OULUN SEUDUN
AMMATTIKORKEAKOULU

Markus Jääskelä

PLM-ERP INTEGRATION

PLM-ERP INTEGRATION

Markus Jääskelä
Bachelor´s thesis
Autumn 2011
Information Technology and
Telecommunications
Oulu University of Applied Sciences

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma

Tekijä(t): Markus Jääskelä
Opinnäytetyön nimi: PLM-ERP -Integraatio
Työn ohjaaja(t): Pertti Heikkilä(OAMK), Rauno Haime(Technia)
Työn valmistumislukukausi ja -vuosi: Syksy 2011
Sivumäärä: 42

Tämä opinnäytetyö tehtiin Technia Oy:lle. Technia Oy on Pohjoismaiden johtava tuotteen elinkaaren hallintaan erikoistunut yritys.

Tämän työn tavoitteena oli kehittää Technia Oy:n asiakasprojektiin kuuluvaa Enovia-PLM-järjestelmän ja SAP-ERP-järjestelmän välistä integraatiota. Integraatiossa siirretään nimikkeitä sekä nimikkeiden muodostamia rakenteita. Nimikkeiden integraatioon toteutettiin uuden toimintamallin vaatimat lisäykset. Myynti- / toimitusrakenneintegraatio suunniteltiin ja toteutettiin asiakasvaatimusten mukaiseksi yhteistyössä SAP-ERP-integraatiotiimin kanssa.

Integraatio toteutettiin käyttäen Technia Oy:n omaa asynkronista työympäristösovellusta AWEa. Integraation toteutuksessa käsitellään myös xml-parsetusta, tiedon keräämistä väliaikaiseen talteen sekä sen muokkaamista lopulliseen muotoonsa.

Työn tavoitteet saavutettiin ja työn lopputuloksena syntynyt integraatio otetaan asiakasyrityksessä käyttöön yhtäaikaa päivitetyn SAP-ERP-järjestelmän kanssa. Työn lopputulosta voidaan käyttää jatkossa muissa asiakasprojekteissa pohjana integraatioille.

Asiasanat:
PLM, ERP, Integraatio, Enovia, SAP

ABSTRACT

Oulu University of Applied Sciences
Information Technology and Telecommunications

Author(s): Markus Jääskelä
Title of thesis: PLM-ERP integration
Supervisor(s): Pertti Heikkilä(OUAS), Rauno Haime(Technia)
Term and year when the thesis was submitted: Autumn 2011
Pages: 42

This Bachelor´s thesis was done for Technia Oy. Technia is the leading supplier in the Nordic area of Product Lifecycle Management (PLM) Solutions for creating and managing product information throughout the entire product lifecycle, from product planning, development and design to production, sales and support.

The object of this thesis was to improve the integration between Enovia PLM system and SAP ERP system in Technia's customer project. The integration is done by configuring and improving the Enovia PLM system and AWE plug-in. As a result of this integration the customer's end user can send Items and sales / delivery structures from Enovia to SAP.

The results of this thesis meet the objectives set in the beginning. The end product has passed the customer's acceptance tests and is ready to be implemented to the customer's production environment.

Keywords:
PLM, ERP, Integration, Enovia, SAP

CONTENTS

TERMINOLOGY

AWE	Asynchronous Work Environment
Baseline	Describes the current Ebom structure of the item at the moment when Baseline is created.
Bom (Bill of Material)	Hierarchical Item structure
Ebom	BOM altered by design and product development
Enovia	The Product Lifecycle Management solution used in this thesis.
ERP	Enterprise Resource Planning
Item	Type of object in information model hierarchy, created in PLM system.
Material	SAP representation of an item
Obom	Sales / delivery Bom
Part	ENOVIAs representation of an item
PLM	Product Lifecycle Management
Promote	Transfer the Item / Obon to the next state of its life-cycle.
SAP	The Enterprise Resource Planning software used in this thesis.
SAP-PI	The SAP-ERP-system plug-in

1 INTRODUCTION

Product lifecycle begins when the first idea of the new product is invented. Over its lifecycle the product goes through the states of definition, design, production, marketing, sales, maintenance, etc. until it reaches the end of its lifecycle. Over these states the data related to the product (drawings, specification documents, marketing documents, etc.) increases enormously. The companies must handle all the data related to all their products. A solution to handle the growing amount of product related data is PLM (Product Lifecycle Management). PLM is not the only one operating with product related data. Enterprises' commonly have other systems (e.g. ERP (Enterprise Resource Planning), sales configurations) and applications (e.g. CAD (Computer Aided Design), customer relationship management tools) that also use and handle product related data. (1.) To be more efficient and cost saving the enterprises have to get all the systems and applications to cooperate with. The cooperation between systems and applications are implemented with integrations.

