenge due to lack of clean interfaces (Wu et al., 1997).

2.4 SOA challenges

The significant challenges faced by organizations while migration to SOA are as follows:

- Since the thesis focus is on migration towards SOA environment (web services), there might be the case that not all legacy system have support XML and SOAP libraries.
- Legacy systems are batch oriented and has been used to perform non real-time data intensive task, which needs to be scheduled at predetermined intervals whereas SOA addresses real-time interactive processes through its request-response pattern.
- Enterprises using commercial products might run into licensing issues when such functionality is exposed to a greater number of consumers through SOA modernization.

2.5 Challenges in migration of ENOVIA Systems to SOA

ENOVIA software architecture has attempted for long to deal with increasing levels of software complexity. The maintenance of such complex legacy systems becomes very inefficient over time and other problems such as lack of up-to-date documentation, skilled manpower and high maintenance costs occur. Despite such problems, Enovia systems cannot be replaced as they are mission critical applications and implement the core business logic. Moreover, any failure of this legacy system can have a significant impact on business. A feasible solution is a migration of the legacy system into new technological environments such that the legacy features can be re-used where plausible.

While moving to SOA, it is important to identify what can be migrated from the original legacy system and choosing the right strategy to migrate the existing application to SOA. Industry experts in recent times have proposed SOA as a promising architectural style enabling legacy applications reusability by exposing their functionalities. However,
a legacy to SOA migration is very challenging and the migration process requires proper migration planning and migration execution.

2.6 Migration of Legacy Systems to SOA

While moving to SOA, it is important to identify what can be migrated from the existing ENOVIA system and choosing the right strategy to migrate the existing application to SOA. Almonaies et al. (Almonaies, Cordy, & Dean, 2010) divided the approach into four types

i. Replacement (rewriting the existing systems’ application or replacing by buying a new system/solution)

ii. Reengineering (reverse engineering the current system and add SOA functionality incrementally)

iii. Wrapping (wrapping the system and providing interfaces to access the functionality of legacy system)

iv. Data Migration (preserving the legacy system’s data and moving the legacy system to SOA).

Only the first three options, which are relevant for the thesis is explored further.

2.6.1 Review of various methods of Migration

The methods to deal with legacy to SOA migration are described below.

Rewriting

Rewriting the application from scratch or replacing the complete system is very expensive, risky and time-consuming. Despite time-consuming and risky this approach has the advantage that it delivers a customized solution if architected correctly and in best case can be built exactly to meet the organization’s need (Almonaies et al., 2010). Another distinct advantage of rewriting the application is that it can provide the optimal use of technologies and processes that are available for the language being used in the future. However regardless of how skilled the developers are in a software development project, there are always bugs that are produced during the development.
Commella-Dorda et al. (Comella-Dorda, Wallnau, Seacord, & Robert, 2000) identified two significant risks involved in the replacement approach. For the entire organizational user base, the maintenance of the newly developed system will not be as familiar as the old system. Moreover, most importantly the lack of a guarantee that the new system will be as functional as the original legacy system. Umar et al. (Umar & Zordan, 2009) in their research have also discussed the risk involved with new systems as it might not work as it is expected and can be time consuming and expensive. Thus, in replacement approach, not only the cost of development occurs, but it also includes the training cost incurred by the newly developed system that results in the huge learning curve resulting loss of productivity.

Other viable solution in replacement of application would be buying COTS (Commercial-off-the-shelf) components, which could be much less risky. However, COTS might become more expensive over the long run since future modifications may be difficult and expensive to perform. Another major risk involved mentioned by Almonaies et al. (Almonaies et al., 2010) is that the organization will lose control of the software code base. Generally, in most cases the replacement strategy for legacy systems is considered the least desirable solution for migration to SOA.

Reengineering

Most legacy systems are tightly coupled and are not architected with “software as a service” methodologies in mind. Almonaies et al. (Almonaies et al., 2010) have discussed three main issues in service-oriented reengineering:

- service identification
- service packaging and
- service deployment (Almonaies et al., 2010).

The services and functionalities that are reusable and reliable embedded with valuable business logic in the legacy system should be identified, as these functions are useful to be exposed as independent services. Re-engineering the legacy system, which has a well-defined structure, makes it easier for reverse engineer incrementally. However, in a tightly coupled legacy system and in systems, which heavily depend on the environment,
it is very hard to achieve re-engineering and a big bang re-engineering is often difficult if not impossible.

**Wrapping**

Wrapping exposes a new SOA interface (e.g. WSDL) to a given legacy component, making it readily available for use by various other software components. Wrapping usually comes handy when the legacy code is very expensive to rewrite. However, this solution is not good when a fast solution is needed. One critical problem with this strategy is that it is hard to solve the existing issues, if there are any (Almonaies et al., 2010).

However, this strategy rarely introduce the new problems or challenges, and it is the least risky approach, and in some cases if designed as per the specifications can be the most appropriate.
Chapter 3

Related work

This chapter aims at reviewing the various migration process in the fuzzy research field of legacy to SOA modernization. Several SOA migration approaches its activities, artifacts, benefits and weakness of existing methods are stated and described.

3.1 Literature review

Various approaches of migrating a Legacy system to the Service-Oriented Architecture is extensively researched over the last decade (Papazoglou, 2003) (McGovern, Sims, Jain, & Little, 2006) (Rosen & Books24x7, 2008). Predominantly, to reuse the valuable business logic that resides within legacy applications. SOA has gained significant consideration both in academic and industry as a promising architectural style, thereby enabling legacy applications to be leveraged by exposing and reusing their functionalities (Khadka et al., 2013). A legacy to SOA Migration aims at reducing maintenance costs, in addition to system adaptability to changing business needs. Such modular components ensure reusability, extensibility and interoperability of multiple entities across the enterprise (Papazoglou et al., 2007).

Lewis et al. (G Lewis, Morris, & Smith, 2006) proposed the reusability of the legacy system, by exposing legacy system functionalities as services. However, this requires an initial analysis of interactions required by SOA and corresponding changes in the legacy system components be considered. Due to the technical nature of the legacy systems, reusability might have substantial complexity. Nevertheless, the author proposes service-oriented migration and reuse technique (SMART) which can determine the feasibility of exposing legacy components as services in service-oriented architecture.

