A STEP Implementation For
Product Data Exchange

Sairam Ghanta
ABSTRACT

SCANIA performs automobile production based on a Modular Approach. A Modular System provides a huge variety of specifications in order to meet the variety of different requirements in relation to different customers. A Modular approach, with the support of reusable components, increases the efficiency in relation to designing different products.

The main stream product design is performed on a CATIA V5 platform with ENOVIA as its PDM vault. However, SCANIA uses its own legacy PDM SPECTRA to maintain the product structure for its modular based product specification.

Thus, in order to design the 3D parts in CATIA, the product structure should be imported from SPECTRA to ENOVIA. This process will, however, include a conversion from the SPECTRA format to the ENOVIA format and, currently, this is performed using a third party component named ECCO. It is now the case that, for a variety of reasons, SCANIA wants to replace ECCO with an in-house build module.

This report discusses the background knowledge relating to the problem, the methodology used in the solution approach, different implementations together with their results and finally concluding with options for future work.

Keywords: CATIA, ENOVIA, CAA, JSDAI, ECCO, EXPRESS, EXPRESS-X
ACKNOWLEDGEMENTS

Everyone has a dream and for me, this was fulfilled by working with a World renowned Automobile Manufacturing Company, SCANIA. I would like to take this part of my thesis document to express my gratitude, appreciation and humble THANKS for every individual in the ILES group who contributed to this project directly or indirectly and made a dream come true.

Firstly I would like to THANK Rodrigo Cid, the ILES manager, who considered me for this thesis job and making me feel completely at ease with his warm welcoming nature and for the time he invested in introducing me to SCANIA and its work culture. His belief in me was by far, the most invaluable gesture that I have ever come across in my academic and professional life.

Our Supervisor, Andreas Nordgren, at SCANIA, contributed significantly with his suggestions and support. He also played a vital role by his assistance in providing a definitive form to the idea and proposed Thesis work. A big THANKS to him for his valuable time in answering my endless questions and clearing the doubts regarding the technical aspects of this Thesis Job.

I also want to THANK my Professor, Tinting Zhang for her support throughout my Masters Program at Mid-Sweden University. I also want to thanks my Supervisor, Wei Shen, for the valuable support in relation to reviewing this report.

JingJing Qian, my colleague and thesis partner during my time at SCANIA, her enormous amounts of patience in considering seriously whatever proposals or approaches I randomly suggested even though some were in reality more “Food for thought” types. I express my earnest regards and best wishes to her for her future. A big THANKS to the confidence and trust she had put in me.

Last but not least, to my brother Arun Kamal Ghanta, for accepting to review this Thesis Documentation despite the entire project sounding like Greek & Latin to him in relation to the technical information and jargon used.
This report has been documented as a final part of my education in relation to Master of Computer Science at Mid-Sweden University, Sundsvall. It is worth 30hp credits and constitutes one semester's work. From the 1st November 2011 to the 31st March 2012 this project was conducted with the ILES Group at SCANIA tekniskt centrum, Södertälje. This project evolved in multiple cycles, each of which included designing, implementation and testing. With limited resources available in modules such as EXPRESS, EXPRESS-X, EXPRESS-C and CAA SDAI, the focus of the team was to learn the information concerning each module before continuing onto the next one even though this proved to be somewhat challenging. This project has assisted in my understanding of the European IT Corporate Environment and Software Development strategies and, in particular, the spirit of "GO LIVE" which was both interesting and inspirational.

All-in-all, it's been a great honor to be part of Infomate and tekniskt centrum.

Södertälje
May 14, 2012
Sairam Ghanta
Table Of Contents

Abstract i

Acknowledgements ii

Preface iii

Terminology / Notation ix

1 Introduction 1
   1.1 Background and problem motivation . . . . . . . . . . . . . . . . . . 1
   1.2 Overall aim . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
   1.3 Scope . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
   1.4 Concrete and verifiable goals . . . . . . . . . . . . . . . . . . . . . 3
   1.5 Outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
   1.6 Target Audience . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
   1.7 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
   1.8 About SCANIA . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2 Theory 5
   2.1 Product Structure . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
      2.1.1 Product Structure Modelling . . . . . . . . . . . . . . . . . . 5
      2.1.2 Breakdown . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   2.2 System Components . . . . . . . . . . . . . . . . . . . . . . . . . . 6
      2.2.1 SPECTRA . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
      2.2.2 CATIA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
      2.2.3 ENOVIA . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
      2.2.4 SIAM . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
      2.2.5 JavaMigrator . . . . . . . . . . . . . . . . . . . . . . . . . . 9
      2.2.6 ECCO . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
      2.2.7 Control-M . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
   2.3 System Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . 11
      2.3.1 XML file . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
      2.3.2 Scania STEP file . . . . . . . . . . . . . . . . . . . . . . . . 11
      2.3.3 ENOV_DATAX STEP file . . . . . . . . . . . . . . . . . . . 14
      2.3.4 Provision Document . . . . . . . . . . . . . . . . . . . . . . 14
# Table Of Contents

## 2.4 STEP Standard
- 2.4.1 Part 11, EXPRESS ........................................ 15
- 2.4.2 STEP P21 File ............................................. 16
- 2.4.3 Part 22-27 SDAI & JSDAI ................................. 17
- 2.4.4 Express-X .................................................. 17
- 2.4.5 EXPRESS-G ................................................ 21
- 2.4.6 EXPRESS-C ............................................... 21
- 2.4.7 EXPRESS-I ................................................ 22
- 2.4.8 EXPRESS-V ............................................... 22
- 2.4.9 EXPRESS-M .............................................. 23

## 2.5 STEP Implementation Environments
- 2.5.1 ECCO Toolkit ............................................ 24
- 2.5.2 JSDAI, Part-27 ............................................ 26
- 2.5.2.1 Java Representation .................................... 27
- 2.5.2.2 Management Classes and Auxiliary Objects ........ 28
- 2.5.2.3 JSDAI Components ..................................... 28
- 2.5.3 CAA/C++ .................................................. 30
- 2.5.3.1 CAA RADE ............................................ 33
- 2.5.3.2 CAA SDAI ............................................. 33
- 2.5.3.3 CAA EXPRESS-X Usage & USER FUNCTIONS ........ 34

## 3 Methodology .................................................. 38
- 3.1 Design Flow .................................................. 38
- 3.1.1 System Level ............................................. 38
- 3.1.2 Lower Level ............................................... 40

## 4 Implementation & Results ..................................... 42
- 4.1 CAA + CAA .................................................. 42
- 4.1.1 Implementation .......................................... 43
- 4.1.1.1 CAA HASHTABLE Limitation ......................... 44
- 4.1.1.2 No STEP import Support ............................. 44
- 4.1.1.3 Provision Document .................................. 45
- 4.1.1.4 Limited EXPRESS-X .................................. 45
- 4.1.1.5 No CAA Execution For Mapping ................... 45
- 4.2 JAVA + CAA ................................................ 47
- 4.2.1 Java Representation ..................................... 48
- 4.2.2 NEW SCANIA Schema .................................... 48
4.2.3 Simplified EXPRESS-X ........................................... 50
4.2.4 Provision Document ........................................... 50
4.2.5 Implementation ................................................ 50
  4.2.5.1 Java Part ................................................ 51
  4.2.5.2 CAA Part ................................................ 53
4.2.6 System Evaluation ............................................ 57
  4.2.6.1 Collections ............................................. 57
  4.2.6.2 Performance .......................................... 57
  4.2.6.3 Functional testing ..................................... 58
  4.2.6.4 System Acceptance .................................... 58
  4.2.6.5 Testing Mapping Specification ......................... 58
4.3 CAA + CAA ......................................................... 59
  4.3.1 C++ STL Containers ....................................... 60
  4.3.2 No New Schema ........................................... 60
  4.3.3 User Functions ........................................... 60
  4.3.4 Provision Document ....................................... 61
  4.3.5 IEnovIn Script for Import Process ....................... 61
  4.3.6 Implementation ........................................... 61
    4.3.6.1 CAA(XML TO SCANIA STEP) ......................... 62
    4.3.6.2 CAA(SCANIA STEP to ENOVIA) ..................... 65
  4.3.7 System Evaluation ....................................... 68
5 Conclusions & Future Work ....................................... 69
  5.1 Outcome ...................................................... 69
  5.2 Feasibility .................................................. 69
  5.3 Impact on Future .......................................... 70
  5.4 Alternative ................................................ 70
6 APPENDIX .......................................................... 72
  6.1 Existing Architecture Sequence Diagram With ECCO ........ 72
References .......................................................... 73
# List Of Figures

1.1 High-Level Problem .............................................. 2  
1.2 Source file and target file Architecture .......................... 3  
2.1 Product Structure Example ........................................ 6  
2.2 AROS application .................................................. 7  
2.3 Example of CATIA .................................................. 8  
2.4 Example of ENOVIA ................................................ 8  
2.5 SIAM MQ ........................................................... 9  
2.6 Overview of JavaMigrator ........................................... 10  
2.7 Architecture Overview ............................................. 12  
2.8 Graphical Representation of SCANIA schema ..................... 13  
2.9 Graphical Representation of ENOVIA schema .................... 14  
2.10 EXPRESS-X Mapping .............................................. 18  
2.11 An Example Express-G(persons) ................................... 21  
2.12 EXPRESS-C Compiler: Input and output files (C++ code generation) \[12\] .................................................. 25  
2.13 JSDAI Framework Usage .......................................... 26  
2.14 JSDAI Compiler: Input and output files (java libraries) \[3\] .............. 27  
2.15 JSDAI Eclipse based Express Editor \[4\] ......................... 30  
2.16 JSDAI Eclipse based Express-G Editor \[4\] ....................... 31  
3.1 System Design Flow ............................................... 39  
4.1 A complete CAA based Proposal ................................... 43  
4.2 XML Data with complex relationships ............................ 43  
4.3 Problem Generating the Intermediate STEP file ................... 44  
4.4 Problem importing the Intermediate STEP file and Executing Mapping module Manually ................................. 45  
4.5 Complex Mapping between SCANIA Schema and ENOVIA Schema 46  
4.6 JSDAI and CAA Based Proposal ................................. 48  
4.7 JavaModule Class Diagrams ....................................... 49  
4.8 Java Part Sequence Diagram ..................................... 52  
4.9 IEEnovIn script with Mapping module ........................... 54  
4.10 CAA Part Sequence Diagram Using IEEnovIn Script .......... 55  
4.11 Error Indicating the Failure in IEEnovIn script ................. 56
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.12</td>
<td>Final proposal Based on CAA using C++ Containers and IEnovIn Script</td>
<td>59</td>
</tr>
<tr>
<td>4.13</td>
<td>CAA-Module Class Diagram</td>
<td>63</td>
</tr>
<tr>
<td>4.14</td>
<td>CAA First Part Sequence Diagram</td>
<td>64</td>
</tr>
<tr>
<td>4.15</td>
<td>IEnovIn script with Mapping module</td>
<td>66</td>
</tr>
<tr>
<td>4.16</td>
<td>CAA Second Part Sequence Diagram</td>
<td>67</td>
</tr>
<tr>
<td>6.1</td>
<td>Spectra Breakdown</td>
<td>72</td>
</tr>
</tbody>
</table>
TERMINOLOGY / NOTATION

Acronyms / Abbreviations

- **AIX** | Unix Operating System from IBM
- **API** | Application Programmer Interface
- **CAA** | Component Application Architecture
- **CATIA** | Computer Aided Three dimensional Interactive Application (CAx System)
- **CAx** | Computer Aided X-System
- **CBSE** | Component Based Software Engineering
- **Control-M** | Job Scheduler software from BMC
- **DS** | Dasalut Sytemes
- **ECCO** | Software Component from PDTech.
- **ENOVIA** | Enterprise InNOvation VIA DASSAULT SYSTEMES (PLM for CATIA)
- **IDE** | Integrated Development Environment
- **ISO** | International Organization for Standard
- **JSDAI** | Software component from LKSoft
- **MQ** | Message Queue
- **NC** | Numerical Control
- **PDE** | Product Data Exchange
- **PDM** | Product Data Management
- **PLM** | Product Lifecycle Management
- **SDAI**: Standard Data Access Interface (Part 22)
- **SIAM**: SCANIA’s integration platform
- **SPECTRA**: SCANIA’s legacy PDM
- **STEP**: STandard for the Exchange of Product model data
- **XML**: eXtensible Markup Language
1 INTRODUCTION

1.1 Background and problem motivation

SCANIA is one of the leading automotive manufacturers in Europe, offering both production and services to its variety of customers. One of the success points of this organization is its Modular Approach. It produces the majority of its vehicle parts but will, additionally, purchase a few from its trusted clients. It assembles these parts as per the specifications of its customer and these are converted, at a later stage, into a Product Structure. It has its own, in-house Product Data Management (PDM) system, named SPECTRA, which holds an enormous amount of Product Structure information.