This thesis describes the integration between Enovia-PLM-system and SAP-ERP-system. Enovia is a PLM system produced by Dassault Systèmes and SAP is an ERP system produced by SAP AG. The integration is done by configuring and improving the Enovia PLM system and AWE (Asynchronous Work Environment) plug-in. As a result of this integration, customer's end user can send Items and Oboms from Enovia to SAP. The customer uses another subcontractor to develop the SAP side of the integration. Therefore this thesis only describes Enovia side of this integration until the message is sent to SAP and how the return messages from SAP are handled.

To make the entity of the thesis better and more valuable for future use, the operating environments chapter (chapter 2 Operating Environments) explains the PLM, ERP and AWE at a general level. Because the actual work contains information that is not intended for the public domain, the integration is described at a general level and the actual code is not attached.

This thesis is a part of the Technia PLM- customer project. Techia is the leading supplier in the Nordic area of Product Lifecycle Management (PLM) Solutions for creating and managing product information throughout the entire product lifecycle, from product planning, development and design to production, sales and support. (2.)

2 OPERATING ENVIRONMENTS

In this chapter the operating environments used in this work are explained so that even readers who are not yet familiar with these environments can see the value of this work.

2.1 Product Lifecycle Management

Product lifecycle management (PLM) is a systematic, controlled concept for managing and developing products and product related information. PLM offers management and control of the product process and product related information over the product lifecycle. (3.)

The core of PLM is the creation, preservation and storage of information relating to the company´s products and activities, in order to ensure the fast, easy and trouble-free finding, refining, distribution and reutilization of the data required for daily operations. The PLM is a holistic business concept developed to manage a product and its lifecycle including not only items, documents and Bom´s, but also analysis results, test specifications, environmental component information, quality standards, engineering requirements, change orders, manufacturing procedures, product performance information, component suppliers and so forth. (3.)

The main benefits of the PLM system is that it makes it possible to cut the time and cost of product development, its ability to act faster to the changed market needs, a better quality and more innovative products and services, improved comprehensiveness and relationships with customers, suppliers and business partners and its simplicity in tracking or sharing product data inside or outside the enterprise.

2.2 Enterprise Resource Planning

Enterprise resource planning (ERP) is an accounting oriented, relational database based, multi-module but integrated software system for identifying and planning the resource needs of an enterprise. ERP provides one user-interface for the entire organization to manage product planning, materials and parts purchasing, inventory control, distribution and logistics, production scheduling, capacity utilization, order tracking as well as planning for finance and human resources. (4.)

In this specific customer case where the customer itself does not produce any of the parts used in the end product, the ERP system is mostly used to manage the manufacturing resources and the logistic and financial part of the customer's business.

The ERP system used in this project is called SAP. SAP is a product of a German company called SAP AG.

2.3 Asynchronous Work Environment

Asynchronous Work Environment (AWE) is a stand-alone, separate java application designed to be used for background and long running activities. The core AWE provides a very limited functionality, basically only a way to extend its functionality with java code via Service Provider Interface (SPI), a similar concept to plug-ins for example in web-browsers. It also has the same core plug-ins that provides a basic access and work control with Matrix. For example, AWE is suitable for background tasks such as report generation, performing long running tasks, monitoring, integrations to other systems etc. (5.)

Normally AWE is used as a separately running application, launched from a command line, but it could also be integrated (or embedded) into some other application easily (5).

AWE is Technia’s own application. AWE is sold to the customer as a part of Enovia PLM solution and never sold as a separate product.

3 SPECIFICATION OF THE INTEGRATION

This chapter describes the situation of the integration before the author started to work with it. This chapter also specifies how the architect specified the integration to work.