Furthermore, SMART technique helps enterprises to decide on the SOA adoption path. SMART family of techniques involves a process, an interview questionnaire; artifacts required for the use in the process and can be targeted to different enterprise needs. Various SMART techniques are described below:
- SMART-MP (Migration Pilot) - helps enterprise to choose and implement a pilot project based on the migration strategy
- SMART-AF (Adoption Feasibility) - helps to validate if the enterprise can migrate toward an SOA environment
- SMART-ESP (Enterprise SystemPortfolio) - channelizes in the selection and creation of services from the legacy systems
- SMART-ENV (Environment) - analyzes the risk and cost involved in migrating the legacy to target SOA environment
- SMART-SYS (System) - helps enterprise to develop and maintain a complete service oriented system (SEI | Carnegie Mellon University, 2015)

Zhang and Yang (Z. Zhang & Yang, 2004) proposed re-engineering approach, which is used to restructure legacy system code and facilitate legacy system code extraction for web service construction in comparison to traditional wrapping methodology. Zhang et al. (B. Zhang, Bao, Zhou, Hu, & Chen, 2008) proposed a black box strategy in which the legacy system functionalities are exposed as web services using a wrapping methodology that are appropriate for GUI-based legacy systems.

Smith (Smith, 2007) has proposed a strategic approach for migrating legacy systems to SOA, when and how to adopt SOA by considering the business feasibility and the technical vision of the organization’s legacy system portfolio.

Street and Gomaa (Street & Gomaa, 2008) have stated that the enterprises’ planning for either migrating to SOA or adopting SOA should have a deep understanding of its legacy system and find the possible areas to be migrated to SOA.

Semith et al. (Cetin et al., 2007) presented a method called MASHUP (MigrAtion to Service Harmonization Computing Platform) wherein AS IS architecture of the existing legacy system is analyzed along with infrastructure and quality attributes of service. Based on existing analysis the next phase would identify set of candidate services in conjunction with the design of TO BE architecture and the definition of Service Level Agreement (SLA) satisfying quality attributes of services.

Though SOA has the potential to offer significant benefits to an enterprise, this new architecture comes with its set of challenges. Despite benefits, adoption of SOA does
not necessarily lead to an advantage in an enterprise but can even result in drifting away from the primary business goals or unnecessarily add layers of complexity to the current IT system without offering any technical or business benefits.

Furthermore, migration planning and execution that are critical elements of the migration process aids in channelizing legacy system modernization to SOA (Balasubramaniam, Lewis, Morris, Simanta, & Smith, 2008). Migration planning is the first step, aims at answering two questions:

1. How to plan the migration
2. Is migration feasible in the given context?

Moreover, the second step is the migration execution where a supporting technology stack is developed to expose legacy applications as a service. Also, the feasibility of such migration process is determined by business drivers and legacy system characteristics (Umar & Zordan, 2009).
Chapter 4

Research Method

This thesis project has been created with the SMART-MP technique, which has helped to structure the work throughout the way. It discusses the strategic approach followed to design a solution to the defined problem.

4.1 SMART-MP Technique

SMART-MP technique has been chosen for this thesis project as it was found that it suits well for this project as per SMART family of techniques proposed by Software Engineering Institute, Carnegie Mellon University as shown in Figure 4-1 (Grace Lewis, Morris, Smith et al., 2005). SMART is an approach that helps enterprises to make preliminary decisions about migration planning, the process that determines the feasibility of the legacy systems (Grace Lewis, Morris, Smith, et al., 2005). SMART approach uses contextual analysis method to gather information about legacy components, target SOA and potential candidate services.

The outcome of SMART provides answers for

- Does migration of legacy system to SOA harmonize with enterprise vision?
- What services have to be created to meet the business needs?
- Identifying legacy components to be migrated as services?
- What changes to these components are required to accomplish the migration?
- What migration strategies are most appropriate?
4.2 SMART activities

The SMART–MP process has six activities, which are iterative. The activities and information flow of SMART are depicted in Figure 4-2 (Grace Lewis, Morris, Smith et al., 2005).

Each activity in SMART framework is used to describe the research process.
4.2.1 Establishing Migration context

This activity in SMART-MP facilitates high-level understanding of legacy systems, candidate services and stakeholder goals within the migration context from a both business and technical perspective.

Scania’s IWJP department has been tasked with developing Web services for the Enovia system that can be used by the different application and teams within the Scania enterprise. The current Enovia system targeted for migration is a collaborative PLM
system, which is reliable and robust to manage the sensitive and mission-critical data. Enovia empowers optimization across product management, design, manufacturing and procurement processes.

A standard web services environment based on the JMG reference architecture has been selected as the target SOA environment. This activity in the SMART process demonstrates the feasibility of Enovia component as a service to be used by OAS system.

The set of candidate services to be developed are described in Chapter 5 (section 5.1). The choice of service is based on their generic functionality so that it can be used by different applications within SCANIA IT.

4.2.2 Migration Feasibility Decision Point

Based on the data obtained from earlier activity, it is clear that identified legacy system is a good candidate for migration to services. This was based on the following factors:

- availability of stakeholders from the service provider (Enovia) and a service consumer (Scania IT and Business team)
- Good understanding of Enovia systems.
- Identify primary candidate services with clear inputs and outputs from existing business process
- Identification of components to be exposed as services and the request-response format

4.2.3 Defining Candidate Services

The goal of this activity is to specify the inputs and outputs in detail for each of the services identified as part of Establish Migration Context along with the quality of service requirements, which is described in the forthcoming chapters.

4.2.4 Describe Existing Capability

This activity gathers information about the legacy system components that contain the functionality to meet the needs of the selected services. Enovia components are primarily written in C++ in a Visual Studio 2008 development environment for low-level command API, whereas high-level business logic API’s is written in Java. It runs on both Windows
and AIX platform. The product documentation from Dassault Systèmes provides significant details about the legacy system, but available architecture documentation was incomplete. In general, the architecture documentation contains information about architectural views such as logical, development, process and physical view of the system. These views primarily describe the system from the viewpoint of different stakeholders such as end-users, developers and project managers. However, the major drawback with Enovia and its associated systems were a lack of logical and development views of the system architecture. This drawback presented a problem during our analysis in determining the features exposed by those legacy systems.