However, ENOVIA, as is the case for the CAx environment, has its own PDM called ENOVIA. To maintain both systems in their environment, data from SPECTRA is migrated to ENOVIA once a breakdown has been ordered. To design a CAD part, the designer should order a breakdown from SPECTRA and should work on that particular product/part structure. At a later stage, these design parts are stored in the ENOVIA vault for further processing including part authorization, integration, simulation, testing etc. Once a breakdown has been ordered, an XML file is generated, which is very complicated in nature.

In order to process this XML into the ENOVIA format, two components named JavaMigrator and ECCO have been used. JavaMigrator is an in-house developed module that processes XML to ECCO objects. ECCO, a software component from PDTech (German Company), processes these ECCO objects and generates a format that ENOVIA can interpret.

However, ECCO does not support the version of AIX in use on the development and production servers. This process is both a serious risk and a major obstacle each time changes are required and the mapping is rebuilt. ECCO requires highly specialized knowledge in order to make changes and is not a feasible solution, in terms of affordability, for this to be part of the development team. It is preferable to use the CAA-Module and hence the IT support team would like to find an ECCO replacement, preferably an in-house developed module in CATIA/ENOVIA customization.
1. Introduction

API CAA/C++.

1.2 Overall aim

In the current system, ECCO is completely responsible for the data translation from the SPECTRA format to the ENOVIA format. ECCO sits alongside the JavaMigrator and co-ordinates with JavaMigrator to process the SPECTRA data so as to generate an intermediate SCANIA format and, further, to target the ENOVIA format. ENOVIA understands the target format and can import the data represented in this format into its vault. SCANIA has designed the format structure for a SCANIA intermediate file and target an ENOVIA file. A valid mapping specification must be used to translate the data between these two formats. However, there are a few problems associated with the current environment for ECCO and it has to be replaced with an in-house written CAA/C++ module. The ECCO specific mapping specifications should be re-written in order to comply with CAA compiler standards. Figure 1.1 represents a high-level view of the problem statement.

Figure 1.1: High-Level Problem
1.3 Scope

The scope of the project is to a) study the current implementation of STEP using ECCO b) develop an CAA-module/similar component c) show whether it is feasible to replace the ECCO component and d) evaluate the commercial component if exists similar to ECCO as a substitute and if not possible in CAA.

1.4 Concrete and verifiable goals

The goal of this work is to process the XML format to create a SCANIA specific STEP file, whose EXPRESS schema is then mapped to a EnovDataX schema, and which is finally imported as a STEP file to ENOVIA. The low level goal is to generate the target file from the source files and intermediate file by using the new module. The following is a description of the file architecture in Figure 1.2.

![Figure 1.2: Source file and target file Architecture](image)

1.5 Outline

The whole report is structured in six chapters.

**Theory:** discusses the theory behind the problem.

**Methodology:** discusses the strategic approach followed to discover a solution in order to replace the current component.

**Implementation & Results:** discusses the implementation details of the work together with the results. There were different solution proposals which were performed at different stages of the project. Each solution involved a variety of obstacles that lead to a new proposal. It is felt that the most sensible solution is to include both the implementation and results in one chapter in order to provide greater clarification for the readers.

**Conclusion & Future Work:** discusses the conclusions considering various factors and future work on this project based on each derived conclusion.
Appendix: contains UML diagrams of the different implementations in addition to the existing work flow.

1.6 Target Audience

This report is aimed at those readers who have sufficient knowledge in Software Development and it would be advantageous to possess a working fair knowledge of C++ and Java.

1.7 Contributions

This project is has been conducted by the author of this work and JingJing Qian, a student form Linköping University. The entire work was conducted incrementally as was demanded by both the problem and limited resources available. The solution to the system was designed after conducting collaborative research. Instead of dividing the tasks into different parts, the approach towards the solution was a collaborative one in relation to each problem as it arose.

1.8 About SCANIA

SCANIA is one of the world’s leading automobile manufacturer and whose main production is in relation to trucks, transportation buses and marine motors. It also provides products and financial services to its customers. This project work was conducted at SCANIA TEKNISKT CENTRUM in Södertälje, in the ILES group. This group handles the most of the CAx red arrow processes in relation to Infomate.

Infomate is an SCANIA IT company providing IT solutions to SCANIA. A part of Infomate, ILES is located in the SCANIA TEKNISKT CENTRUM.
2 THEORY

This chapter discusses the Product Structure, System Components, System Architecture, STEP standard and available STEP implementation environments. The architecture consists of many components but this discussion of them is limited to those involved within the scope of this thesis work. Background knowledge of these concepts is provided in order to assist the reader in an understanding of the later chapters.

2.1 Product Structure

2.1.1 Product Structure Modelling

Product structure is a hierarchical decomposition of a product, typically known as the bill of materials (BOM) [18]. Decomposition consists of an assembly which in turn, which consists of the decomposition of several parts of a product which form relationships in the complete product structure. Each individual part in a product structure might have interfaces to other parts in the structure in a hierarchical arrangement. In SCANIA, decomposition works in a modular fashion and thus offers scope to these individual parts to be part of any product developed in SCANIA. For example, an engine designed in SCANIA can be assembled in a truck or bus produced in SCANIA. Such a generic design assists in covering the wide variety of customers in the market and reduces the effort relating to an individual product. 2.1 provides a simple example in order to describe the product structure.

2.1.2 Breakdown

Product breakdown is a collection of parts or a complete product structure. Breakdown describes the list of product parts together with their relationships in the scope. In SCANIA, the 3D designing of various automotive vehicles depends on this breakdown, while retaining same the underlying product information for all products. For example, an engine in a vehicle together with its interfaces in the product structure is a breakdown and might contain parts such as screws, bolts and pipes.
2. System Components

The architecture consists of various systems interconnected to each other in order to perform multiple tasks on demand. The main system components that play a part in this thesis are SPECTRA, CATIA, ENOVIA, SIAM, JavaMigrator, ECCO and Control-M which are briefly described in the following section.

2.2.1 SPECTRA

SCANIA’s legacy PDM system contains product information including the product structure, product configurations and positioning within the mainframe environment Aros/Rosam (Figure 2.2) [15]. It was developed at SCANIA R&D. It is an application for handling conditional and time dependent product structure and can be considered as a backbone to CAD activities in SCANIA. It manages the product structure for SCANIA Truck, Bus/Coach and Engines. SPECTRA uses ATS as its interface and this uses SQL query language to obtain the geometrical data stored in it by external applications. The 3D design parts are stored in ENOVIA and the 2D design parts in ModArc. The information which relates the concern between these parts is stored in SPECTRA. These relations are imported to ENOVIA if a breakdown is ordered by CATIA user. When an external application orders the breakdown, the product structure in the Spectra database is extracted in the form of an XML file containing the information of the user ordering the breakdown, the parts and the geometrical information of these parts and the relationships between these parts in the breakdown.
2.2.2 CATIA

CATIA is the multi-platform CAx system developed by Dassault Systems which provides functionalities to users like 3D modeling, cabling design, surface modeling, NC processing, analysis and product simulations. These multi functionalities support product engineering from initial specification to product-in-service. CATIA is widely used in engineering industry especially, in automobile and aerospace domains. SCA-NIA initially used CATIAv4 which is a 2D modeling system. A project named Carina replaced most of CATIAv4 systems with CATIAv5. The user interface of CATIA is provided in windows with either collaboration or interactive sessions running on the server. To design a 3D model in CATIA, user has to import Product Information into its Workspace. In SCANIA, this can be done by ordering a Breakdown from SPECTRA. A simple user interface of CATIA can be seen in Figure 2.3.

2.2.3 ENOVIA

ENOVIA is a PDM system (preferably called PLM System) developed by the Dassault Systems as a solution for Information Management in relation to CATIA designs and parts fabrication. This is built on the Oracle RDBMS. ENOVIA repositories include information about space modeling, configuration of digital models and process knowledge from an initial specification to a product-in service [2]. SCANIA customized ENOVIA to store only the information about parts in this vault, whereas the relationships between these parts are stored in SPECTRA and are imported to ENOVIA on demand. Prior to the Carina project a Matrix system named ModArc was
implemented for CATIAv4 data maintenance and this co-exists in the current architecture with ENOVIA. ModArc and its existence fall outside the scope of this project. A typical ENOVIA V5 interface has an appearance as shown in example Figure 2.4.
2.2.4 SIAM

It is the common name of the integration platform used at SCANIA. SIAM IP is a SCANIA developed system integration software on an AIX platform that ensures that there is communication between ENOVIAv5 and the SIAM platform and handles the transportation of an XML files that are generated in SPECTRA to ENOVIA. It connects to SPECTRA ATS using JDBC, a Java database connector. SIAM IP is built on an IBM MQ API with an FTP adapter 2.5. The generated XML files in SPECTRA are moved to SIAM IP servers by the FTP adapter and are placed on the MQ queue [16].

![SIAM MQ Diagram]

Figure 2.5: SIAM MQ

2.2.5 JavaMigrator

As the name suggests, this component is completely written in the Java programming language. It has many responsibilities and can be considered as the heart of the SCA-NIA System. This exams work is concerned with the data migration from SPECTRA to ENOVIA in which this software component plays a vital role. It sits on top of the UNIX (AIX) operating system and communicates with SIAM MQ. Its listener starts a transaction once an XML file is identified in the SIAM IP Servers MQ, receives and processes it so as to yield an ENOVIA STEP file with assistance from a component named ECCO. Later, this STEP file is imported to ENOVIA using a Korn-Shell script with the assistance of a Control-M batch scheduler. In order for this system to run, the JavaMigrator must communicate with the Control-M as soon as it receives a transaction. It supports parallel threads running in the system in order to improve the performance. A typical JM interaction with Control-M is illustrated in Figure 2.6.
2.2.6 ECCO

ECCO is an EXPRESS-based toolkit developed by the German company PDTec, which (reference here) is used to provide an environment for specification and manipulation of the mapping between different data models. The ECCO toolkit is the key component in the whole project and section 2.5.1 will provide a detailed introduction regarding the ECCO toolkit and its components. ECCO interacts with the JavaMigrator to process XML data from SPECTRA to a format that ENOVIA is able to understand.

2.2.7 Control-M

Control-M is a workload automation tool from BMC software for scheduling the jobs in an organized flow [19]. It is mainly used to manage load balancing and it is one of the job schedulers used at SCANIA. It conducts a pure scheduling in ENOVIA import by making sure that no two imports are executed simultaneously as this might cause the entire system to crash. These imports are performed in nightly batches and should be in a synchronized fashion. The jobs are wrapped in a Korn-Shell script and are forwarded to the Control-M so that they are executed by the Control-M scheduler.
2.3 System Architecture

This section discusses the high level architecture including all the system components discussed in the previous section. Figure 2.7 offers a high level view to the system and it is currently used in the SCANIA 3D design environment. The main transaction is initiated in SPECTRA once a breakdown has been generated and, at a later stage this breakdown XML file is transferred to the SIAM IP servers MQ by means of an FTP adapter and further received by the JavaMigrator, then processed to ENOVIA format with the assistance of ECCO and imported into the ENOVIA vault by the Control-M job scheduler. In the whole process the breakdown information is represented in three formats: *a) XML file, b) SCANIA STEP P21 file and c) an ENOV_DATAX STEP P21 file.*

2.3.1 XML file

The generated XML file at SPECTRA is in a generic format and is a human readable file which represents the complex product structure including the parts and their relationships. Each node list consists of four child nodes and the complete breakdown information is represented in the form of this generic nodelists. Each nodelist has ParentInstance, ParentType, ChildInstance, ChildType and ChildPos child nodes which describe the information regarding the piece of user information or parts information. Such nodelists have relationships with other similar nodelists in a hierarchical fashion or sibling fashion and, collectively a few groups of this nodelist contribute to a complete part or assembly or user information. If a ChildInstance is not a Parent Instance of another nodelist then the nodelist can be considered as leaf node and has a hierarchical Parent-Child relationship to that nodelist. If more than one nodelist has same ParentInstance value, this group contributes a single amount of information which is divided among these group members. This complex grouping among the Parent-Child and in the siblings contributes to the complete breakdown information.

2.3.2 Scania STEP file

This is the intermediate result generated after the XML file is processed by the JavaMigrator with the assistance of ECCO. This is generated based on an in-house SCANIA designed schema and is very similar to the SPECTRA structure. The generic format in the SPECTRA XML is processed to a reasonable format and where the related information is grouped into a single structure. The reason for the existence of this intermediate STEP file is twofold:

* an independent file, which means any change in the XML file or ENOV_DATAX file would not affect this file and

* once data is represented on the SCANIA Step schema, by means of EXPRESS-X in the second transformation step, it is easier to generate ENOV_DATAX. In
Figure 2.7: Architecture Overview
EXPRESS-G notation, the SCANIA EXPRESS schema appears as that shown in Figure 2.8.