3.1 Starting point

The client has multiple plants all over the world and every plant have their own system entities at the moment. Even though multiple different programs are used, the main concept has been the same. The customer is using Enovia PLM system as a PLM solution. Enovia is used to handle documents and items from other programs. Enovia also assembles and handles the structures of items. The customer is using SAP ERP system as an ERP system. SAP handles the manufacturing resources, logistic and financial part of the customer's business.

Previous release of the integration contained item integration. The aim of the previous integration was to cut down the manually made work and errors that were consequences of human made work. In the previous release the customer was able to send an individual item to SAP. The integration function was only available for some types of items. Even when the items were transmitted correctly to SAP, the customer had to create the structures manually before it was able to exploit the items.

In the previous release the end user was not able to see from Enovia if the transmission was successful or if it had failed somewhere on its way. If the message reached the SAP system but was not transmitted successfully or the message did not have the correct data, SAP sent a return message with error information to AWE. When the error message arrived, AWE handled it and forwarded the error information to pre-defined persons via e-mail. The email notification was not perceived to be effective enough and therefore a new solution to handle errors in the integration had to be invented.

3.2 Requirements

The aim of this project is to improve the customer's systems in the way that the customer can correspond better to the existing business needs. After this project the customer should be able to act faster and more reliably and operate globally. The customer also wants to reduce and harmonize the operating systems in several sub-regions to improve the re-usability and visibility of data and to cut back the number of systems to be maintained. As a result of the harmonizing of the operating systems the client should be able to combine all the product information created in different plants all over the world and transfer the information to one global ERP system.

At this part of the project the customer wants to improve the Enovia and SAP systems and the integration between them to handle Oboms(Sales / delivery Bill of Material). Because the customer orders all the necessary items from subcontractors and does not actually produce any item internally the Ebom integration is not implemented at this point. The item integration also has to be updated to send the design items to SAP and some improvements to the message content have to be done as well. The design items from CAD-systems are brought to Enovia by a separate integration.

The integration expands to transfer the hierarchical structures of the items. Therefore the success of the earlier transferred items must be able to be verified from the user interface.

3.3 Specification

The Integrations are specified to begin with the promote action (see figure 1). Before the Obom integration can be activated, the end user must create a Baseline from the top item. The baseline describes the current Ebom structure of the item at the moment when the Baseline is created. The Obom integration begins when the end user promotes the baseline from the WIP state to the "As Purchased" state (6). The item integration begins when user promotes the item

from the “Approved” state to the “Release” state or from the “Release” state to the “Obsolete” state. The items can also be sent to SAP with a special button. If an item is sent to SAP with the button action, the item state does not change. This button is mostly used to send the Sales items and the top item of the unfinished structures to SAP. The sales items and top items must be able to be sent SAP because usually a part of the order has to be ordered before the order has been fully designed.