As shown in Figure 4-3 in the current environment rely heavily on the background batch processing system, to support continued business process. Traditionally, to provide the requested information Scania Enovia systems used batch jobs as the cornerstone of application processing which helped in effective utilization of existing computing resources and manages workload with minimal intervention. The key problem here was batch processes need to be scheduled at predetermined intervals, which require extensive logging and restart capabilities. In addition, the complexity is increased due to the invocation of different routines to provide true end-to-end business processing. This increases the total turnaround time, the time taken between the submission and return of complete output.
4.2.5 Describe Target SOA Environment

The target SOA environment for the migration of this research work is standard Web Services. The “to-be” system briefly represents two aspects of the target architecture:

- The target architecture focuses on functional as well as non-functional characteristics such as performance, security and availability QoS requirements.
- The selection of technology to be used for web service implementation (SOAP or REST) is determined at this stage. Additionally, the need to support both SOAP and REST technologies is also committed to facilitating backward compatibility with existing application integration. Decisions are made regarding the selection communication protocols, execution platforms and middleware and deployment requirements of the target SOA environment, which helps in planning a good migration strategy from the start.
4.2.6 Analyze the Gap

The activity in the SMART process describes existing capability eliciting legacy system’s architecture and highlighting the changes required in the legacy components to support the candidate services identified in step 1. In addition, it analyzes the potential tradeoffs between the current and future architecture. If there are gaps in the data analysis about the available legacy components, additional data about SOA requirements to build required services is obtained via hands-on architectural and code analysis.

4.2.7 Develop Migration strategy

Given the set of identified migration issues, the following migration strategy is followed:

- Describing the state of target architecture and identified services helps in concretely finalizing the service inputs and outputs and developing appropriate service interfaces.

- Work closely with the target architecture group that is defining the service reference architecture. Service reference oriented architecture provides a common platform for developing services, reusability of common service operations, and isolation of service code changes from service contracts.

- This step also helped in recalculating the effort required for such migration tasks based on the new understanding of business requirements and SOA constraints.
Chapter 5

Functional and Non-functional requirements and constraints

This chapter discusses the prototype requirements, goals and constraints. In addition, this section summaries the functional and non-functional requirements of the thesis to gain a better understanding of the implementation prerequisites.

5.1 Requirements specification

The thesis requirements were formulated based on interpretation of various information received from different groups with SCANIA IT. The given specifications were analyzed and conclusions were drawn to create web service interfaces. The hypothesis here is to deliver requested information using these web service interfaces. Each requirement specified was modeled, implemented and tested using an open source java library (REST assured) in the final prototype application.

The goals of the thesis address three major requirements as mentioned below.

Requirement - BR01: (Access to document preview stored in ENOVIA)

Goal: To retrieve JPEG file location along with part details stored in the vault. (HTTP GET)

Expected Output: [Figure 5-1] shows ENOVIA part detail preview
Requirement - BR02: (Access information including GEO Publish status stored in Enovia)

Goal: XML result set with the denomination, GEO publish status (published status of the spatial (geometric) position of part details in the CAD environment) and release status. (HTTP GET)

Expected output pattern: The XML result set along with its output attributes are given below:

```xml
<row>
    <partnumber>7777777</partnumber>
    <denomination>CABLE</denomination>
    <published>TRUE</published>
    <status>RELEASED</status>
</row>

<row>
    <partnumber>5555555</partnumber>
    <denomination>CABLE</denomination>
    <published>FALSE</published>
    <status>INWORK</status>
</row>
```


Requirement - BR03: (Access to Document and PartAxis values stored in ENOVIA)

Goal: To retrieve XML result set containing the mandatory twelve Part Axis values (Information carriers of positioning in x, y, z of products / articles CAD environment) stored in ENOVIA using HTTP GET. The Part Axis describes the geo-positioning of the articles (e.g.: Chassis)

Expected output pattern: The excepted output PARTAXIS1 - Number till PARTAXIS12 – Number is given below:

```
<row>
  <partaxis1>0</partaxis1>
  <partaxis2>1</partaxis2>
  <partaxis3>1</partaxis3>
  <partaxis4>1</partaxis4>
  <partaxis5>0</partaxis5>
  <partaxis6>1</partaxis6>
  <partaxis7>1</partaxis7>
  <partaxis8>0</partaxis8>
  <partaxis9>0</partaxis9>
  <partaxis10>1</partaxis10>
  <partaxis11>1</partaxis11>
  <partaxis12>1</partaxis12>
</row>
```

5.2 Experimental Lab setup

An experimental test bed consisting of two computers with similar hardware (an Intel Pentium i7- 2720QM (8 MB cache, 2.40 GHz clock) CPU, 16 GB of RAM, an Intel Ultimate-N 6300 (802.11 a/b/g/n) network interface, and a Solid State 500GB hard disk drive) were connected with a crossover cable. Each physical computer is running a copy of Microsoft’s Windows 7 Enterprise operating system. We used IPv4 as the communication protocol stack for the physical machines.

Java version 1.7.0_3 was installed as a prerequisite in each physical computer and Maven was the build automation tool used to manage dependencies. Perforce was used for efficient and effective source code management. Spring boot, a lightweight
standalone application server, in combination with Spring Data JPA\(^2\) an API used to implement JPA based repositories was used to create RESTful web services. This technology was used initially for developing a new system, which was aimed at potentially replacing the existing legacy system. However, this technology was later used to design the prototype application, which focused on migrating existing legacy system by exposing RESTful API’s. WebSphere technology was applied for reusing the functionalities of the existing legacy systems where it was feasible.

5.3 Non-functional requirements

a. Application and Hardware Standards

Compatible - The system must be compatible with the existing SCANIA IT infrastructure

Compliance - The system must be compliant with the hardware and software standards and guidelines of SCANIA IT

b. Performance

The system shall conform the performance requirements as per SCANIA IT standards and scalable to meet the evolving user base.

c. Access and Security

The REST web service should be available to all existing and new customers. People and Organization system will form the basis for user registry. Application access rights should be defined by the People and Organization administrative module.

d. Authentication

All users connecting to ENOVIA systems will be authenticated against People and Organization user registry. The credential information is expected to be propagated from People and Organization in the case of both customers and for internal users.

\(^2\) Java Persistence API (Application Programming Interface) is a specification for relational data management in applications using Java platform.
e. Authorization

People and Organization system will have a designated super user, capable of granting access rights and privileges to client users and internal users. Any user accessing a functionality in ENOVIA system should have the necessary authorization to perform such an operation. Authorization decision will be role based. Authenticated user will be verified against application roles before allowing an operation.

f. Encryption

The data entering or leaving ENOVIA system is classified as “Sensitive” and must be encrypted to adhere to the agreed upon security policy between People and Organization and ENOVIA Systems.

g. Host Security

The middleware architecture encompasses firewall in front of any interaction with it. All synchronous communication will be handled through https (Web Services) and asynchronous communication will be handled through secured FTP.

h. Audit Control, Archive and Data Retention

One of the key requirements of the system is to provide manual validation and exception handling mechanisms to control reports and mechanisms as proofs for the processes indicating who did, what and when. This is achieved an audit log for all the transactions.