Figure 2.8: Graphical Representation of SCANIA schema
2.3.3 ENOV_DATAx STEP file

This is the acceptable format for the ENOVIA server to import the data into its vault. It typically represents a PDM schema, which has EXPRESS entities describing the PART, ASSEMBLY, DOCUMENT and ROOT of a breakdown. The current implementation has a complex mapping schema written in a combination of EXPRESS-X and EXPRESS-C standards using ECCO to map the data represented in the SCANIA schema so as to target the ENOVIA schema. These standards will be discussed in the next section 2.4. Using this complex mapping specification the source SCANIA data is translated so as to target the ENOVIA format. A graphical representation of SCANIA using the ENOVIA schema is shown in Figure 2.9.

![Figure 2.9: Graphical Representation of ENOVIA schema](image)

2.3.4 Provision Document

A provision document has to be generated in addition to the target ENOVIA format for the imported breakdown. A provision document consists of the user information for the person ordering the breakdown and the parts information together with geometrical data in the Breakdown. This is an important specification but did not exist in the overall system goal but had to be considered as an implicit specification.
2.4 **STEP Standard**

Before discussing the further sections, it is necessary to understand the STEP standard. STEP is an acronym for Standard for Exchange of Product Model Data. STEP is a standard for the computer interpretable representation of Product Data. It is intended to provide a neutral format for product data exchange throughout the product life cycle. The primary goal of STEP is to create a bridge between CAx applications. This bridge can be defined as a standard format for data transfer between these applications. STEP consists of many parts. This standard defines the structure and the architecture in STEP implementations. STEP implementations cover a range from CAx systems, bill of materials system, to stand-alone translators, to packages that make it possible to develop the above system[11]. Generally, this standard is used to define the format for Application Protocols. AP203(Configuration Controlled 3D Assemblies) is one such application protocol designed for the automotive industry. This standard also defines the mechanism for implementing the standard in the form of databases and archives. The parts of STEP that fall within the scope of this work are:

1. **Part 11**: The EXPRESS information modeling language used to describe data models
2. **Part 21**: An EXPRESS-driven clear text encoding of the product data to physical storage files.
3. **Part 22-27**: An EXPRESS-driven application programming interface (SDAI, acronym for Standard Data Access Interface) to perform CRUD operations on EXPRESS defined data. This API is bound to the C (Part 24), C++ (Part 23), and Java languages.
4. **Part 14**: An EXPRESS-driven standard for data translation between different data models defined in EXPRESS language.

### 2.4.1 Part 11, EXPRESS

The EXPRESS standard has evolved over years. The EXPRESS language is an ISO standard (ISO 10303: Part 11) language to describe data models. EXPRESS is a textual data specification language and it defines the data model based on an Entity-Attribute Relationship. EXPRESS includes both generalization and constraint specification in order to restrict the instances. EXPRESS is both computer-interpretable and human readable. This is completely robust and scalable. An EXPRESS schema consists of entities which are made up of attributes, references, aggregates and rules. **Entity** is a whole collection of information which represents conceptual or real objects and it describes value, relationship and structures. **Attribute** refers to the property of the entity or objects. EXPRESS has no execution model and also has no possibility in relation to creating or manipulating data. EXPRESS includes algorithms, expressions
and statements which are used only for constraint specifications. EXPRESS has many built-in functions such as NVL, ABS, EXISTS and SizeOf etc. EXPRESS does not support programming of entities. Here is a simple example that represents the general information of a family.

**Example 2.1: Family Entity**

```plaintext
ENTITY family;
    Number_of_people: INTEGER;
    member: person;
END_ENTITY;
```

In the example, the family entity attribute member is represented by other entity person in the model. The name in the model is represented by a string (character), which is an atomic type in EXPRESS. The ID in the model is described by an integer, and the Age is defined by an integer in the example. The complete representation of the Base Schema is:

**Example 2.2: Base Schema, persons.exp**

```plaintext
SCHEMA persons;
ENTITY person;
    name: STRING;
    ID: INTEGER;
    Age: INTEGER;
    company: STRING;
END_ENTITY;
ENTITY family;
    Number_of_people: INTEGER;
    member: person;
END_ENTITY;
END_SCHEMA;
```

### 2.4.2 STEP P21 File

The Clear Text Encoding of the Exchange Structure as defined in part21 (ISO 10303-21) -also called STEP Physical File -is the most widely used exchange format throughout STEP [4]. The P21 file is illustrated in following Example code.

**Example 2.3: persons.p21**

```plaintext
#1= person ( 'Jen', '12345', '26') ;
#2= person ( 'Tove', '12353', '37') ;
#3= person ( 'Erik', '34345', '42') ;
#4= person ( 'Linda', '90545', '53') ;
#5= family (6, 'Jen') ;
```
2.4.3 Part 22-27 SDAI & JSDAI

SDAI stands for Standard Data Access Interface. It is an EXPRESS-driven application programming interface to perform CRUD operations on EXPRESS defined data. It provides a binding with C, C++ and, JAVA programming languages. The SDAI environment is always specific to the implementation language, but, the standards for this implementation are defined in the EXPRESS manual. A detailed discussion concerning this environment is discussed in separate topics under Java (JSDAI) and CAA (SDAI) environments.

2.4.4 Express-X

The EXPRESS-X standard has evolved over many years along with the EXPRESS standard. The objective of EXPRESS-X is to provide a mechanism capable of describing product data throughout the life cycle of a product, independent of any particular system. The EXPRESS-X language allows a user to create an alternate representation of the EXPRESS models and mappings between the EXPRESS models and other applications [9]. The articles [8] and [13] discussed two different mapping standards. 1996 EXPRESS-X standards were built on VIEW specifications whereas 1999, 2005 EXPRESS-X standards were built on both EXPRESS-V and EXPRESS-M [13], [1]. Figure 2.10 provides an overview of the EXPRESS-X standard based on VIEW specifications.

Here is a simple example to make the conception more clear and easy to understand. In EXPRESS section, we defined a base schema. Here is another EXPRESS example defining VIEW schema which has an entity company. The two versions of EXPRESS-X are discussed in this section along with mapping specification example each:

Example 2.4: View_Schema, companies.exp

```
SCHEMA companies;
ENTITY company;
  name : STRING;
  employee_name : STRING;
  employee_age : INTEGER;
END_ENTITY;
END_SCHEMA;
```
It was built on EXPRESS-V concepts and is obsolete. The algorithm for deriving the entity types in a view from the entities in an original EXPRESS model is specified using types of mapping declarations, which are defined in the EXPRESS-X language [9].

The source model is generally termed as an EXPRESS Information model whereas the target model is a VIEW Information model. In runtime, model instances are materialized in order to view instances. Different applications have their own views on a product model and an optimization for one application may not be same for another. This caused there to be a demand for an EXPRESS-X initial development for VIEW standards. These views are alternative representations of the original models. The algorithm for deriving the entity types in a view from the entities in an original EXPRESS model are specified using various types of mapping declarations[8]. The view of an EXPRESS model is created in two phases, materialize and compose. In the materialize phase, entity instances are created and simple attributes data are mapped from source entities attributes. In the compose phase, complex relations are composed by creating new instances that depend upon other view instances.

Example 2.5: VIEW Based Mapping, persons2company.exx

```plaintext
SCHEMA_MAP persons2companies;
GLOBAL
    DECLARE base INSTANCE OF persons;
    DECLARE view INSTANCE OF companies;
END_GLOBAL;
```
In the given example, the schema defines two entities person and family and the target schema defines the entity company. The mapping schema translates the data defined in the base schema data model to the target data model by defining a VIEW company by applying conditions on the mapping specification. Once the mapping has been performed, the data is translated into the target schema data model.

- **EXPRESS-X, Latest Specifications:**
  It is built on both EXPRESS-V and EXPRESS-M standards.

1. **VIEW:**
   A VIEW construct is possible together with a MAP construct. VIEW declarations describe the information from the data in a source schema to the VIEW structure. It enumerates instances using FROM, IDENTIFIED _BY_ and WHERE classes. A FROM clause is used to specify which type of source entities are being referred, IDENTIFIED _BY_ and WHERE restricts these entities based on a given condition. WHERE is a universal clause of the VIEW and it is possible to restrict the mapping defined in the view. VIEW can reference another VIEW together with source schema entities. The VIEW specifications in the latest EXPRESS-X standard are mainly used to represent alternative representations of EXPRESS-X data models rather than being used for mapping between data models.

2. **MAP:**
   The enumerations FROM, IDENTIFIED _BY_ and WHERE also apply to map declarations. The main difference between MAP and VIEW declarations is that MAP references target structures where, VIEW defines them as entity type definitions in a target schema. In MAP, multiple target structures can be referred but, VIEW can only define one target structure at a time. Map declares the entity types of target schema and it provides information as to how the individual instances being created can be identified. EXPRESS-X refers the source and target schema using USE/REF-
ERENCE keywords. It also specifies explicit binding to obtain an instance of the target entity used in an expression.

Example 2.6: MAP Based Mapping, persons2company.exx

```plaintext
SCHEMA_MAP persons2companies;
REFERENCE FROM persons AS SOURCE;
REFERENCE FROM companies AS TARGET;

MAP base2view AS
  c : company;
FROM
  p : person;
  f : family;
WHERE ((person.name = family.Member.name) And (Not (family.Number_of_people = 5)));
SELECT
  Company.Name := 'SCANIA';
  Company.employee_name := person.Name;
  Company.employee_age := person.age;
END_MAP;

END_SCHEMA_MAP;
```

Using both mapping techniques provides the same results. The source and the target STEP instances are shown below:

Example 2.7: Source STEP P21 File, persons.p21

```plaintext
#1= person ('Jen', '12345', '26');
#2= person ('Tove', '12353', '37');
#3= person ('Erik', '34345', '42');
#4= person ('Linda', '90545', '53');
#5= family (6, 'Jen');
#6= family (4, 'Linda');
#7= family (7, 'Erik');
#8= family (5, 'Julia');
```

Example 2.8: Target STEP P21 File, company.p21

```plaintext
#1= company ('SCANIA', 'Jen', '26');
#2= company ('SCANIA', 'Erik', '42');
```
2.4.5 EXPRESS-G

EXPRESS-G is the graphical representation EXPRESS lexical model. It is intended for human communication. It is used as a graphical modeling technology. EXPRESS-G is a subset of the EXPRESS language because it does not support all the features of EXPRESS and its elements compliments the EXPRESS constructs. Entities are represented by means of rectangular boxes and have attributes of interest which may be simple or complex. Attributes are connected to entities with lines and are labeled with the name of the attribute together with its cardinality constraint, if this exists. EXPRESS-G also supports supertype and subtype hierarchies similar to EXPRESS. The above EXPRESS example is illustrated in Figure 2.11 in EXPRESS-G format.

![Figure 2.11: An Example Express-G(persons)](image)

2.4.6 EXPRESS-C

EXPRESS-C is a minimal extension of EXPRESS providing an execution and behavior model. This supports the programming of EXPRESS models in lexical language. This is an upwardly compatible extension of EXPRESS that provides conceptual modeling of STEP based applications. It was developed as a part of the PISA (Platform for Information Sharing by CIME Applications) project, at the University of Karlsruhe, Germany [7]. It also provides functionality to create and manipulate the STEP data. It supports OPERATIONS (methods) to provide a descriptive specification of the behavior of entities, events and transactions for program execution and manipulating instances. It has additional functions including POPULATION, create, destroy, extend, drop and change to master over an execution model. Any compiler that understands EXPRESS, EXPRESS-C and EXPRESS-X (Based on Latest EXPRESS-X manual[1] ) can create a successful STEP implementation without the assistance of an SDAI environment and any programming language interface. EXPRESS-C assists in embedding the C++ code inside in order to enhance the execution model and the
behavior model. An example illustrating the powerful functionality of EXPRESS-C is given below:

Example 2.9: A Simple EXPRESS-C Schema

```c
SCHEMA person

BEGIN_C++
#include <stdlib.h>
Int main()
{

Class ABC

}

END_C++;
Procedure start(type: INTEGER);
BEGIN_C++
long start_type = @type@ . get_long();
start(int(start_type));
END_C++;
END_procedure;
END_SCHEMA;
```

### 2.4.7 EXPRESS-I

EXPRESS-I allows the creation of instances for EXPRESS entities that have values instead of references to data types. EXPRESS-I represents entity instances in terms of values of its attributes [9]. The EXPRESS-I language is used to display the instances of EXPRESS data elements. It is intended to generate EXPRESS-I element instances for definitions in EXPRESS schema. EXPRESS-I consists of two parts a) Part 1 for displaying data instances and b) Part 2 for specifying abstract test cases. These test cases are used to describe tests for the data model defined in the EXPRESS language. The language constructs of EXPRESS-I are test cases and context by allowing the use of procedural aspects of the EXPRESS language. EXPRESS-I is part of STEP Part 12 standard.