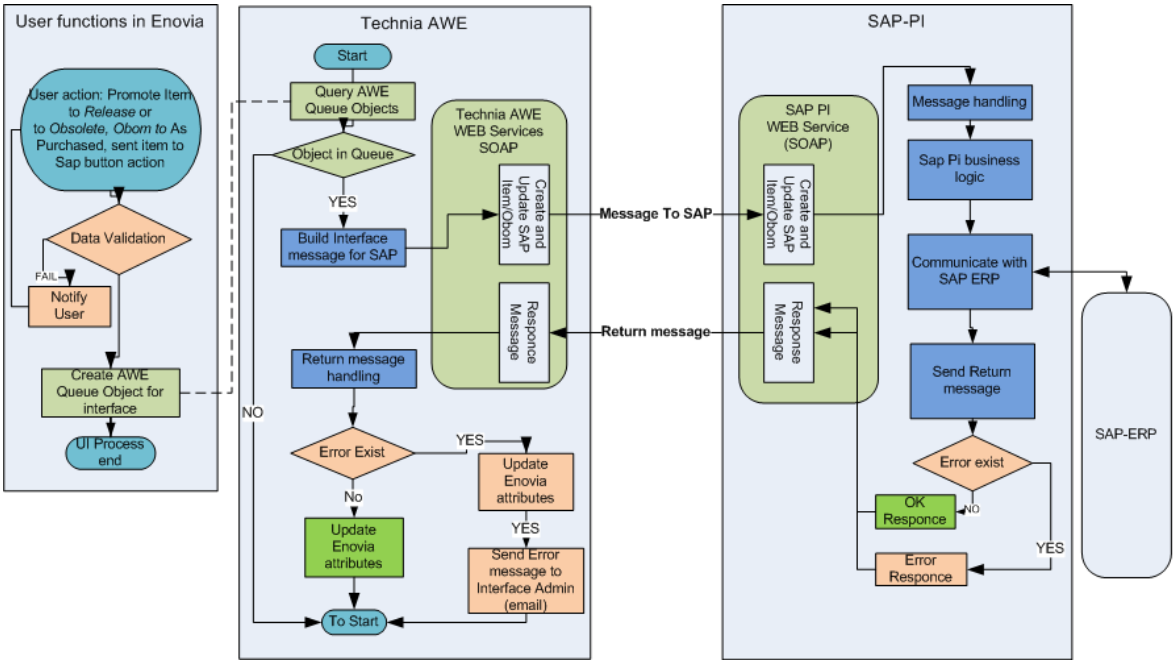

FIGURE 1. Simplified picture of Integration specification

When the Obom is sent, the integration only sends the released branches of the structure. All items in the Obom message must have been sent separately and successfully to SAP before the Obom message can be sent. Item and Obom messages have to pass the data validation before they are sent to SAP.

Figure 1 also describes in which system the functionalities take place. The user actions and data validation (described in chapter 4.2 Data validation) take place in Enovia. The build interface message for SAP in the Technia AWE area take place in AWE (described in chapters 4.3 Item message formation and 4.4 Obom

message formation). The xml-file parsing takes place before the Obom message data validation in Enovia and the build interface message in AWE.

SAP sends a return message via SAP-PI. The return message is handled in AWE, and AWE acts regarding to the success of the integration.

3.4 Concept

The project is specified to follow the AGILE software development methodologies. All activities in this project are scheduled according to the AGILE software development principles.

Technia's project group consists of 14 persons all together: eight developers, three business consultants, two managers and one test person. The ERP integration is carried out with the author as a developer and an integration specialist as a consultant. The rest of the group concentrates to improve the Enovia system and CAD integration.

Concept of the ERP integration

To avoid long waiting times in the Enovia usage, the integration is specified to be asynchronous. Asynchronous integration needs a separate return message to notify the end user of the errors.

The Integration consists of four systems: Enovia, AWE, SAP-PI and SAP. AWE is connected to Enovia's database via RMI. AWE core plug-ins contains the required data model and functionalities to be used with Enovia. AWE and SAP-PI use SOAP (Simple Object Access Protocol) over the HTTP protocol to communicate. The information is transferred in WSDL (Web Service Description Language) files. If any connection errors occur the Web Service notifies of them in the AWE log.

SAP-PI is an asynchronous integration platform. SAP-PI validates the message from AWE, formats information to the specified form and forwards the formatted message to SAP. After SAP has received the message, it sends a return message to AWE via SAP-PI. SAP-PI connects the return message with the current message information before it is sent to AWE.

4 PLM-ERP INTEGRATION

This chapter of this thesis describes the actual work the author has done in this project. Everything the author has done is done with the guidance and supervision of an integration specialist consultant. It is mentioned if someone else has done the work instead of the author or if it is done together with someone.

The integrations begin when end user promotes the item/baseline or item is sent to SAP with the button existing in user interface. The promotion/button action first launches the validation check trigger.

The validation trigger refers to a program which then validates all the mandatory attributes. The baseline integration has two validation triggers; the first one validates the baseline object mandatory attributes and the second one validates the data in the xml-file connected to the baseline object. The item integration has only the mandatory attribute check trigger. If the validations are not successful, the end user is notified and the program returns to the state where the integration has been launched.

If the validation is passed successfully, the action trigger is launched. The action trigger creates an AWE queue object. AWE has an endless loop which checks if new queue objects exist in Enovia database. If a new queue object exists, the integration begins on the AWE side. The integration action depends on if the current object is an item or a baseline, and if it is an item, it also matters from which state the item is promoted. AWE creates the message based on the action used, and after the message is formatted, AWE sends it to SAP-PI.