- Audit Control – The system must leave a digital trail of all the transactions indicating who did, what and when. A complete audit trail of the system must be available for the system administrator to create control reports.
- Archive – All information in the system is classified “Sensitive” and must be archived as per the archival policy of Scania IT.

5.4 Business vision and objectives

The long-term vision of the enterprise is to establish a web service infrastructure that can support the following:

- Improved information flows i.e. ability to expose internal functionality across different systems
• Ability to integrate existing assets (re-use) and ability to develop new function rapidly
• Improved reliability i.e. ability to scale operations to meet different demand levels.
Chapter 6

Implementation

This chapter discusses the various implementation approaches and its challenges.

6.1 Implementation Approaches

From the various migration methods as described in Chapter 2, the following approaches were used in this research.

1. Reuse approach
2. Migration approach

6.1.1 Reuse approach

Firstly, since the focus of the thesis is SOA adoption, system functional goals must be clearly defined through a high-level understanding of legacy systems where the emphasis is on the effective reuse of business services at the macro level (a business unit of work). In addition, stakeholder goals analyzed within the migration context from a both business and technical perspective is aligned towards the project-specific adoption for flexible reconstruction of web services.

The key idea is to establish the procedures around identifying service candidates, as well as discovering reuse from the existing service catalog which determines if an existing service can be viable for reuse in the proposed manner. Once the candidate services are identified, the service definition defines the service boundaries, service and data contracts whereas service design define procedures required to design service interface. Finally, service implementation helps in determining the technical stack for the target SOA environment, which enables the reuse of services or components, operation and maintenance. Moreover, service operation, administration and maintenance are required to support SOA’s operational environment thereby enabling reuse and evolution despite measuring SOA's adoption and success.
Motivation

The goal of this approach is to consume Component Application Architecture (CAA) Web Services deployed with ENOVIA LCA Navigator application with IBM WebSphere application server as the container. The purpose is to analyze the language independent web services provided by the CAA architecture, identifying extension points to provide further development or customization from these components. The client application consuming the web services can be a Java, C++ or .NET application.

CAA RADE Web Services provides a standard integration to Enovia LCA Navigator application, which is primarily used in the visualization of the business process assisting real-time business decisions. Furthermore, it facilitates in business process evaluation to identify the gaps and propose improved business design process.

In our implementation approach, Java was preferred because of the enterprise IT landscape of Scania. According to CAA Web Services API documentation, only clients using the Apache Axis technology are officially supported for consuming CAA Web services.

Apache Axis (Apache eXtensible Interaction System) is an XML-based Web service framework, which supports both C++ and Java implementation of the SOAP server and APIs for deploying web service applications. (Wiki, n.d.)

In order to consume CAA web services, the client application requires a proxy generated using WSDL2Java tool. This tool is used to produce a Java client binding which is based on the WSDL definition of a CAA web service. (WSDL chosen is based on the business requirement mentioned earlier). Prior to proxy generation using WSDL, there are certain Apache Axis prerequisites required to configure the execution environment.

Prerequisites

In order for the WSDL2Java emitter and Axis SOAP runtime to run smoothly, it requires a list of Axis binary release version 1.3 files, which includes axis.jar, axis-schema.jar,

3 http://www.maruf.ca/files/caadoc/CAAWSTechArticles/CAAWSOverview.htm
jaxrpc.jar, saaj.jar, commons-discovery-0.2.jar, commons-logging-1.0.4.jar and wsdl4j-1.5.1.jar as listed below:

```
<Axis-installation-folder>\axis-1_3\lib\axis.jar
<Axis-installation-folder>\axis-1_3\lib\axis-schema.jar
<Axis-installation-folder>\axis-1_3\lib\jaxrpc.jar
<Axis-installation-folder>\axis-1_3\lib\saaj.jar
<Axis-installation-folder>\axis-1_3\lib\commons-discovery-0.2.jar
<Axis-installation-folder>\axis-1_3\lib\commons-logging-1.0.4.jar
<Axis-installation-folder>\axis-1_3\lib\wsdl4j-1.5.1.jar
```

Apart from binary dependencies, WSDL2Java emitter requires the CLASSPATH 4 environment variables to be set on Microsoft Windows, Solaris and Linux as shown below:

```
SET AXIS_HOME = <axis-installation-folder>
SET JDK_HOME = C:\Program Files\Java\jdk1.7.0_51
SET CLASSPATH = %CLASSPATH%;%AXIS_HOME%\axis.jar
SET CLASSPATH = %CLASSPATH%;%AXIS_HOME%\axis-schema.jar
SET CLASSPATH = %CLASSPATH%;%AXIS_HOME%\jaxrpc.jar
SET CLASSPATH = %CLASSPATH%;%AXIS_HOME%\saaj.jar
SET CLASSPATH = %CLASSPATH%;%AXIS_HOME%\commons-discovery-0.2.jar
SET CLASSPATH = %CLASSPATH%;%AXIS_HOME%\commons-logging-1.0.4.jar
SET CLASSPATH = %CLASSPATH%;%AXIS_HOME%\wsdl4j-1.5.1.jar
%JDK_HOME%\bin\java org.apache.axis.wsdl.WSDL2Java -h
```

From the command prompt configured in the above section, you can invoke the WSDL2Java emitter using the following syntax to generate the client proxy:

```
%JDK_HOME%\bin\java org.apache.axis.wsdl.WSDL2Java -o <Output-folder> -p <Package-name> -U <UserName> -P <Password> <WSDL-URI>
```

<Username>, <Password>: valid set of credentials for web service basic authentication.

As mentioned earlier, CAA Web Services relies on Apache AXIS and generates Java artifacts required for web services clients consuming the WSDL file. The following Java artefacts are created using the Apace Axis tool:

---

4 CLASSPATH is a parameter, set either on the command-line or through an environment variable, which helps Java compiler or JVM in finding the user-defined classes and packages.
- Service endpoint interface that the proxy client uses to invoke the remote SOAP service
- XML binding generated type values, which are Java classes, mapped from XML schema types and they represent the required input parameters.
- The web service class

Feasibility checks and Validation

CAA Web services require that each web service request provides a valid login via a standard HTTP Basic Authorization header. The default CAA authentication Web service responds with a status code 401 and a message containing “ResponseCode: 401 (Unauthorized)” if the Web Service request does not contain valid authorization information. If the request is valid, then the server sends a response message containing the list of operations supported by the Web Services. For our approach windows login is used for the basic authentication.