### 2.4.8 EXPRESS-V

EXPRESS-V is an extension of EXPRESS, developed by the Laboratory for Industrial Information Infrastructure at the Rensselaer Polytechnic Institute [10]. It specifies a simplified views for the EXPRESS data model. It maps the entities from the source
schema to one target schema. Typically a source schema is the resource in which the data resides and the target schema defines the entities to be included in a view of the source schema. The mapping results in an instantiation of the view from an instantiation of the source schema. The EXPRESS-V language uses a syntax that allows the view designer to specify the WHEN condition under which a view entity is created and how the attribute values for the new view entity are derived in the original model and the entities that have already been created in the view [9]. There can be multiple WHEN clauses in one mapping. Attributes are only assigned when the WHEN clause is true. The EXPRESS-V mappings are divided into two phases:

- **Materialize Phase:** In the materialize phase the target entity instances are created based on the mapping VIEW specification.
- **Compose Phase:** In the compose phase, the entities, whose attributes were not processed in the Materialize Phase are assigned. For example, complex entities which are impossible to instantiate in the materialize phase are composed in this phase. More than one pass may be required in the compose phase if complex, circular, or dependent attributes exist [10].

### 2.4.9 EXPRESS-M

This is similar to EXPRESS-V but mapping can be performed from one schema source to many target schemas. It has been developed by CIMIO Ltd., which has developed an EXPRESS-M compiler that can automatically generate mapping programs [7]. The language is responsible for controlling the mapping process by defining which attributes and entities of one schema match those in another [9]. EXPRESS-M is built upon EXPRESSS to maintain complete compatibility of data types and structures. It borrows data types, expressions, statements, and functions/procedures from EXPRESS language. It supports only uni-directional mapping. The source and target schemas can be included in the mapping file using USE and REFERENCE keywords. EXPRESS-M isolates the source instances and maps the entities in source instances to target instances. It contains a WHERE clause for providing conditions and prevents an automatic transformation of all possible combinations of sources. Some conditional mappings can be achieved through OPTIONAL and ONE OF operators.

### 2.5 STEP Implementation Environments

There are various software components that are either part of the existing architecture or in the proposed solution. These are the core areas of this exams work and the reader should possess a basic knowledge of these components in order to understand the following chapters. This section provides a brief description of each of such components.
2.5.1 ECCO Toolkit

In the current architecture, the JavaMigrator communicates with the ECCO using ECCO Java API to create the ENOVIA STEP file from the processed XML file. ECCO is a software component developed by a German based company PDTech Gmbh, which is an EXPRESS based integrated environment allowing for quick and flexible specification and implementation of mapping between data models [6]. It translates the EXPRESS/EXPRESS-C written specification to an executable application without the assistance of any programming language. ECCO comes with a toolkit and an environment to create STEP implementations in EXPRESS language using an EXPRESS-C extension. The ECCO Tool Kit permits the prototyping of STEP based applications that have their conceptual models specified in EXPRESS-C [17]. This toolkit performs semantic and syntactic checks for the input files and generates the C++ code for the given source EXPRESS/EXPRESS-C files. This generated code is converted at later stage to an executable and can perform fully functional STEP implementations. The components of the ECCO toolkit are designed in a modular fashion. An efficient STEP implementation can be conducted in EXPRESS by encompassing these components.

* **ECCO Editor:**
  
  This editor provides functionalities such as auto-completion, EXPRESS/EXPRESS-C keywords (syntax) highlighting in addition to the normal text editor. This also assists the user to quickly create an EXPRESS schema.

* **ECCO Compiler** (Figure 2.12):
  
  This compiler compiles the created EXPRESS, EXPRESS-X and EXPRESS-C specifications easily, accurately, and rapidly to an C++ executable or a library which can be easily linked to an existing system. It also provides the compilation for EXPRESS-C/EXPRESS-X which assists a user to build a STEP implementation written in EXPRESS language portable to the existing system.

* **ECCO Debugger:**
  
  This component assists in debugging the code by providing usual functionalities similar to other editors such as, setting and deleting breakpoints, single step or step over modes etc.[6].

* **ECCO Constraint Checker:**
  
  Analyze, check, and verify the integrity of EXPRESS-based data with respect to all constraints and rules defined in the schema[12]. It generates the reports about the data quality.

* **ECCO Data Adapter:**
The ECCO Data Adapter provides an interface between external data sources and applications built using ECCO [6]. These assists in read and write EXPRESS based data in line with ISO 10303-21. It includes multiple schema support and connects the application to different data sources.

* **ECCO Utility Libraries:**

These libraries provide high level EXPRESS functions and procedures for special requirements during the implementation of applications and, in addition, access to operating system specific functions[6]. These are implemented in EXPRESS/EXPRESS-C so that they can be included in the EXPRESS file by reducing the implementation overhead.

* **ECCO EXPRESS Engine:**

This runs the ECCO compiled program and provides an opportunity to perform CRUD operations on compiled schemas data. The algorithms written in EXPRESS-C can be invoked using this component interactively. It has a built-in garbage collector providing high performance to ECCO written application and also avoids memory leaks within the application.

* **ECCO EXPRESS-X Engine:**

It executes declarative mapping specifications based on the latest specification of EXPRESS-X [12].

* **ECCO API:**
The ECCO toolkit comes with APIs in Java, Tcl, C++, C, Visual Basic or Perl, or other programming languages to connect ECCO to an external application [12]. Thus ECCO APIs ensure a high level of portability and compatibility. Using these APIs, it is particularly easy to trigger the procedures or transactions written in EXPRESS-C from an external application.

* **ECCO GUI:**

ECCO GUI allows access to all the ECCO components by providing an intuitive graphical user interface [6].

### 2.5.2 JSDAI, Part-27

JSDAI is the proposed substitute for the ECCO and can be easily integrated into the architecture. This integration is discussed in the implementation chapter. JSDAI is a Java framework for CAX applications written for reading, writing and runtime manipulation of object oriented data defined by an EXPRESS based data model. It provides references for a) bi-directional fast access between model and entity-instance b) bi-directional fast access for explicit attribute and c) direct link from application level to meta level. It is compact and provides specialized binary format for persistent storage in a repository[3]. It provides late and early bindings to the EXPRESS for accessing entity type, attributes and aggregates. JSDAI comes with a toolkit and applications supporting STEP implementations. The JSDAI usage is illustrated in Figure 2.13.

![Figure 2.13: JSDAI Framework Usage](image-url)
JSDAI supports, in parallel APIs, a) Early binding b) SDAI dictionary c) Late binding and d) ARM-AIM mappings which are tightly related to each other[4]. The application data can be stored and exchanged in a) STEP format b) STEP XML format c) SDAI file d) local repositories and e) multi-user remote repositories.

2.5.2.1 Java Representation

The EXPRESS data model is compiled to represent Java classes so that entities in the data model are accessed similar to Java classes. The majority of the ISO standard data models are available in the JSDAI core library and custom data models are compiled to Java libraries and can be made available to applications similar to standard models. To use standard models in an application it is necessary to import the Java package of the particular standard (say AP203) in the API. A custom data model written in EXPRESS compiled to Java library is illustrated in Figure 2.14. This custom data model once compiled to Java libraries has same properties those exist in the framework and can be used into the application by merely importing the generated Java package (actually a jar is produced, referencing this jar in build-path gives access to the generated package) into the application.

Figure 2.14: JSDAI Compiler: Input and output files (java libraries) [3]
In early binding, entity data types are represented by Java interfaces by means of the prefix "E", complex entity data types are represented by the prefix "C", multiple leave entity data types are separated by means of an "&" and finally and finally aggregates of entity data types are prefixed by A[3]. In late binding, all the entity data types and complex data types are represented by predefined Java Interface EEntity and aggregate entity data are represented by "AEntity"[3]. The trick behind this implementation is, either directly or indirectly, the early binding entity data types are extended interfaces of "EEntity" and aggregate entity data types are extended interfaces of "AEntity".

2.5.2.2 Management Classes and Auxiliary Objects

A STEP implementation performed using JSDAI uses various management classes to perform CRUD operations on data. SdaiSession is a root management class from which every SDAI activity is triggered. It is used to write out warnings, debug message and measure times etc. The SdaiRepository management class is used to create a physical storage space for SchemaInstances and SdaiModels. The SchemaInstance management class logically groups the models within repository to a defined valid population of an express schema. SdaiModel class groups the entity instances based on one express schema. The SdaiTransaction controls simultaneous updating of changes made by multiple clients at an instant and exists on various levels of transactions. "Level1" is considered as no transaction,"Level2" is considered as a SdaiModel based transaction and "Level3" is a full JSDAI transaction. The SdaiImplementation object provides the runtime information of the implementation by providing information in relation to the name, version and features of the implementation and can be accessed through the SdaiSession. EntityExtent is an auxiliary and provides easy access to entity instances and a SdaiException is raised when an error is encountered.

2.5.2.3 JSDAI Components

JSDAI toolkit is a set of four components and can be used to develop customized STEP implementations but not all components are compulsory to develop a STEP implementation. These components are discussed in this section.

i) JSDAI Core:

This core framework should be compulsorily included in the STEP implementation project. It consists of a JSDAI runtime platform accompanied by technologies and tools for the development of application using JSDAI with support for early and late bindings of EXPRESS schemas [5]. JSDAI core has following main parts:

- JSDAI Runtime & Schema Library:
  It is an API to work on STEP data and can be considered as being the heart of the JSDAI and works in collaboration with the JSDAI EXPRESS com-
piler results and it is capable of dealing with any kind of data model defined in EXPRESS modeling language. It implements the standards defined in ISO 10303-22, which is a part of the implementation methods of STEP and known as SDAI. The important standards that are defined in SDAI are SDAI dictionary schema (implemented as SDAI_dictionary_schema in JSDAI), SDAI session schema (implemented as SDAI_session_schema in JSDAI) and SDAI operations providing various extensions for P21 files, network access, mapping operations, event support and many. Application schemas (data models written in EXPRESS language) are converted by the JSDAI compiler to a) a population of dictionary schema for late binding access which assists in the working at the runtime in JSDAI and b) Java package to access the Java classes/interfaces which are mapped to their entities in the data model internally.

For example, SDAI_session_schema is converted to jsdai.lang package and the purpose of this package is to manage populations of STEP data by providing late and early binding entities and attribute access simultaneously. This package contains management classes and auxiliary objects previously discussed 2.5.2.2.

The SDAI_dictionary_schema is converted to jsdai.dictionary package and it contains the meta information of EXPRESS schemas. This information is only required by late binding applications. The main dictionaries inside this package are a) schema_definition b) entity_definition c) attribute d) defined_type e) where_rule f) global_rule and etc.

♦ JSDAI EXPRESS Compiler:

It is used to parse one or several EXPRESS written data models, generate necessary dictionary information and Java classes/interfaces and makes them available to JSDAI [5]. Its main feature is the ability to operate on EXPRESS short form schemas and these are internally mapped to original schema names. The naming styles for these generated classes/interfaces were discussed in 2.5.2.1.

♦ SDAI File:

This is a binary encoded open-documented format file similar to those in STEP P21 and STEP-XML. It is created with features for the exchange of EXPRESS defined data and is optimized for writing and reading speed and can include external documents. It is ideally suited for the exchange and backup of the entire SdaiRepository of any size [5]. It is built in zip compression. It includes the meta information of EXPRESS schemas in order to convert the data in the repository to STEP P21 or STEPXML format.

ii ) JSDAI for Eclipse:

JSDAI has an Eclipse IDE (Java based IDE which is widely used for developing Java based applications) a plug-in providing a graphical environment to a user
for creating, editing and compiling express files. It also provides a project environment in this IDE making an JSDAI based application development easier. Creating, editing and compiling, keywords highlighting, syntax and semantic checking EXPRESS schema are the key features of EXPRESS editor (Figure 2.15).

Using a JSDAI project window, JSDAI applications can be developed similar to those of other Java applications, eliminating build-path setting overhead (JSDAI applications also developed in Eclipse without this plugin but needs JSDAI core API in its project build-path). It also has an EXPRESS-G editor (Figure 2.16) which assists in converting the EXPRESS defined schema to an EXPRESS-G graphical representation together with the relationships between the entities defined in schema.

### iii ) XIM Library & JSDAI Database

These are further JSDAI further extensions which enhance the JSDAI features and making a toolkit which is a highly evolved resource for STEP implementations. The explanation for these concepts is beyond the scope of this thesis as they are neither included in the work nor tested as part of this work.