If the item is promoted from the “Approved” state, the message to SAP is CreateOrUpdateItem. If the item is promoted from the “Release” state the message to SAP is EOLItem. In the Obom integration the message is CreateOrUpdateOrderBOM.

The integration process is asynchronous but the end user can follow the progress of the integration from object attributes. The “ERP transfer status” -attribute value is updated regarding to the integration progress.

4.1 Message structure to SAP-PI

The schemas of the integration messages are defined together with the SAP-PI integration team. The schemas are designed based on the customer requirement.

4.1.1 Common message parts

Some of the classes created earlier for the item integration are able to be reused in the Obom integration. Both integrations use the MessageHeader and the ChangeObjectHeader sections to convey the basic message information. Table 1 describes the contents of the MessageHeader section and ChangeObjectheader section.

MessageHeader	sourceSystem
	targetSystem
	interfaceOperation
	dateTime
	controlObject
	controlObjectOwner

ChangeObjectHeader	changeNumber
	changeType
	changeDescription
	changeStatus
	effectiveDate

TABLE 1. The content of the sections MessageHeader and ChangeObjectHeader.

The MessageHeader section includes elements for target system, source system, interface operation, current date, control object and control object owner. The client has the test environment, development environment and production

environment. The target system element is in the message to clarify to which SAP environment the message will be sent. The interface action element declares the integration used.

The ChangeObjectHeader section is in the message as a future option. The SAP system includes the ECM (Engineering Change Module) module which is used to manage engineering changes. The customer does not use this feature at the moment but the feature will be introduced in the future.

4.1.1 Item message

With the item message client can create a new item to SAP, update the existing item in SAP or deactivate the item in SAP. The deactivation is done by sending EOLItem (End of Lifetime) message, and other actions are done by sending CreateOrUpdateItem message. Even though the messages are used differentially, both messages use the same form.

The Item message contains MessageHeader, ChangeObjectHeader and Material sections. MessageHeader and ChangeObjectHeader are described in chapter 4.1.1 Common message parts. The material section content had been already done in the previous release. To this release the client wanted to update the material section to meet the new requirements. Table 2 below describes the updated content of the Material section of the item message.

Material	xplantMaterialStatus		
	xplantMaterialTerminalDate		
	objectName		
	materialNumber		
	division		
	materialType		
	itemCategoryGroup		
	materialGroup		
	unitOfMeasure		
	netWeight		
	grossWeight		
	weightUnit		
	sizeDimensions		
	mfgSpecification		
	basicMaterial		
	serialNumberControl		
	textualDescriptions	shortTexts	itemDescription
			ItemAdditionalDescription1
			ItemAdditionalDescription2
		longTexts	language
			text
		purchaseOrderTexts	line
		InfoText	line
	manufacturerPart	partNumber	
		manufacturerId	
		manufacturerName	
		manufacturerShortName	
		country	
	plant	plantId	
		countryOfOrigin	
		mrpType	
		lotSize	
		reorderPoint	
		commodityCode	
		storageLocation	
		productionMakeBuy	

TABLE 2. The content of the Material section in the Item -message.

To this release the client wanted to add the division and PlantId elements. The value to the division element comes from Object which is connected to the current item with relationship “Design Responsibility”. PlantId is a new attribute for all the items. The customer also wanted to modify some elements to behave differently. If the item state is obsolete the “Xplant status”- element value is set to “Z1”. In any other state the value is set to the empty string. The “GrossWeight” element value is not set anymore. The values for elements “storageLocation”, “productionMakeBuy” and “serialNumberControl” are set to the empty string.