Prior to calling the actual business web service, the CAA login web service mentioned below needs to be invoked. This service provides the real entry point to the Enovia systems and creates the session context.

```
/resources/wsd1/ENOPosWS/ENOPos_ws/IApplicationBinder.wsdl(CAENOPosAppBinder)
```

To access a given WSDL interface document here is the URL syntax:

```
```

- `<Server>`: server name, including domain,
- `<Port>`: server port,
- `<Context-root>`: context root of the Web application,
- `/resources/wsd1/<Framework>/<Module>/<Interface>.wsdl`: relative path to the WSDL interface document

Example: `/resources/wsd1/ENOPosWS/ENOPos_ws/IApplicationBinder.wsdl`

The functionality exposed by CAA web services are

---

5 http://www.maruf.ca/files/caadoc/CAAPLMSecUseCases/CAENOPosAppBinderDotNetClient.htm
- getUserContexts – this retrieves the contexts associated with a particular user declared in the People and Organization database
- bindToApplication - log on to ENOVIA LCA V5 using a specific context
- releaseFromApplication - log off from ENOVIA LCA V5 in order to terminate an opened session

In order to describe the data types that represent the request/response SOAP messages, the CAA services rely on XSD, which is deployed in the following location along with default ENOVIA installation. The strategic steps in reusing the existing CAA web service is stated in the reference\(^6\) and it details more on “**Building and Launching a CAA Web Service Use Case**”.

\[/resources/xsd/*.xsd\]

To maintain HTTP session state with Apache Axis, ENOVIA LCA V5 uses custom class (CustomSessionHandler). After the build is complete, the project structure should look similar to \[\text{Figure 6-1}\]. In addition, the WSDL document structure is shown in \[\text{Figure 6-2}\].

\(^6\) http://www.maruf.ca/files/caadoc/CAAWSUseCases/CAAWSBuildAndLaunchUsingAxis.htm
Figure 6-1: Generated Project Structure for ENOVIA using Apache Axis
To run the above-specified use case, the following arguments are passed to the web service URL as given below:

```
-w http://<Scania web server Host/IP>:<Port>/<applicationname> -e <envoia username> -u <web server username> -p <web server password>
```

6.1.2 Migration Approach

Motivation

This goal of this approach is to leverage existing legacy system functionalities by providing a service interface, which can expose legacy functionality thereby enabling enterprises to respond rapidly to changing business requirements and to support business
processes. The key advantage is to allow integration of existing applications within an enterprise by providing functionality as services, which can be deployed within the existing IT infrastructure.

Migration activities:

Legacy System Understanding

The first step consists of analyzing Enovia and its associated systems. The purpose of this step is to understand the system architecture and identifying main functionalities that align business vision and objectives. It also facilitates in creating an inventory of existing features associated with the legacy application.

The following techniques were used to collect information and understand about the legacy system:

1. Interview developers and knowledge transition sessions with IWJP team who work with ENOVIA customizations
2. Manual inspection of existing documentation
3. Furthermore, reverse engineering techniques are used to analyze the system artifacts including source code and execution traces that can be used to recover and represent the architecture of the target application. Table 6-1 outlines the migration evaluation criteria for ENOVIA systems.

<table>
<thead>
<tr>
<th>#</th>
<th>Criteria</th>
<th>Applicability</th>
<th>Description</th>
<th>Importance</th>
<th>Ease of Data Availability</th>
<th>Parameter Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dependencies</td>
<td>Applications</td>
<td>Application dependencies implicitly determine on number of interfaces and call points</td>
<td>Y</td>
<td>Y</td>
<td>YY</td>
</tr>
</tbody>
</table>

Table 6-1: Migration Evaluation Criteria for ENOVIA Systems

<table>
<thead>
<tr>
<th></th>
<th>Business Criticality</th>
<th>Applications and interfaces</th>
<th>The assigned business criticality as per the organization</th>
<th>Y</th>
<th>Y</th>
<th>YY</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Test Data Availability</td>
<td>Applications and interfaces</td>
<td>Availability of the test data in the target development and test environment for the interfaces being migrated</td>
<td>Y</td>
<td>Y</td>
<td>YY</td>
</tr>
<tr>
<td>4</td>
<td>SLA Sensitivity of Interfaces</td>
<td>Interfaces</td>
<td>SLA sensitivity, coupled with the current distance of the actual performance from expected SLA limits</td>
<td>N</td>
<td>N</td>
<td>NN</td>
</tr>
<tr>
<td>5</td>
<td>Interface Complexity</td>
<td>Interfaces</td>
<td>Complexity of interface contract and data types used in the contract</td>
<td>Y</td>
<td>N</td>
<td>YN</td>
</tr>
<tr>
<td>6</td>
<td>Implementation Coupling</td>
<td>All (call points, interfaces, applications)</td>
<td>How much coupling exists between the integration endpoints and functional logic of the applications/service implementation or invocation unit</td>
<td>N</td>
<td>N</td>
<td>NN</td>
</tr>
<tr>
<td>7</td>
<td>Reengineering Effort</td>
<td>Applications</td>
<td>Existence of reengineering efforts within the provider or consumer</td>
<td>N</td>
<td>N</td>
<td>NN</td>
</tr>
<tr>
<td></td>
<td>Migration State of Dependencies</td>
<td>Applications</td>
<td>Migration state of the participating applications—especially useful in plan adaptations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---------------------------------</td>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Documentation Quality</td>
<td>Applications</td>
<td>Quality of application documentation, as it will impact the efficiency of the integration</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Selecting candidate services**

The knowledge gained from legacy system understanding helps in describing the functionalities to define an initial set of candidate services. UML Use Case diagram is chosen to represent the application functionalities and their relationships as shown in **Figure 6-3**.

![use cases](image)

**Figure 6-3: ENOVIA Use cases**

The candidate services can be classified as three sets of services.

---

53
1. Task services represent the functionalities of the application. Each main functionality of target application is a possible candidate task service.

2. Entity services represent the classes implementing the CRUD (CREATE, READ, UPDATE and DELETE) operations for persistent storage.

3. Functionalities such as exception handling, event logging and tracing, notification, data transformations are handled as utility services.

Service interface design

The Enovia Web Service interface follows REST-style architecture commonly referred to as RESTful web services. Scania has chosen to expose data as REST resources instead of a set of SOAP methods. The reason being, REST promotes the use of HTTP and its fundamental operations for building large-scale distributed systems. Furthermore, in REST approach, the object model can change at run-time and this type of dynamic behavior is hard to express for a SOAP service where the WSDL sets the data format. In addition, one important REST principle is that the consumer should only care about data it knows about and simply ignore added data that it does not understand. By adhering to this simple rule, changes can be made to the system and new data can be added to existing resources without affecting the consumer. If the consumer, on the other hand, validates data strictly according to a schema, it may be difficult to add new functionality to the system. Since the service provides only read operations via GET, this relaxed rule of how to process data is easiest to implement for consumers that only read data.