### 2.5.3 CAA/C++

Dassault Systemes developed an API built upon C++ language features to customize the ENOVIA/CATIA systems. The core components of the ENOVIA/CATIA are delivered to their customers along with a CAA API for customizing these components
to their specific requirements. This framework works on the principles of an "Object Model" and the inspiration for this is based on the Microsoft Component Object Model (COM). Dassault Systemes adapted such a technology because of the versatility of the demands of their customers. Each individual customer has his/her own specifications and it is impossible to realize an end product which is suitable for all customers. Thus they sell the core components of their products to their customers and provide an API so that these components can be customized. This API is robust, flexible and scalable in terms of the customization. Before discussing the SDAI features in CAA, it would advantageous if the fundamentals of this software architecture were understood. The main intention of this architecture is to develop reusable components and a further development or customization is thus possible from these components. These reusable components are compared to building bricks which are used to construct any kind of structure. These components are granular in structure and are meant for a single targeted purpose. By integrating such individual parts together it is possible to achieve a new component which has a different behavior. The basic components of this architecture (defined in CAA perspective) are:

♠ **Component/Module:**

It should be a reusable unit of deployment and composition [14]. It should exist irrespective of the context. Adding a component should provide some additional functionality to the context and this service should be part of this component. Components can be a group of objects or even more. These are more granular in nature than objects. The design of a component should be conducted in such a way that the implementation is separated from its interface. The definitions will be in the interface whereas these definitions are implemented in its respective
class. The service is accessed by means of the interface definitions. These inner objects are accessed via the exposed interfaces of the component and other means are strictly restricted. It should satisfy the re-usable principle and can with work any system with which it is integrated. A component in CAA is termed as a module. These modules exist as either SHARED or LOAD. A SHARED module is a component that cannot execute on its own where as a LOAD module has a defined entry point. Each component in the CAA has an "Imakefile" file to specify its linked components, target operating systems and program type.

♠ Interface:

The interface is the only access point to the component. Depending upon the number of services the component has, it should have respective interfaces to expose these services to the outer world. The important dependencies of these services are described in the interface. This should define a protocol to communicate with the component for delivering the service. The interface should describe the usage of the component. Interface Definition Language (IDL) is used to describe the interfaces. There are different Interfaces found in a framework.

1. **PublicInterfaces**: contains the interfaces that are available for frameworks of other modelers.

2. **ProtectedInterfaces**: contains the interfaces that are available for the frameworks of the modeler to which this framework belongs.

3. **PrivateInterfaces**: contains interfaces that the framework wants to provide but for its own needs. These interfaces are not visible outside the framework scope.

♠ Framework:

A framework is component infrastructure which ties a group of components under one roof. This is the context under which these components are assembled. The components must fulfill the contract in the framework and the framework is a skeleton of the application.

♠ Identity Card:

It is considered as an interface to the framework. It contains the list of other frameworks whose components are referenced by components in a framework.

♠ Dictionary File:

It defines the dictionary dependencies of the framework. There might some modules within the framework depending upon some predefined dictionaries and dependency entries should be entered into the dictionary file of the framework in order to have full functionality.
2.5.3.1 CAA RADE

CAA RADE is the Rapid Application Development Environment plugin to Microsoft Visual Studio IDE from Dassault Systemes to ease the ENOVIA/CATIA customizations. Once installed together with the Microsoft Visual Studio, the CAA environment is plugged to the IDE. This assists developers to develop ENOVIA/CATIA customizations rapidly.

2.5.3.2 CAA SDAI

The ENOVIA vault has computer design parts relating to a product in which the geometrical structures of these parts are located in SPECTRA (only a SCANIA Solution). When the breakdown is ordered, this product structure should be imported to ENOVIA and made available to the CATIA environment. This import is called an ENOVIA import and Dassault Systemes provided a pre-defined "Korn-Shell" script for this process. The ENOVIA server expects the product structure in the form of a P21 STEP file. This imported STEP file is checked or merged with pre-defined conditions. The CAA framework has various components that can assist in customizing these checkers and mergers together with the Dassault Systemes pre-defined checkers and mergers. These components have a built-in customized SDAI environment. The important frameworks and modules that provide SDAI environment in CAA are:

* SDMRuntime:
  This framework enhances the CAA compiler (mkmk) to compile the EXPRESS data model to a CAA C++ module/Component. This generated CAA module has similar properties to those of a normal CAA module and can be considered as a pure CAA component. **KS0LATE, KS0SIMPL** are the modules inside this framework that enhance the functionality of the CAA compiler. Including them into the compiled module **Imakefile** converts the EXPRESS data model to a fully functional CAA module. This framework also provides access to entities, attributes in the EXPRESS data model in CAA runtime and it performs automatic SDAI session handling.

* SDMExpressX:
  This framework converts the EXPRESS-X specification to a CAA module by compiling it with a CAA compiler. The **KS0EXXXMAP** module inside this framework in addition to the SDMRuntime modules **KS0LATE, KS0SIMPL** enhances the CAA compiler to compile EXPRESS-X specifications. In a similar manner to that of the EXPRESS compilation, these modules must be added to the **Imakefile** of the EXPRESS-X compiled module.

There are a few limitations in the usage of EXPRESS/EXPRESS-X compiled CAA modules. Dassault Systemes provide various interfaces to create a SDAI model and without using these interfaces, it is not possible to create a fully customized CAA module for ENOVIA. After compiling the EXPRESS-X mapping
file, an interface should exist in addition to its implementation and this exposes this mapping service to the ENOVIA import Korn-Shell script. To create such an interface an ENOVStdCmpt module is required from the VPMInterfaces framework and GUIDVPMSTEPExchanges from the VPMSTEPExchanges framework. This interface (actually implementation class) exposes the mapping functionality and enables it to convert the data from the source container (SDAI model will be in this container) to target container (target SDAI module). EXPRESS-X mapping file defines the source and target data models and has declarative mapping between the source entities to the target entities.

* ObjectModelerSDM:

This framework should also be used because it provides the basic SDAI services such as creating and deleting in relation to a SDAI model, converting the SDAI model to a CAA specific generic container (termed as CATIContainer) in which the ENOVIA customized modules are able to easily recognize the data in it. CATSDM_Services Services is one such interface that provides these services. Along with these frameworks, exist many other frameworks that are to be included so as to develop a STEP implementation using CAA/C++ API. However the above mentioned frameworks are considered as being the heart of the whole implementation and without these modules the CAA compiler will not recognize the EXPRESS data model and EXPRESS-X mapping file.

2.5.3.3 CAA EXPRESS-X Usage & USER FUNCTIONS

CAx applications have to work around the STEP implementations. CATIA is a CAx platform consisting of an user interface for each PLM stage. These user interfaces work with the same data in the ENOVIA vault but, have their own VIEW while interacting with data. For example, a CAD interface creates a complete 3D model data based on the product structure. However, this data can be viewed by other CAx interfaces such as Graphic Design or Mechanical Engineering from another perspective. To have these different perspectives, these applications have their own defined EXPRESS VIEW. To communicate with these perspectives among CAx applications, DASSAULT SYSTEMES enhanced the CAA features by including a SDAI environment. This environment enhances the CAA compiler to compile EXPRESS or EXPRESS-X specifications to CAA modules. This compiler was customized to the old standards for EXPRESS-X and defines the mapping specifications based on VIEW standards. Please refer to Section 2.4.4 for complete information with regards to this standard. VIEW is a typical straightforward declarative approach to define mapping and it is impossible to achieve complex operations in this standard. So to enhance its capabilities DASSAULT SYSTEMES introduced a concept called USER FUNCTIONS.

- EXPRESS-X
The EXPRESS-X standard in CAA is based on EXPRESS-X, 1994 standards. The compiler was designed to compile the EXPRESS-X specifications. This feature is used to exchange the information between different CAx applications. For example, considering the EXPRESS examples discussed in Section 2.4.4, a simple express mapping appears as follows:

**Example 2.10: VIEW Based Mapping, persons2company.exx**

```plaintext
SCHEMA_MAP persons2companies;
GLOBAL
    DECLARE base INSTANCE OF persons;
    DECLARE view INSTANCE OF companies;
END_GLOBAL;

VIEW company;
FROM (person, family)
WHEN ((person.name = family.Member.name)
    And (Not (family.Number_of_people = 5)));
BEGIN_VIEW
    company.Name := ToUpper('scania');
    company.employee_name := person.Name;
    company.employee_age := person.age;
END_VIEW;
END_SCHEMA_MAP;
```

The above specifications are converted to a CAA module by the EXPRESS-X compiler. However, in order to recognize the source and target schemas, the workspace should consist of `persons` and `companies` EXPRESS compiled CAA modules. This EXPRESS-X module should also consist of CAA `Mapper` class. The basic mapping standard was implemented in an API named `StdMapper` and this API is extended by a Mapper class which enhances this CAA module to link with EXPRESS-X specifications through the SDAI environment. A typical mapper class has the following appearance:

**Example 2.11: Mapper Class, Mapper.cpp**

```plaintext
#include "Mapper.h"
#include "UserFncts.h"

CATImplementClass(Mapper, DataExtension, StdMapper, persons2companies);
#include "TIE_VPMIExMapper.h"
TIEchain_VPMIExMapper(Mapper);
```
2. Theory

In the above example, the CATImplementClass defines that the current class is an extended version of StdMapper in the CAA module persons2companies. Convert is the exposed API of the CAA:StdMapper to map the data between different data modules. CATIContainer is used to represent the SDAI model in CATIA generic format where as _var is the smart pointer of CATIContainer. This class also includes the defined “User Functions” module interface so that user functions defined in this module can be called in EXPRESS-X specifications. ToUpper is a USER FUNCTION defined in the UserFncts module directly called from inside the EXPRESS-X environment, which is linked to EXPRESS-X CAA generated code at compile time.

- USER FUNCTIONS

Typically USER FUNCTIONS are used to perform some complex operations which cannot be performed in the EXPRESS-X environment. DASSAULT SYSTEMES provided two formats to define a USER FUNCTION, namely a single parameterized or double parameterized. Both formats take only SdaiPrimitiveH as the input parameters and return SdaiPrimitiveH as the output. Generally, any CAA SDAI data type can be represented in SdaiPrimitiveH because all these data types are inherited from this single data type. A simple USER FUNCTION that was called in the above EXPRESS-X specification has the following appearance:

Example 2.12: User Function, ToUpper.cpp
The above user function is part of the UserFncts module and exposed to other modules by means of the module interface UserFncts.h. This module should be included in the makefile of the EXPRESS-X module in order to access this user function in EXPRESS-X specification. The module interface UserFncts.h should export this user function so as to provide it with global access. For Example, UserFncts.h can have the following appearance:

Example 2.13: Module Interface Exporting User Function "ToUpper"

```c
#ifndef UserFncts_H
#define UserFncts_H

#include "SdaiPrimitiveH.h"

#if defined WINDOWS_SOURCE
#define _UserFncts
#else
#define ExportedByUserFncts __declspec(dllexport)
#endif

#define ExportedByUserFncts __declspec(dllimport)
#endif

ExportedByUserFncts
SdaiPrimitiveH ToUpper(const SdaiPrimitiveH& string);
```
3 METHODOLOGY

This chapter discusses the strategic approach followed so as to design a solution for the defined problem. It discusses the design flow and methodology used in each design step.

3.1 Design Flow

Figure 3.1 shows the design flow from the system level to the lower level. The solution mainly follows the design flow and methodology as provided below:

The complete design flow is divided into two parts a) system level and b) lower level.

3.1.1 System Level

This is the higher abstract level of the problem and focuses on a solution without entering into internal design details (the system is looked at as a black box). The design goal in this level is to optimize the system in order for it to run more efficiently in the overall architecture. Methodologies are followed to solve the problems in this level. The design starts with the informal specification constraints which are very general and informal regarding the project. These informal specifications act as inputs for the generation of the formal functional specifications of the model. The model is designed based on modeling concepts. The model can be based on a programming perspective or UML perspective. Given the specifications to generate a CAA module, in this case the model is CAA environment based but a Java model is also proposed because of specific limitations which will be discussed in the next chapter.

In the system model, the primary task is to decide upon the architecture. The task is to replace a component that currently exists within the high level architecture in the SCANIA environment. Hence, in this level there is no requirement to design a complete system level model but to use the current model in order to derive the new one. The desired goal was that the designed component was to integrate with the other components in the architecture. The model architecture had different design proposals which will be discussed in the next chapter. To make an appropriate decision based on the different solutions, consideration was given to various estimations. For each so-
Figure 3.1: System Design Flow
olution, advantages, disadvantages, obstacles and limitations existed. In Figure 3.1 the loop that covers the architecture selection, estimation, mapping and scheduling work is an iterative approach which is adopted in order to achieve the optimal solution. If any problem is faced in the implementation stage, a new design has to be proposed which has to consider and satisfy the termed conditions including the architecture selection, estimation, and mapping, time consuming and cost limitations. SCANIA’s current architecture optimization work never stops. New technology, new methodologies, new software, new hardware platform will always have an impact on the current architecture which enables its product quality and work efficiency to be improved.

3.1.2 Lower Level

In this level programming platform, selections and decisions regarding data ow, software model, software API, and the hardware platform will be made. The implementation of the software should co-ordinate with the chosen hardware. The simulation should be synchronized between these platforms. In this project, the final product should accept the conditions of an IBM AIX machine in both the production and development servers in SCANIA but, the initial development can only be performed on any CAA installed system. The RADE plugin to Visual Studio takes care of this by delivering a platform independent product from a code written in this IDE. Generally for development, a desktop or a laptop installed with the CAA specific software and Microsoft Visual Studio 2005 would be to sufficient in order to implement the design. In addition to the CAA, Java runtime and JSDAI and ECLIPSE tools would be necessary in order to implement the alternate proposal. The generation of the software product for the proposed solution is divided into different stages which can be seen in the Low Level Design in Figure 3.1.