4.1.2 Obom message

The Baseline structure is created from the Ebom (Engineering Bill of Material) structure when the baseline object is created. The Obom message is derived from the baseline objects structure and only the released branches below the top item connected with the “Ebom” or the “Manufacturing Equivalent” relation-ship are selected. To the Obom message the Sales item is added even it is not copied when the baseline structure is created from the Ebom structure. FIGURE 2 shows an example of the structure lifecycle

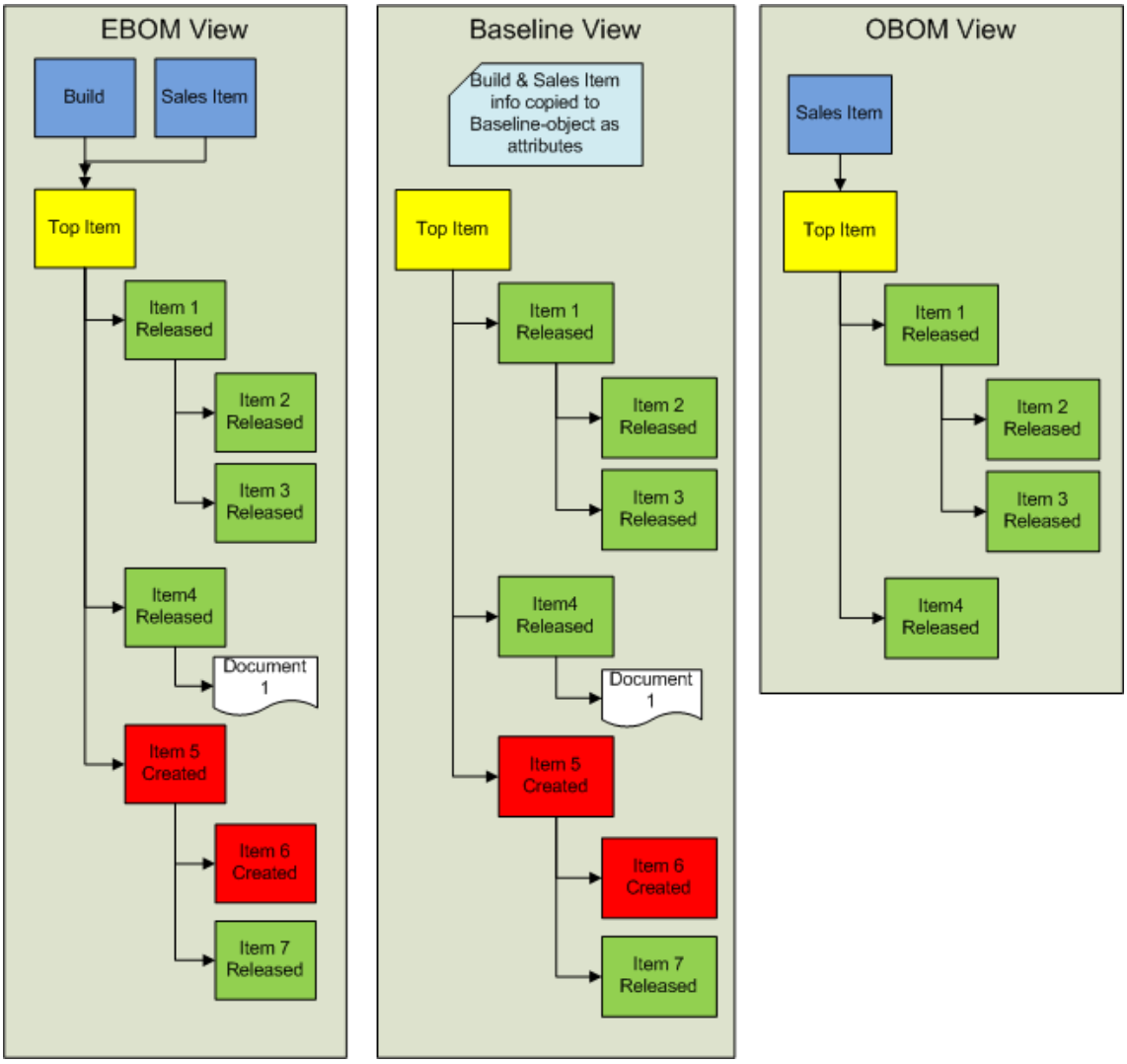

FIGURE 2. An example of Structures Lifecycle from Ebom to Obom

The Obom message had not been executed earlier and the structure could not be sent as a whole structure. The structure had to be divided into smaller pieces.