A service interface contains more than one operation and if they are semantically related, they are defined as part of the same interface. In order to support REST architecture style, constrained interfaces are used which contain a fixed set of standardized methods. These methods are then applied to resources located with Unified Resource Locators (URLs). A resource is an object with a type, associated data, relationships to other resources. The operations defined in the REST service interface depict the basic retrieval operations along with input parameters/output response needed for requesting information from ENOVIA systems.
**REST request format**

The REST request format is the simplest request format to use and allowed methods are shown in Table 6-2.

<table>
<thead>
<tr>
<th>HTTP method</th>
<th>Action</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Retrieve or read a resource</td>
<td>View a document</td>
</tr>
<tr>
<td>POST</td>
<td>Create a resource or perform an action</td>
<td>Add a document</td>
</tr>
<tr>
<td>PUT</td>
<td>Replace a resource</td>
<td>Update a document</td>
</tr>
<tr>
<td>DELETE</td>
<td>Remove a resource</td>
<td>Delete a document</td>
</tr>
</tbody>
</table>

*Content-Type is application/json.*

**REST response format**

In REST, the response to a REST request method defaults to JSON format. To return an API response in XML format one must update the accept header as:

**ACCEPT: application/xml**

The selected candidate services of the target application were defined as fine-grained, where a separate candidate service was designed for specific functionality as shown in the UML use case diagram.

The Table 6-3 below summarizes the semantic of HTTP operations when executed over the ENOVIA data model for the part master, part version and data model for document revision resources.

<table>
<thead>
<tr>
<th>URL</th>
<th>Method</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>http://&lt;Scania web server Host/IP&gt;:&lt;Port&gt;/&lt;resource&gt;/:partnumber</td>
<td>GET</td>
<td>partNumber</td>
<td>Get the part details along with publishing status, denomination details and part axis values. Also, the</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>Action</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>partNumber with [optional] version</td>
<td>Get the details of a specific part number with the given version passed via URL. Also, it provides the URL to the file location in Vault as (3D Jpeg format and Catia Graphical Representation - CGR is a CAD file created by CATIA)</td>
<td></td>
</tr>
<tr>
<td>PUT</td>
<td>-</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>-</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>-</td>
<td>n/a</td>
<td></td>
</tr>
</tbody>
</table>

**URL:**

- `http://<Scania web server Host/IP>:/<resource>/:partnumber[&version]`
- `http://<Scania web server Host/IP>:/<resource>/:partnumber[&version&drawingtype]`

**GET**

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partNumber with [optional] version and drawingtype=drw</td>
<td>Get the details of a specific part number with the given version and drawing type passed via URL. Also, it provides the URL to the file location in Vault as (3D Jpeg format and CGR file format)</td>
</tr>
<tr>
<td>PUT</td>
<td>n/a</td>
</tr>
<tr>
<td>POST</td>
<td>n/a</td>
</tr>
<tr>
<td>DELETE</td>
<td>n/a</td>
</tr>
</tbody>
</table>
corresponding to the given version passed via URL. Also, the service provides the URL to the file location in Vault i.e. returns all the detailed drawing sheets with version number corresponding to the given part number.

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT</td>
<td>-</td>
<td>n/a</td>
</tr>
<tr>
<td>POST</td>
<td>-</td>
<td>n/a</td>
</tr>
<tr>
<td>DELETE</td>
<td>-</td>
<td>n/a</td>
</tr>
</tbody>
</table>

From the above exposed interface operations it is clear that REST services are affected by the operation parameter structures thereby demanding a careful examination of the parameter structures passed to and from the operations. However, compared to the message-centric approach (SOAP) constrained interfaces (REST) do provide a coarse-grained overview of the service functionality.
Chapter 7

Results and Evaluations

This chapter compares the results of both the Reuse and Migration implementation approaches.

7.1 Reuse Approach

The following output is obtained as a result of reuse approach.

The CAA architecture has an advanced command line interface, which helps in interacting with the CAA web services. The steps for the real-time scenario usage of the system comprises of

- Instantiating and configuring CAA Web service proxy
- getUserContexts – retrieves the contexts associated with a given user declared in the People and Organization database
- bindToApplication - log on to ENOVIA LCA V5 using a specific context
- releaseFromApplication - log off from ENOVIA LCA V5 in order to terminate an opened session.

Figure 7-1 depicts expected output and Figure 7-2 depicts actual output. The diagram shows actual output results in “{400} Bad request” error is different from expected output. The occurred error could not be further debugged because the needed functionalities were vendor locked.
Displaying inputs

Web application context root URI: http://<Scania web server Host/IP>
ENOVIA LCA V5 P&O username: <Enovia user name>
Basic authentication username:<web server user name>
Basic authentication password: <web server password>

Running ENOPosApplicationBinderImpl CAA Web service use case

STEP 1: Instantiating and configuring CAA Web service proxy

WARNING: Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.

STEP 2: Getting contexts for user specified

>>> Invoking 'getUserContexts' on ENOPosApplicationBinderImpl CAA Web service proxy
Username: <Enovia user name>
Reading 2 cookies from HTTP response
Cookie[0]: LtpaToken=mVD8PxR7Vah08zM+h9BOQ3px7/nnOpCi0cu2JSDBcvbyT5nKwMpMWVvK1Huaxo...
Cookie[1]: JSESSIONID=0000-JrFl00KK4Lrnz7Y0M+WuYt-:-1
Storing cookies in memory. Client ID: 1138294673313
Status: Success
Found 1 context(s) in database
Displaying list of available contexts:
Context[0]: VPMADMIN.ADMIN.DEFAULT

STEP 3: Logging on to ENOVIA LCA V5

>>> Invoking 'bindToApplication' on ENOPosApplicationBinderImpl CAA Web service proxy
Using context: VPMADMIN.ADMIN.DEFAULT
Reading cookies in memory. Client ID: 1138294673313
Setting cookies on HTTP request:
LtpaToken=mVD8PxR7Vah08zM+h9BOQ3px7/nnOpCi0cu2JSDBcvbyT5nKwMpMWVvK1Huaxo...
JSESSIONID=0000-JrFl00KK4Lrnz7Y0M+WuYt-:-1
Status: Success