1. Implementation of design using programming languages such as Java, CAA/C++, EXPRESS, EXPRESS-X

2. Debugging the code and correcting the compile time and runtime errors. ECLIPSE IDE supports Java code syntax checking and highlights the syntax errors whereas Visual Studio is used to correct the C++ written code syntax errors. The code should be compiled once and it must be checked for grammatical errors.

3. Performing a platform acceptance test on the written modules and optimizing the available environments on the production and development servers. Data consistency should be considered and fulfilled.

4. Performing other tests by writing simple test cases.

At the end of low level design, a successful prototype is generated which matches the systems specifications. During the intermediate stages of this designing process if any problem or obstacle arises, the solution should be estimated again and the work should
be re-initiated from the beginning in order to determine an optimal solution that over-comes the encountered obstacles. The solution design should ow from architectural level to the system level.
4 Implementation & Results

This chapter discusses the different solutions proposed and their implementation details. In order to be able to determine the best solution for the task, an analysis of the proposed solution has been conducted which considers factors such as the system acceptance, advantages, disadvantages, requirements, obstacles or limitations in relation to the implementation. The optimal design is selected based on the estimated results of these factors. To satisfy the problem statement, each proposed solution follows the system level design in Figure 3.1 satisfying the architecture selection, estimation, mapping, scheduling work in a iterative loop until the optimal decision has been taken. To maintain equilibrium in the workflow it has been assumed that each solution consists of two modules. The first module processes the XML file using the available XML API parsers and generates the intermediate SCANIA STEP file using the available SDAI environment. The second module takes the intermediate result as its input and respective resources in order to generate the target ENOVIA format. Based on the available resources and the experienced implementation factors, three solutions have been proposed in different stages: a) CAA+CAA, b) JAVA+CAA using IEnovIn script and c) CAA+CAA using IEnovIn script. The necessities for each design and their implementation factors are explained in further sections. Due to SCANIA secure policies, the results were not included in this report.

4.1 CAA + CAA

The vision of this project is to design a solution in a CAA/C++ API environment. Based on limited knowledge and as it involved an area in the SDAI environment which had not been explored in this API, a model was proposed. According to the API documentation, the implementation of this model is possible. Figure 4.1 consists of two CAA modules internally, in which the first module is expected to deliver the SCANIA format intermediate STEP file and the second module takes this result as its input and delivers the target ENOVIA format. At a later stage the target file is imported into the ENOVIA server using the Korn-Shell written script which is executed by the Control-M. After generating the target file, the job is handed over to the Control-M by the second module. In the design phase, this appeared to be promising but during the implementation obstacles were encountered and the final result was that the implementation had been too ambitious. This section discusses the implementation
details together with the obstacles faced for this model.

4.1.1 Implementation

As discussed previously, the implementation is divided into two parts. In the first part, SPECTRA XML data is considered as the input source with SCANIA intermediate data as the output. XML handles the complex relations describing the breakdown and user information. For example, the source XML file having similar relationships is illustrated in 4.2.

![Image of XML Data with complex relationships]

Figure 4.2: XML Data with complex relationships

The second part was supposed to load the intermediate SCANIA STEP data and, using the EXPRESS-X mapping specification, the final result ENOV DATAX has to be delivered. This result is also a STEP file but the ENOVIA server understands this
format and imports into the database. The import is performed using the DASSAULT SYSTEMES provided IEnovIn.sh script.

4.1.1.1 CAA HASHTABLE Limitation

In the first module of this proposed solution, the XML file which containing the complex assembly relations within the generic format have to be processed to a SCANIA format. To achieve this goal, an API with strong XML parsers together with the finest collections are required. CAA API has the necessary parser but failed to provide the Java HASHTABLE similar collection. HashTable functionality is necessary in order to process the XML nodes to discrete objects which are later converted to the SCANIA format. Without having this collection, it is impossible to achieve the target file and eventually the implementation of this module became impossible with the existing resources. Figure 4.3 represents the implementation failure of this model because of failures in the XML processing. This led to a re-think for a new proposal.

![Figure 4.3: Problem Generating the Intermediate STEP file](image)

4.1.1.2 No STEP import Support

It also required the importing of the STEP data into the CAA SDAI environment for further processing. But there is no exposed CAA API that takes the STEP file path to import the data into its SDAI model. After making several attempts, a service request was retained for the DASSAULT SYSTEMES as the possible solution. However, the DASSAULT SYSTEMES support team eventually confirmed that it is impossible to achieve this with the currently available API. This was the second obstacle to this model and thus caused further progress to fail with regards to the implementation. Figure 4.4 shows the SDAI environment indicating the obstacle in implementing the design.
4. Implementation & Results

4.1.1.3 Provision Document

In the ECCO specific mapping specifications, ECCO libraries exist that were used to create the provision document in an EXPRESS-C manner. However, there is no such strong functionality provided in the CAA specific EXPRESS-X standard and eventually this also became an obstacle in the implementation. To create the provision document, a SCANIA intermediate format has to be processed in depth-first and breadth-first traversals. Usually, EXPRESS queries are used to process this step but CAA EXPRESS-X specifications has limitations in using queries. ECCO uses the latest EXPRESS and EXPRESS-X standards and can be customized to realize this complex job in an EXPRESS environment.

4.1.1.4 Limited EXPRESS-X

DASSAULT SYSTEMES customized the EXPRESS-X 1994 standards to provide the data mapping API with CAA. However, the EXPRESS-X mapping standards are based on VIEW specifications with outmatching functionality when compared with ECCO EXPRESS-X standards. It was impossible to adapt the current ECCO standard mapping specification into a CAA EXPRESS-X standard. The real mapping between the SCANIA and ENOVIA EXPERESS appears similar to that in Figure 4.5. This was the major obstacle in the second part.

4.1.1.5 No CAA Execution For Mapping

It was assumed that the mapping module can be called internally by another executable and the data translation processed between the two modules. However, API errors exist which are able to take this control internally by means of a CAA executable.
Figure 4.5: Complex Mapping between SCANIA Schema and ENOVIA Schema
This was also reported to DASSAULT SYSTEMES and it is under investigation as an API bug. Eventually, this also made the implementation of this proposal impossible.

After considering the above limitations and obstacles, a proposal for a new CAA independent solution was required. It demanded that some other platform be chosen which had a similar API to that of CAA SDAI so as to interact with EXPRESS data models. Java is chosen for two reasons. It can be easily integrated into the system architecture by plugging into JavaMigrator and an Open Source API named JSDAI from LKSoft exists which has strong components such as those for a SDAI environment and EXPRESS compilers. This eventually led to proposals for a second solution.

4.2 JAVA + CAA

In the previous proposal, the target file ENOVIA format had to be generated from a SCANIA intermediate file. A SCANIA intermediate file is very similar to a SPECTRA format and has generic relationships between the entities of a SCANIA model schema. These led to complications in relation to the EXPRESS-X mapping file with complex mapping specifications. To realize such complex specifications in a CAA specific EXPRESS-X standard would be impossible. Mapping specifications in ECCO are coupled with EXPRESS-C and EXPRESS-X which made it difficult to realize the current specifications in the latest EXPRESS-X standards. This limitation led to the designing of a new EXPRESS data model for the SCANIA format. It is designed based on general PDM schema specifications. The complex assembly related conversions from SPECTRA data to the new SCANIA EXPRESS model are performed in a JSDAI environment which eventually provided a successful module in JAVA by leveraging JSDAI support. Trials were also conducted to design a second module in JAVA using JSDAI EXPRESS-X support but this eventually became a failed attempt because of limited EXPRESS-X standard implementation within it. This led to the choice of a second module in CAA but it required the assistance of the IEnovIn input script to import STEP data into SDAI environment in CAA. This is the import tool from DS to import the STEP data into the ENOVIA vault. In the documentation, it describes the usage of the IEnovIn script in two ways. A direct ENOVIA format can be imported to the ENOVIA vault or any STEP data together with a mapping module between the STEP data original schema and it is also possible to use the ENOVIA schema to import non-ENOVIA standard data into its vault. However, the second approach works by converting the non-ENOVIA standard to an ENOVIA format internally by using the mapping module and the final version is used at a later stage for the real import. This two-step process helped to design a new solution by overcoming the obstacles which had been faced. However, this solution also resulted in solvable obstacles. This model is illustrated in Figure 4.6.

This module has two parts. The first part deals with the conversion of SPECTRA XML data into an intermediate SCANIA format. But here, a new EXPRESS schema has been proposed for the SCANIA format instead of using the old one. In order to
4. Implementation & Results

May 14, 2012

4. Implementation & Results

Figure 4.6: JSDAI and CAA Based Proposal

efficiently interact with XML data in Java, simple Java Classes were written. Instead of the old SCANIA schema, a new schema has been proposed for the intermediate result. This result has target complex relations. This was proposed to simplify the target EXPRESS-X mapping specification. There have been a few changes made to the first model in order to realize this model.

4.2.1 Java Representation

To process the XML file in the existing manner, the old SCANIA EXPRESS schema is represented in the pure JAVA format. JAVA classes were written and used instead of the old EXPRESS schema. Thus instead of the SDAI Model, Java collections such as ARRAYLIST, HASHMAP and HASHTABLE are used to process the XML file in order to filter the related data. Catia_Object, Catia_Structure and Catia_Structure_Row were created for ease of interaction in the Java environment. In Figure 4.7, the class diagram has a complete list of the Java classes. SPECTRA XML data has data with complex relations. Java HASHTABLE is used to process this XML data to separate the data into Catia_Object list and relations between these objects are maintained using Catia_Structure and Catia_Structure_Row lists. Instead of using an SDAI environment, it now became possible to interact the processed XML data in a pure Java environment.

4.2.2 NEW SCANIA Schema

JSDAI from LKSoft is used for the SDAI environment in JAVA and also for creating a SCA-NIA intermediate format. This toolkit has JSDAI API in addition to an EXPRESS compiler. To map the Java objects to the target ENOVIA format, a source EX-PRESS model must exist. Thus, a new SCANIA EXPRESS schema is proposed
Figure 4.7: JavaModule Class Diagrams
and suggested to be used instead of the old schema. This new SCANIA format has all the complex relations of the ENOVIA format but represented in a organized fashion. This allowed for a possible approach in order to realize the EXPRESS-C standards using JAVA and JSDAI. This solution also has the scope to use the strong functionality of JAVA to make some complex decisions in the SDAI environment. An EXPRESS compiler is used to compile the new SCANIA schema to Java libraries. These libraries are included in the implementation in order to create intermediate SCANIA STEP data using an available SDAI environment. The Java Objects created from the XML data are processed further in order to create this STEP data. In Figure 4.7, XML2SCANIACOnverter performs all the JSDAI interactions. Due to SCANIA secure policies, no illustration is included for the new SCANIA schema.

4.2.3 Simplified EXPRESS-X

The complex source format a similar to the target format but with a simplified mapping specifications between the two models. It has been constructed so as to realize the new mapping specifications either in the new EXPRESS-X standards or the old EXPRESS-X standards. A successful CAA module has been created for this mapping specification and a CAA compiler has been successfully compiled for the written specifications and has been created a runtime view that promising in relation to translating the data between the two models. In Figure 4.7, SC2EN is the successful CAA mapping module between the SCANIA intermediate format and the ENOVIA format. It has EXPRESS-X specification in a simple declarative style.

4.2.4 Provision Document

The provision document is generated in the JSDAI environment instead of the EXPRESS environment. Using a Java IO package and using processed Java objects the provision document is created. The Java Objects which have the user information and the breakdown ordered parts information are processed for the creation of the provision document.

4.2.5 Implementation

After performing the above discussed changes, a successful JAVA module has been written which fulfills the first part of the whole module. The vision of this model is to provide the generated STEP P21 file in this part as the input to the second part, where it imports the STEP data into the ENOVIA vault. The implementation of this model is explained using two sequence diagrams. The first sequence diagram describes the conversion of XML data to intermediate SCANIA data. The second diagram assumes that it was triggered by the JavaMigrator in order to perform the import process. This model has two parts. The Java part creates the of intermediate SCANIA STEP data along with the provision document from the XML data. The CAA part performs the
conversion of SCANIA STEP data to ENOVIA STEP data and performs the import using an IEnovIn script file.

4.2.5.1 Java Part

When the JavaMigrator finds the XML file on the SIAM IP servers, it retrieves it in order to process it to a simple string and this will be forwarded to its listener named SpectraBreakdownCarinaVersion1. This contains two methods takeAction and takeActionProcessing. takeAction is responsible for creating the SCANIA intermediate format from XML data. This can be seen in Figure 4.8. takeActionProcessing interacts with control-M and creates the job using Korn-Shell script for the import process. The method is responsible for the execution of the second part. This is not visualized in any sequence diagram but has to be assumed for the second part.