The message includes the MessageHeader and the ChangeObjectHeader sections which are described chapter 4.1.1 Common message parts. The PlantItems and the OrderBom section are new and only used in the Obom message. The content of the PlantItems and the OrderBom sections are described in table 3.

PlantItems (Item)	ObjectName		
	MaterialNumber		
	MaterialType		
	Plant		
OrderBom	**OBomWBSHeader**	**Plant**	
		Order	Customer Code
			Customer Name
			Project Number
			Order Number
			Order Row Number
			Order Description
	OrderBomItem	**MaterialNumber**	
		Status	
		Plant	
		BaseQuantity	
		BaseQuantityUnit	
		Valid From Date	
		Item	Item Number
			Item Category
			Reference Point
			Material Number
			Item Quantity
			Item Bom Unit

TABLE 3. The content of PlantItems and OrderBom sections.

The PlantItems section includes a list of all the Items in the Obom structure. The OrderBom section includes the ObomWbsHeader and a list of OrderBomItems. Because the whole structure could not be sent as one, the structure is divided into multiple one level Boms. Those one level Boms are called OrderBomItems. For example, in figure 2 Obom View, Sales Item and Top Item are the first Or-

derBomItem. The top Item and Item1 and 4 are the second OrderBomItem. Item 1 and Items 2 and 3 are the third OrderBomItem. One Obom message typically contains hundreds of Items at many levels.

4.2 Data validation

To avoid problems in the integration and SAP system, all the information must be validated before it can be sent. Some of the information is additional and the integration can be successful without them, but most of the information is mandatory.

Data validations are implemented to the Enovia side in the way that the end user can be noticed and the data can be corrected before the actual integration begins. The data validations are done by using check triggers.

4.2.1 Item data validation

The item integration data is coming from one source. Therefore the item-integration data validation can be done with only one check trigger.

When the item data validation trigger is launched, the Java program to which the trigger refers is executed. The Java program gets the current object id as a parameter and the attributes needed in the validation from the Java page object. The program connects to the database and checks from the current object if any of the mandatory attributes are empty. The program does not validate if the data is correct, it only checks that the attributes are not empty. If an attribute is empty, the program puts the attribute name to a list, and after all the attributes have gone through, the program checks if the list is empty. If the list is not empty, the program notifies the user of that attributes are empty and prints out the content of the list below. After the user is notified, the program returns to the state where it was before the validation trigger was launched. If the list is empty, it means that all the mandatory attributes are filled and the program continues to the next trigger.

4.2.2 Obom data validation

The Obom message data comes from two sources: from the current baseline object and from the xml-file connected to the current baseline object. The attributes of the current baseline object can be validated with the same validation program used in the item integration. The xml data content validation requires a new validation trigger of its own. The xml data content validation validates the data inside the xml-file. The xml schema validation is done in the AWE core functions, and it is not a part of this thesis.

Order information and sales item information are set as attribute values to the baseline object. Those attributes are the ones that are specified to be validated from the baseline object. The program works exactly as the item validation works, only the validated attributes are different in the page object. When the baseline object validation is passed successfully, the program continues to the xml-content validation.

The Xml data content validation requires the xml-file handling. The program parses the xml data and creates hte OrderBomItems regarding to the Obom-structure (explained more detailed in chapter 4.4.3 Parsing the xml-file). The xml data content validation is divided into three methods. The first method checks that all the items in OrderBomItems are successfully transferred to the SAP. The second method checks that every OrderBomItem child items have the attributes quantity, find number and material group filled. The third method checks if any OrderBomItem parent item has several Commercial items connected to it with the relationship “Manufacturing Equivalent” as a child item.

Every section returns a list of items that does not pass the validation. The program collects the lists together, notifies the user of the errors and returns to the state where the validation trigger was launched. The user must fix the errors and disconnect the extra Commercial Items and then promote the baseline again until every list is empty.

4.3 Item message formation

The item message consists of three parts; MessageHeader, ChangeObjectHeader and Material sections. The ChangeObjectHeader is left empty and the MessageHeader element values are filled with context object attribute information. Therefore the only more complicate part is the Material section.