STEP 4: Logging out from ENOVIA LCA V5

>>> Invoking 'releaseFromApplication' on ENOPosApplicationBinderImpl CAA Web service proxy
Reading cookies in memory. Client ID: 1138294673313
Setting cookies on HTTP request:
LtpaToken=mVD8PxR7Vah08zM+h9BOQ3px7/nnOpCi0cu2JSDBcvbyT5nKwMpMWVvK1Huaxo...
JSESSIONID=0000JrFl00KK4Lrnz7Y0M+WuYt-:-1
Status: Success

Use case execution successful

Figure 7-1: Reuse approach expected result
Displaying inputs

Web application context root URI: http://<Scania web server Host/IP>
ENOVIA LCA V5 P&O username: <Enovia user name>
Basic authentication username:<web server user name>
Basic authentication password: <web server password>

Running... ENOPosApplicationBinderImpl CAA Web service use case

STEP 1: Instantiating and configuring CAA Web service proxy

Mar 23, 2015 1:10:55 PM org.apache.axis.utils.JavaUtils isAttachmentSupported
WARNING: Unable to find required classes (javax.activation.DataHandler and javax.mail.internet.MimeMultipart). Attachment support is disabled.

STEP 2: Getting contexts for user specified

>>> Invoking 'getUserContexts' on ENOPosApplicationBinderImpl CAA Web service proxy
Username: <Enovia user name>

Use case execution failed

Failed to get user contexts
Exception details:
AxisFault
 faultCode: {http://xml.apache.org/axis}/HTTP
faultSubcode:
faultString: (400)Bad Request
faultActor:
faultNode:
faultDetail:
{http://xml.apache.org/axis}/HTTPErrorCode:400

{400}Bad Request
at org.apache.axis.transport.http.HTTPSender.readFromSocket(HTTPSender.java:744)
at org.apache.axis.transport.http.HTTPSender.invoke(HTTPSender.java:144)
at org.apache.axis.SimpleChain.doVisiting(SimpleChain.java:118)
at org.apache.axis.SimpleChain.invoke(SimpleChain.java:83)
at org.apache.axis.client.AxisClient.invoke(AxisClient.java:165)
at org.apache.axis.client.Call.invokeEngine(Call.java:2784)
at org.apache.axis.client.Call.invoke(Call.java:2767)
at org.apache.axis.client.Call.invoke(Call.java:2443)
at org.apache.axis.client.Call.invoke(Call.java:2366)
at org.apache.axis.client.Call.invoke(Call.java:1812)
Status: Failure

Use case execution failed

Figure 7-2: Reuse approach actual result
Conclusion
This approach not only strengthens the business case reuse and encapsulation of technical details, but they also validate and quantify benefits of SOA as opposed to development from scratch, which results in potential savings. However, the results observed implies that due to technical constraints imposed by ENOVIA systems, impairs the reuse of legacy assets as services from both strategic and technological perspective.

7.2 Migration Approach

The following output is obtained as a result of migration approach

Testing the ENOVIA web services API using Postman REST client

Postman REST client is a Google Chrome App for direct HTTP resource discovery, manipulation and testing. This tool provides an easy way to perform HTTP requests and view the responses.

- Install Postman REST Client
- Launch Postman REST Client
- Point to Enovia Web Services and make API calls

Pre-requisites

The services are tested using Google Chrome browser app Postman REST client.

Install Postman REST client

- Using a Google Chrome browser, go to the following URL:
  https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcdcbncdd domop
- Follow the instructions from Chrome to install Postman REST client.

Launch Postman REST client

- Open a new Chrome tab.
- In the list of Apps, click the Postman REST client.
Point to ENOVIA Web Services and make API calls

- Perform any of the queries as described in the Table 6-3 API documentation.
- Enter the URL as <host>:<port>/ Where <host> is the hostname or virtual IP address of the machine pointing to the service where it is hosted, and <port> is the HTTP port used to connect to the Enovia web services API (typically 8080)
- Select the HTTP action (GET/POST/PUT/DELETE)
- Verify that the headers, if any (the services does not use any HTTP headers).
- Click Send.

Figure 7-3 shows an HTTP GET request to the ENOVIA Rest API <url>/getpartmasterdetail which takes in unique part number as the input and returns corresponding key/value pair comprising of version, description, organization, cgr3Durl, jpeg3Durl, jpegDRWUrl, status, partNumber, partType, material and geoPublishStatus.

Figure 7-3: REST API output response from POSTMAN for unique partnumber
Figure 7-4 shows an HTTP GET request to the ENOVIA Rest API <url>//getpartmasterdetail which takes in the part number and unique version as the input and returns corresponding key/value pair comprising of version, description, organization, cgr3Durl, jpeg3Durl, jpegDRWUrl, status, partNumber, partType, material and geoPublishStatus.

**PREVIEW**

GET /getpartmasterdetail?version=1 HTTP/1.1
Host: 8080
Cache-Control: no-cache

**Body**

```
{
    "version": "1",
    "description": "PREVIEW",
    "organization": "",
    "cgr3Durl": "http://www.data/vault/xxxxxx_Vault",
    "jpeg3Durl": "http://www.data/vault/xxxxxx_Vault",
    "jpegDRWurl": "",
    "status": "InWorkFrozen",
    "partNumber": "xxxxxx",
    "partType": "A",
    "material": "",
    "geoPublishStatus": "F"
}
```

Figure 7-4: REST API output response from POSTMAN for part number and unique version

**Figure 7-5:** REST API output response from POSTMAN for part number, unique version and drawing type shows an HTTP GET request to the ENOVIA Rest API <url>//getpartmasterdetail which takes in part number, unique version and drawing type as the input and returns corresponding key/value pair comprising of version, description, organization, cgr3Durl, jpeg3Durl, jpegDRWUrl, status, partNumber, partType, material and geoPublishStatus.
The Table 7-1 gives a reasonable idea why Migration approach is advantageous over Reuse approach for the prototype implementation.