- takeAction
  This method once triggered, creates the job directory in the spool area and copies the contract id of XML data into this directory. As a further step, it creates the instance of GeoXMLParser(Please refer to the Class Diagram in Figure 4.7), where the real processing of the XML data will be performed. SpectraBreakdownCarinaVersion1 is the master of this class and directs it by a controlled sequence in the following steps.

  * readXMLtoDOM
    Once the GeoXMLParser instance is created, SpectraBreakdownCarinaVersion1 requests GeoXMLParser to process the XML data into Java DomObject using Java API DomParser.
  * CreateCustomObjects
    This method conducts the real processing of DomObject to Java Objects. It recursively enters into the XML structure and builds a HASHTABLE from the XML data. Once HASHTABLE has been created, it traverses the HASHTABLE and creates the CATIA_OBJECT list.
  * CreateCustomObjectReferences
    This method performs the processing of DomObject to Java reference objects. It recursively enters into the XML structure and builds the references for the created Java Catia_Object in the list. Creates Catia_Structures and Catia_Structure_ROWS using the references.
  * initializeMapEnvironment
    This initializes the environment for creating the SCANIA intermediate STEP data along with the provision document. The real interaction for STEP and PROVISON creation is performed in the XML2SCANIAConverter. So, SpectraBreakdownCarinaVersion1 requests the GeoXMLParser using the initializeMapEnvironment and this creates the environment.
Figure 4.8: Java Part Sequence Diagram
setup using `initializeMap` method of `XML2SCANIAConverter`. This initializes the scope of the `XML2SCANIAConverter` globally inside the `GeoXMLParser`.

* `writeProvisionDocument`

Provision Document is created inside the scope of `ProvisionContext` class. To create this document, the Catia Object list has to be initialized with XML data. This initialization is performed by this method. Later, inside the scope of this method `writeProvisionFile` method is triggered to create the Provision Document.

* `performXMLtoStep`

STEP creation scope is inside the `XML2ScaniaConvertor` with in the `beginMapping` method. This class is inherited from `ProvisionContext` so as to access all the initialized values while creating the Provision Document. For Example, Provision values are used in creating the STEP data. `beginMapping` implicitly interacts with JSDAI to open an SDAI session, starting the transaction, creating the repository, populating the SDAI model, saving a SDAI model to a STEP P21 file and finally ending the transaction by deleting the repository and closing the SDAI session. A transaction in JSDAI is created before creating the repository and can be done after opening the JSDAI session. Once the STEP file is created, session should be closed to commit the changes.

On a successful result,

- `takeActionProcessing`

  This creates the job in Control-M and allocates the resource in it. The variables in the Korn-Shell script are initialized and the execution of this script will be handed to Control-M. Control-M executes in a priority fashion and performs the import process.

### 4.2.5.2 CAA Part

This has the complete workspace that is required for the import process. The workspace should consist of a new SCANDIA EXPRESS schema CAA module, ENOV_DATAX EXPRESS schema CAA module, SC2EN EXPRESS-X specification CAA module along with the mapper class. The import process is illustrated in Figure 4.9. This CAA compiled runtime is given as the input to the IEnovIn import script along with the SCANDIA custom template named `inputTemplate.xml` which consists of checkers and mergers for the ENOV_DATAX format, policy script file and SCANIA intermediate step file. The import parameters of the IEnovIn script are assigned with these above values indicating that the mapping should be performed before the import process. IEnovIn script has internally called the DASSAULT SYSTEMES API VPMimportCDM to perform the import process. To trace the values or errors, the `.optionsfile`
has to be initialized. Once, the environment is initialized, the mapping process will be executed triggering the SC2EN module. This model is executed in two step process. In the first STEP, the data translation is performed using the mapping module SC2EN as shown in the sequence diagram (Figure 4.10) and continues with a second step involving the importing the data into the ENOVIA vault.
Figure 4.10: CAA Part Sequence Diagram Using IEnovIn Script
Even though the documentation supports the import of Non-ENOVIA standard STEP file which is SCANIA STEP data using a mapping module, the real implementation faced obstacles which had to be overcome in order to achieve a successful module. The obstacles faced in this stage were:

- **Failed IEnovIn**

  This is the only obstacle in the model. According to the documentation, any format STEP import can be performed into the ENOVIA vault using the IEnovIn the script file but script expects a valid mapping specification between the STEP data original EX-PRESS schema and the target ENOVIA model schema. The STEP import conducted in this manner performs the import in a two-step process. Initially it maps the non-ENOVIA standard to an ENOVIA format and creates a STEP file and in the second step it performs the real import of ENOVIA data into the ENOVIA vault. However, this method fails in the implementation stage as it resulted in execution errors although it had a successfully compiled CAA module. The IEnovIn script failed (indicated in Figure 4.11) to recognize the source and target entities for the execution of the mapping. This issue was retained as an service request to the DASSAULT SYSTEMES support team and is currently under investigation.

![Figure 4.11: Error Indicating the Failure in IEnovIn script](image-url)
4.2.6 System Evaluation

In the JAVA+CAA module, JAVA is a widely used programming language for software developers, and is a general and standard module which provides a normal developer with a good platform for obtaining knowledge in relation to specific data modeling for those who do not possess the relevant knowledge about product data modeling language. In the first JAVA module part, an XML file with around 5 MB size could be processed in about 7 seconds and a correct intermediate step file obtained. The JAVA module involves the most complicated job in the solution, it first structures the relationships between different objects in the XML into a hierarchy, and then handles the complicated relationships between two schemas. The execution time of the JAVA module is similar to the existing ECCO component, which converts the XML file into the intermediate result, but in a simple manner as it does not handle the complicated relationships between different entities. So from this side, the performance of the JAVA module is better than the existing ECCO component. In addition, the JAVA module could be customized easily based on different system requirements. For example, when a customer changed the schemas, it was possible for a developer without background knowledge of data modeling to modify the JAVA code, which simplified the specific requirements and also saved the cost of programming development. In the second CAA module, since the mapping has been simplified by the first JAVA module, it takes the simple mapping function to map from the Scania schema to the Enovia schema. The execution time of the mapping module is around 8 seconds, but since the IEnovIn.sh issue affects the importing of the final result, it was not possible to determine the time for the entire import.

4.2.6.1 Collections

There have been evaluations on many collections in the available framework for the processing of XML data to SCANIA intermediate file. The conclusion is that “HASHTABLE” is the best approach for the system after these testing scenarios because of its easy lookup. When dealing with a large data collection, consideration should be given regarding the execution time and “HASHTABLE” appears to be the best fit for the system. XML data with a complex product structure should be processed in as short a time as possible in order to obtain faster outputs. The time complexity of this collection is O(1) in the best case which would suit the environment for obtaining productive results. Detailed implementation discussed in the previous section had productive results on tests made on different sizes of XML data.

4.2.6.2 Performance

As explained above the functionality implemented with “HASHTABLE” had very positive results. An XML with an average of 5MB takes around 7 seconds to transform the SCANIA intermediate format which is more efficient than has been estimated and
because the import process takes longer intervals this execution time is almost similar to the ECCO based output. The design and implementation matched the requirements.

4.2.6.3 Functional testing

The complete design can be considered as functional modules and each module had successful results in its individual execution and in the final integrated environment. The provision document can be produced without producing the intermediate result and this shows that the modules are functionally well designed and have positive results on their execution.

4.2.6.4 System Acceptance

According to the software system acceptance test, a JAVA based module is very integral into the current environment because it can easily fit into the system architecture. The CAA part cannot be evaluated because of unsuccessful results. However, it was designed as per DASSAULT SYSTEMES specification and the customization is very limited in this area. So even after the successful response from DASSAULT SYSTEMES for current requests, the test results will depend on XML data and pre-defined system analysis.

4.2.6.5 Testing Mapping Specification

To test the Mapping specification the execution environment of IEnovIn script should be setup as a touch file (.optionsfile) as is shown in the example below which was provided by DASSAULT SYSTEMES. This tracks the source, target schemas and mapping instructions. If the mapping file mentions the source and target schemas, eventually it will prompt recognition in the test.out file. The test.out contains all the tracked information and the result is that the written mapping specification is syntactically and semantically correct. But unfortunately, VPMCDMImpor inside the IEnovIn.sh file is unable to bind the CATEXXMAP to the maps defined in the mapping specification. This was the bug inside the DASSAULT SYSTEMES tool.

Example 4.1: Testing Mapping Specification on AIX machine

```bash
$ > touch .optionsfile1
$ > echo ImportCDM ImportData
+ VPMExAttrMngt k > .optionsfile1
IEnovIn.sh l SC2EN m l t p r c p
+ Scania c O f s
+ inputTemplate .xml>test.txt
```
4.3 CAA + CAA

This is Jingjing Qian’s work. It must be discussed here because the complete work should be seen as a collaborative approach instead of individual thoughts and the design phase of this module included contributions from both. The ideal goal of this thesis job is to deliver a CAA/C++ component that matches the project requirements. Even though, a Java based solution is designed which achieved the goal of the project, the basic intention was to create a CAA based component. This intention thus caused a re-think in relation to the solution which was using an out of the box mindset. According to the CAA documentation, a CAA component should use CAA based collections and this was the recommended means of creating a component using a CAA API. CAA has its own collections framework and can be used to satisfy the needs of the CAA applications. However, unfortunately, a complex data file such as XML cannot be processed with the help of these collections. This is particularly true as the data file varies in its size and a collection that has been chosen should support the processing of this file. The dynamic size, easy storage and easy look-up are the primary factors in choosing a collection to process this XML data. CAA CATHashTable fails in this perspective and it became an obstacle in designing the first part of the CAA module in the first proposed solution. This model also uses the IEnovIn script file provided by DASSAULT SYSTEMES to import the STEP data into the ENOVIA vault. The complete model architecture can be seen in Figure 4.12. There have been a few changes to the module that enabled there to be a successful implementation in relation to the first part enabled there to be the module.

Figure 4.12: Final proposal Based on CAA using C++ Containers and IEnovIn Script
4.3.1 C++ STL Containers

An API built on a basic framework should provide the functionality to access the underneath features. This idea offered an opportunity to experiment with C++ containers in CAA. The CAA API was designed based on a C++ STL library and supports the C++ containers in its application. This was the conclusion after using the API with the available containers. Thus this experiment has opened up the opportunity to introduce the most suitable container with an approximate functionality of HASHTABLE. C++ map is based on a RED-BLACK binary tree and internally it has the support for dynamic size allocation. This suited the basic requirement of a collection to process the SPECTRA XML data. Time complexity for HASHTABLE key based look-up is:

- a) best case is O(1) and
- b) worst case is O(n).

But a C++ map has constant look-up complexity with O(logn). This suited the second requirement of the collection requirement. Hence these factors offered the hope for a successful CAA module. C++ map was chosen to process the XML data to the target SCANIA format data with the assistance of a CAA SDAI environment.

4.3.2 No New Schema

Instead of changing the SCANIA schema to a new SCANIA schema, a CAA module has been implemented without modifying the requirements. This was the optimistic vision of the goal. This module made achieve the target. To interact with this EXPRESS schema in CAA SDAI, the schema should be compiled with a CAA compiler to generate an late type SDAI runtime environment. This late type runtime environment is used to create the empty SDAI model with the help of DS provided APIs. Later, the data is populated into this model with SPECTRA XML data after processing with a C++ STL container map. The detailed implementation will be discussed in the implementation section.

4.3.3 User Functions

There has been an opportunity to explore DASSAULT SYSTEMES EXPRESS-X customization features in this model. This extension is known as "USER FUNCTIONS" and helps to trigger CAA/C++ written code from an EXPRESS-X environment. When the first model was proposed, the vision of project was to maintain the standard EXPRESS-X specifications. but EXPRESS-X customization with "USER FUNCTIONS" is DASSAULT SYSTEMES proprietary and fell outside the scope with regards to the first model. However, to have a successful CAA model having all the features of ECCO, "USER FUNCTIONS" has to achieve complex jobs inside the EXPRESS-X environment. This model utilizes this feature to achieve the goal of the project. This feature assists in writing EXPRESS-C similar specifications but can be performed in a usual C++ environment. Along with the mapping module, a helper module that contains "USER FUNCTIONS" in CAA/C++ was written to handle the
EXPRESS-C customization in the old mapping file. For instance, a simple VIEW based mapping was written based on DASSAULT SYSTEMES EXPRESS-X specifications and the complicated iterations and manipulations were called from a helper module using the "USER FUNCTIONS" from the EXPRESS-X environment.

4.3.4 Provision Document

The provision document discussed in the previous sections is generated in the CAA SDAI environment similar to that for the JSDAI module. The creation of the provision document can be performed in two ways using this module: a) CAA/C++ module can be triggered from EXPRESS-X using USER FUNCTIONS or b) at the creation of a SCANIA intermediate file. The former is conducted while performing mapping between the two data models but is conducted at a later stage while creating SCANIA intermediate file. To create the provision document C++ STL libraries were used.