Table 7-1: Comparison Reuse vs. Migration Approach

<table>
<thead>
<tr>
<th>Property</th>
<th>Discussion parameters</th>
<th>Implementation using Reuse approach</th>
<th>Implementation using Migration approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legacy System Architecture</td>
<td>Architecture views and system documentation</td>
<td>Limited system documentation</td>
<td>The reverse engineering technique (code analysis) helped in understanding the system's components and their interrelationships. This facilitated in creating</td>
</tr>
<tr>
<td>and Characteristics</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7.3 Comparison Reuse vs. Migration Approach

The Table 7-1 gives a reasonable idea why Migration approach is advantageous over Reuse approach for the prototype implementation.
<table>
<thead>
<tr>
<th>Degree of coupling(^8) and system dependencies</th>
<th>Tight coupling between the components and the complex interactions between objects made it difficult to extract necessary business logic to represent a service. Since service represents a particular functionality in a business process.</th>
<th>A decomposition of the legacy system through reverse engineering facilitated in composing services with high cohesion and loose coupling.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality attributes</td>
<td>Quality of Service – Performance and Response time</td>
<td>Applying reuse strategy requires an additional resource in terms of hardware and technologies to enable such service orchestration. In addition, there is a difficulty in identifying the requests, which can be processed sequentially or in parallel due to complex nature of service hosting environment.</td>
</tr>
<tr>
<td>High-Level Understanding of Target SOA Environment and Architecture</td>
<td>Target SOA architecture</td>
<td>As discussed earlier, the absence of logical and development views of the system architecture failed to devise a reference model that</td>
</tr>
</tbody>
</table>

\(^8\)https://en.wikipedia.org/wiki/Coupling_(computer_programming) – Coupling refers to the degree to which software components depend on each other.
could capture solution related knowledge such as architecture, business processes and business rules. This resulted in a risk of not providing appropriate direction towards developing or extending service orientation of existing components through reuse implementation approach. Techniques, which migrate these legacy components into meaningful business services are well constructed by effectively exploiting the architectural knowledge of the target system. Furthermore, the services adhere to Scania SOA principles and guidelines.
Chapter 8

Conclusion

This chapter concludes the thesis with major insights from the research study, solutions to the research questions and the research goal and suggestions for future research.

8.1 Conclusion and Future Work

Migrating to a service-based system can help an organization evolve its IT system by helping in achieving business agility. But this migration comes with its own set of challenges, however if done right, developing these services that can be reused across different applications can help organizations in saving time and money while maintaining the legacy system and developing new functionality to meet changing business needs. It can also improve the useful life of their legacy systems, by reusing the business critical functionalities wherever possible. The goal of this thesis focuses on finding various ways to evolve Scania’s IT systems, more specifically to implement web service interfaces for the existing legacy system.

In order to migrate legacy to SOA modernization, a literature review was conducted about existing technologies and approaches. From literature study, three various plausible methods Reuse, Migration and Wrapping were selected to achieve the goal of the thesis work. Among those three, wrapping method was not selected for further investigation because of the complicated process involved in extracting and decoupling all the necessary code required for creating an appropriate service. Additionally, the core CAA Web Service API needed to build the service interface was vendor locked. The other methods Reuse and Migration approach were prototyped and compared for efficiency. From the research, it is understood that migration method has more pros, despite the security feature being hardcoded.

Finally, migration approach was chosen to be used for this thesis by comparing the maturity, applicability, adaptive and flexibility nature of the legacy system to ease such migration. The developed prototype was tested using Postman REST client and REST assured library, which simplifies automated testing of such REST based services.
Nevertheless, despite the working prototype that was created, there is a need for objective metrics to measure the quality attributes such as security, performance and reliability of the resulting services as mentioned in the business objectives.
APPENDIX A

Scania Java Reference Architecture

In order to adhere to the ever-changing business requirements, Scania Java Management Group has come up with a Java Reference Architecture, which is a recommendation of tools and frameworks that compose a working architecture. The reference architecture comprises a set of architectural best practices for usage across all teams within the enterprise. The reference architecture is based on the N-tier architecture consisting of persistence, business, service and presentation layers that are well suited for designing and constructing an enterprise application including the development of reusable business services. These services facilitate in building applications by composition; wherein services are combined to produce another reusable service. The key architecture goal is to leverage industry best practices for designing and developing highly available and scalable applications.

The core guiding principles of Scania Java Reference Architecture are:

- The Architecture supports the notion of reusable software components
- The Architecture provides the flexibility to extend and adapt to future IT infrastructure while minimizing the impact to the existing line of business
- The Architecture provides separation of concerns through logical layering (decoupled vs. coupled design) and application partitioning for maintainability i.e. separating the user interface from the business logic and the business logic from the data store
- The Architecture enables the business application to scale, to be continuously available and to provide the opportunity to tune for performance
- The architecture allows different components to be deployable and testable independently
- The Architecture leverages on existing IT infrastructure, assets and skill sets where appropriate
- The Architecture supports the physical distribution across different tiers
The Architecture incorporates “defensive coding” designs, providing confidentiality and integrity.

**Conceptual view:**
The service-based architecture implementation using Java Reference Architecture significantly elevates the abstraction level for code re-use, enables better application binding to services that evolve and improve over time without requiring modification to the applications that consume them. Services provide a simple model to integrate software systems both inside the organization and across organizational boundaries.

ENOVIA services are implemented using a combination of business logic, business rules and business processes and capable of supporting different Scania business units. These services are exposed via the Scania application servers. Appendix A-1 illustrates this.

---

ENOVIA web service interface meets this requirement by providing a set of services, which it considers core to any Scania application – this core set of services are described in Table 6-3.

Logical View:
The logical view builds on the conceptual view providing a detailed description of key elements of the architecture. It translates the functions identified in the conceptual view into an abstract model that identifies the cooperating logical components. It describes the grouping of design elements (classes and interfaces) into packages represented as namespaces. Also, it describes the static and dynamic relationships between the classes.

While the conceptual view described at a high level, how a user’s request moves through the elements of the web service architecture, the logical view introduces significant components of the framework, describes how they work and discusses specific framework properties and issues.

The main sections of the logical view discussion are:

Logical Layers: The target system is designed according to the general principals of n-tier design, with logically similar components being grouped together in layers.

Significant design elements: These are the classes and interfaces that are principally responsible for processing both client and service requests.

Infrastructure services: The framework provides infrastructure services, such as exception handling, security, and transactions. depicts the logical view of the system and to be architecture is illustrated in.
Appendix A-2: Logical Architecture

*Note: The architecture is based on Java reference architecture from JMG.*

Appendix A-3: TO-BE Architecture
REFERENCES


74


Rosen, M., & Books24x7, I. (2008). Applied SOA: service-oriented architecture and design strategies. Indianapolis, IN: Wiley Pub. Retrieved from http://uu.summon.serialssolutions.com/2.0/linx0/eLvHCXMwdV3PT8IwFH4BP-CgXFUumGlh3ENq645oIMYEPcjJS70OLiEssNjN80fb7gfZE1_d65K-1-319b1-XwEwGo2HRz4hsFeBLXYPlc-hzgliXwhhBCRXIFhKnuZ3FF5us7cFm9WggKIUhznWsRolKUt7D1fyRprFckv9J6WuyrsyL9mbp2HepsYq53mD7jQx7G1Koo