4.3.5 IEnovIn Script for Import Process

In the second part as no other alternative were available to explore the IEnovIn script was settled upon with the hope of making a successful STEP import with new mapping specifications.

4.3.6 Implementation

A complete CAA module has been implemented after performing the discussed changes. The first CAA part adopts the SPECTRA XML path together with the output directory location and creates SCANIA intermediate STEP data. The second part is executed inside the IEnovIn script to perform the import process. Both parts are executed by a Control-M batch scheduler. JavaMigrator, after receiving the XML file interacts with the Control-M and allocates the resource in order to execute XML to SCANIA STEP conversion by means of the Korn-Shell script. Finally, the vision was to execute the second part by Korn-Shell script using the Control-M to perform import process. This is not included in the sequence diagram but was the proposed design. The first sequence diagram assumes that it was triggered by the JavaMigrator after receiving the XML file and describes the conversion of XML data to intermediate SCANIA data. The second diagram assumes that it was triggered by the first CAA module with the assistance of the Control-M to perform the import process. The first CAA part performs the creation of the intermediate SCANIA STEP data along with the provision document from the XML data. The second CAA part does the conversion of the SCANIA STEP data to ENOVIA STEP data and performs the import using the IEnovIn script file.
4.3.6.1 CAA(XML TO SCANIA STEP)

When the JavaMigrator finds the XML file on the SIAM IP servers, it creates a directory in the spool area and communicates with the Control-M to allocate resources. It transfers the job to the Control-M after the initializing the parameters of the Korn-Shell script. This Korn-Shell script runs the CATEnvironment which has the CAA runtime. The runtime consists of the CAA workspace with three modules. The SCANIA module is the CAA module with a SCANIA format EXPRESS schema which is compiled by the CAA compiler to the CAA runtime. XML2STEP is a CAA module that performs the real conversion of XML data to SCANIA STEP data. XML2STEPExe is a load module that has the control of the execution which calls the XML2STEP module exposed methods. Once the script execution is triggered by the Control-M, it looks for the load module inside the provided workspace and finds the XML2STEPExe modules exposed main method. The GeoXML2STEP is the class in the XML2STEPExe which has the main method. It creates the instance of the GeoXMLParser of the XML2STEP module and interacts with its exposed methods for further processing. Figure 4.13 has an illustration of class diagrams of the two classes. The main workflow is inside the GeoXMLParser class of the XML2STEP module. The interaction between the GeoXML2STEP and GeoXMLParser is explained in the following steps.
Figure 4.13: CAA-Module Class Diagrams
The complete interaction between the two classes is well illustrated in the sequence diagram 4.14. The GeoXMLParser has three exposed methods named createSdaiModel, writeProvisionFile and saveSdaiModel.
4. Implementation & Results

- **createSdaiModel:**
  This model creates the SCANIA schema CAA SDAI model from the SCANIA model. To create the empty CAA SDAI model for this EXPRESS schema it uses the **CATSDM_Services API** provided by DASSAULT SYSTEMES in CAA. This API has a method named **CreateModel** which is used to create the empty SDAI model for the SCANIA schema. Once the SDAI model has been successfully created, it uses the SCANIA customized common logic **SCVXMLParser** to process the XML data to DomObject. As a further step, the method triggers its private methods named **createCustomObjects** and **createCustomObjectReferences**.

  - **createCustomObjects:**
    This method recursively enters into the XML tree structure inside the DomObject and builds a C++ container map from the XML objects. Later, the map is iterated to create the SCANIA schema objects. To create the entity instances of the SCANIA schema, **SDARuntime API** is used. This API has an exposed method similar to GetEntityDefinition, CreateEntityInstance, PutAttr which are used to populate the entity instances of SCANIA schema.

  - **createCustomObjectReferences:**
    This method traverses the XML tree in DomObject to create the references between the previously created SCANIA objects. Based on these references, Structures and Structure.Rows are created to maintain the relationship between the objects in SDAI environment.

- **writeProvisionFile** Provision document is created by processing the request and part objects of the populated SCANIA SDAI model. It uses some complex calculations to process the SDAI model to create this document. This provision document contains the breakdown information including the requested user and parts in the breakdown. A C++ STL **fstream** API was used to create this document.

- **SaveSdaiModel:**
  This method created the STEP P21 physical file for the SCANIA SDAI model. **SaveFileFromModel** is the exposed method of **CATSDM_Services** to create the STEP P21 physical file from the SDAI model. The SDAI model in the scope of one CAA module can be accessed in another module, So the **GeoXMLParser** acts as an intermediate bridge between the LoadModule and CATSDM.Services.

### 4.3.6.2 CAA(SCANIA STEP to ENOVIA)

The execution flow of this part is assumed to be inside a Korn-Shell script which is handled by the Control-M. After the SCANIA STEP data has been created in the first part,
another Korn-Shell script describing the import process is triggered in the Control-M. This Korn-Shell script is a wrapper script for the IEnovIn script file. The parameters are initialized and an execution job will be transferred to the Control-M job queue. The import process is illustrated in Figure 4.15. This script file expects many inputs including 

- a) SCANIA EXPRESS schema compiled CAA module,
- b) ENOV_DATAX compiled CAA Module,
- c) scriptTemplate
- d) customer policy script
- e) EXPRESS-X mapping CAA module between two EXPRESS schema files
- f) SCANIA STEP P21

![Diagram of import process](image)

**Figure 4.15: IEnovIn script with Mapping module**

After assigning the respective values for the parameters in the IEnovIn script, the wrapper script is executed. The import process is performed in two step process. In the first step, the SCANIA STEP file is processed and translated to the ENOVIA format and in the second step the data is imported into the ENOVIA vault. In the first step, the mapping module is used to translate the data but internally the mapping module triggers EXPRESS-X compiled specifications coupled with "USER FUNCTIONS" to perform the complex translation job. The complete interaction between the SC2EN mapping module and USERFUNCTIONS is illustrated in the sequence diagram 4.16. After initializing the various variables in the wrapper script, the SC2EN module is triggered to perform the data translation process. SC2EN consists of a Mapper class which is inherited from the DASSAULT SYSTEMES provided API named StdMapper. This API has two abstract methods named Convert and Close. The Close is already implemented in the API level and need not be inherited but to perform the data translation the Convert method should be implemented. The complete interactions with "User Functions" should be conducted within the scope of the method. This method has two input parameters named fromContner and toContner. These two CATICContainer objects contain the CAA SDAI models of the source and target EXPRESS schema files. The source SDAI model is populated with SCANIA STEP data in the import process and has to be translated to the target ENOVIA format in toContner CATICContainer object.

EXPRESS-X specifications in the SC2EN CAA model contains USERFUNCTION calls. These "USER FUNCTIONS" are defined in a separate CAA module
Figure 4.16: CAA Second Part Sequence Diagram
named “STEPFunctions” inside the same framework. To enable the User Function coupling with the EXPRESS-X specifications, the module interface should be included in the Map-per class. At various intervals and on demand, different ”User Functions” were created to perform the ECCO standard EXPRESS-C instructions in CAA/C++ notations. It is difficult to demonstrate the “User Functions” without exhibiting the EXPRESS-X specifications. But SCANIA to ENOVIA EXPRESS-X specifications are not included in the report because of SCANIA secure policy. Figure 4.16 can provide some idea related to their usage in the current model.

However, after compiling the mapping module in CAA environment successfully, an execution error appeared at the importing stage. The target schema and source schema could not be recognized in ENOVIA server although the mapping file could be. Different mapping modules and step files were attempted, but an execution error appeared at the importing stage the importing task. The assumption was that there was a bug in the importing step. An urgent service request has been assigned to DS, it is handed into the research and development team in DS, the issue is still under investigation.

4.3.7 System Evaluation

In the CAA+CAA module, the first CAA module takes responsibility for converting the XML file to the intermediate result, which actually performed the same task as the ECCO with no significant difference in the execution time, but the CAA module benefits the Scania’s software developers in that no specific knowledge is required in relation to the product data modeling language and they are able to read and edit the code in the CAA environment, which thus totally achieved the project goal. In the second CAA module, this takes the complicated mapping relationships between Scania schema and Enovia schema. It used the user function in the CAA to develop the mapping features and it also provided the developers with more in-depth thinking in relation to the usage of the user function in the CAA, the user function is embedded in the CAA environment but could perform specific data modeling functionality, and also provides a good interface to the CAA when some C or C++ limitations in the CAA module are required by the system.

Two solutions are available, at present, in the current system and it broadens the developer horizons in specific product data modeling fields, and also offers users a wider thinking about the usage of JAVA and CAA. Both JAVA and CAA could be used as development tools to develop the applications in some specific area, not just in the product data modeling field but also in some other industrial application areas.
5 Conclusions & Future Work

Many challenges existed throughout the project and many work-around techniques were used to overcome the obstacles. After months of research on the proposed problem domain, the final conclusion is to answer the following questions which can be considered as the conclusions and future work to this project.

5.1 Outcome

What was the overall outcome from the months research on this problem?

To some extent, there was no solution from Dassault Systems for the various API related problems. Even though solutions will be delivered in the future for the current obstacles in API, they will still have to be incorporated into these models and should be tested as to whether the goal has been reached. The solutions to these problems might give rise to new problems and might need more support from the Dassault Systemes. Hopefully, if concrete results are obtained then it may be possible for these to be implemented by some members of the team into the system architecture But what happens if the solutions lead to new problems? More research would then be required. It will be too ambitious to replace the current component and it might prove more productive to build up technical knowledge in order for there to be future support until the complete goal has been achieved.

5.2 Feasibility

How far it is feasible to use the proposed solutions in the current situation?

The problem had posed to different solutions based on different technologies. Each successful solution has its own advantages and disadvantages. The usage of C++ containers have not explored to date and this was an experimental part of this thesis work. An acceptance test in the current production environment should be conducted before there is any integration within the architecture. However, much depends on Dassault Systemes response and further work is required after their solutions to the problems.
5. Conclusions & Future Work

May 14, 2012

So, at this stage, the recommendation is not to proceed with any current component replacement unless a strong reason exists.

5.3 Impact on Future

What will be impact on future?

There is a significant impact to the future of this project based on these solutions. A number of disadvantages exist in the JAVA based module and the CAA based module. A Java module is platform independent but it requires exibility in EXPRESS-X mapping specifications. A standardized approach in a JAVA module is a better but a new standard EXPRESS VIEW should be proposed for SPECTRA instead of using simple XML standard. If a new VIEW is proposed, other means should be adapted for the creation of the provision document. A great deal of research is required in the SPECTRA area as life is to be made easier in relation to the current problems.

It is possible to make EXPRESS-X mapping simple but requires a modification in the current SCANIA standards. A pure Java based component can be developed if the API currently in use has both EXPRESS and EXPRESS-X support. But JSDAI which is used in one of the solutions has limitations in its EXPRESS-X specifications. Currently the implementation of EXPRESS-X in JSDAI is in a draft form and it might evolve to full standards in the near future. A complete Java based solution can assist the team to handover the support process to the JAVA support team but the current solution is a mix of JAVA and CAA which might lead to ambiguities in support. Thus, to adapt this JAVA module, the proposal is to await the complete package release of JSDAI.

The third proposal does satisfy the goal of the project but it uses the C++ containers to perform many tasks. It also uses the Dassault Systemes customization of EXPRESS-X mapping specifications. EXPRESS-X customization is a non-ISO standard and it will prove difficult to implement again in the future if a new CAx environment has been chosen in place of the Dassault Systemes applications. C++ containers usage is not appreciated in a CAA application according to the CAA documentation.

5.4 Alternative

Do other alternatives exist to the current component in market?

Many other components such as NIST STEP tools and Jotne EXPRESS package which is similar to ECCO. If a replacement for ECCO is mandatory and there is no positive response from Dassault Systemes for the current issues, SCANIA will have to make a choice from those which are available to test their functionalities and support features as to whether they are similar to ECCO standards and if, in fact, they
do exist. They are not chosen as part of this thesis job to experiment with because of non-availability of non-commercial versions.
6 APPENDIX

6.1 Existing Architecture Sequence Diagram With ECCO

Figure 6.1: Spectra Breakdown
REFERENCES


[3] LKSOFT GMBH. JSDAI Tutorial. vii, 26, 27, 28


[7] WARREN HASLAM. Mapping Between EXPRESS Models. 21, 23

[8] ADRIAN LAUD DAVID L. SPOONER IAN BAILEY, MARTIN HARDWICK. OVERVIEW OF THE EXPRESS-X LANGUAGE. 17, 18


[12] PDETECH. PROGRAMMERS GUIDE. vii, 24, 25, 26


[14] JOHAN REGNSTRM. Introduction of Component-Based Architecture on a Data Migrating Java Application. 31

[16] ISSUED BY: KARL HALLIN UTE/FREDRIK GUSTING. Software Design Description SDD019. 9

[17] UNIVERSITY OF MANCHESTER UK WARREN HASLAM, CAD GROUP. Mapping Between EXPRESS Models. 24