The AWE queue object contains information of the Item it is created of. The most important information is the current item object id. The id is used in the database query which collects attribute information from the current item and returns the item object with the attribute information. Most of the elements in the Material section can be populated directly with the corresponding attributes, but in some elements the collected data has to be handled before it can be used to fill the element. For example, the “ItemCategoryGroup” element value depends on if the current item attribute “MRPType” begins with the letters “ND” or not.

4.4 OBOM message formation

The OBOM message consists of four different sections: MessageHeader, ChangeObjectHeader, PlantItems and OrderBom. The MessageHeader and the ChangeObjectHeader are common message parts also used in item message.

4.4.1 The format of the PlantItems and the OrderBom sections

The formation of the Obom message is more complicated compared to the item message formation. In the item integration all the required information is available with one database query of the current item. In the Obom message information comes for several different locations. The baseline object includes the order information as attribute information. The most important information source in the Obom integration is an xml-file attached to the current baseline object. The schema of the xml-file connected to the baseline object is defined earlier by TVC (Technia Value Components) team when they created the functionalities of the baselines.

The OrderBom section consists of the ObomWbsHeader and the OrderBomItems sections. The ObomWbsHeader section elements are populated with the current baseline object attribute information. The baseline attribute information is available with the database query. The OrderBomItems section is a list of the OrderBomItem objects. The OrderBomItem-objects are received from the xml-file attached to the current baseline object.

The PlantItems section is a list of item objects that exist in the Obom structure. Every object includes basic information of the item as name, “ERP material number”, type and plant. Every item is in the list only once even though it might be in the structure several times.

After most of the work had been done the customer changed the specification and in the new specification the sales item had to be added as a PlantItem to the PlantItems list. The sales item also had to be added to the Obom structure above the top item. The PlantItem enlargement could be done by using the baseline-object information for the element’s name, “ERP Material number” and plant. The sales item name is set to the values to Name and “ERP Material number” elements. The item type element is set to the value ”Sales Item”.

The OrderBomItem is done by using the sales item information and the top item from the xml-file. The parent item element “ERP Material number” is the name of the sales item and the plant element is coming from the baseline object attribute plant value. Other parent item elements are filled with default values. Child item element “Item Number” is set to baseline-object attribute “Sales Order Position Number” value. The child item “ERP Material number” is the top item “ERP Material number” and had to be parsed out from the xml-file. The rest of the values are set as default values.

4.4.2 Check-out the xml-file from baseline object

Xml is generated and connected to the baseline object when the baseline is created. Every time the baseline object is modified, Enovia creates a new xml-file and replaces the old xml-file with the new one. The xml-file schema was de-

fined by the TVC-team when they created the functionalities for the baseline. The xml-file lists all the items and relationships in the structure with their attributes. The xml-file also includes a section where the items are listed with all their child items and the relationships connecting the items. This section also holds the information if any items are removed from the structure.

Before any parsing was able to be done the xml-file had to be checked-out from the baseline object. TVC (Tecnhia Value Components) provides the checkout-File method which writes the file to the output stream. The method needs the object id, a format, file name and output stream as a parameters. One parameter is a boolean value and defines if the baseline object is locked or not. The object id is received from SAPOrderBOMIntegrationContext. The format and the filename are set to default strings “generic” and “baseline.xml.gz”. A new output stream is created and it points to a new temp file. The boolean value for the lock is set to false because the baseline object must be editable after the check-out.

The checked-out file is an archived format and it must be decompressed before it can be used. After the check-out is done the decompressing is done by setting the temp file to the file input stream, reading the file input stream to the buffer and writing the buffer to the new output stream which points to the new temp file.

4.4.3 Parsing the xml-file

Even the xml-file included all the information needed, it could not be used directly because it was in the xml-format. Parsing is a way to get the needed information from the xml-format to the wanted format. In this case the wanted format is Java objects.

The xml parse was first tried to handle with a digester. The digesting did not proceed as needed and the deadline was approaching, and therefore the way to parse the xml file had to be changed. Even though a baseline includes thou-

sands of items, the size of the xml-file stays reasonable. Therefore a DOM-parser which reads and loads the file into the memory could be used.

The xml-file includes all the structure and items exist in the Ebom structure. In the Obom message structure includes only the released branches of the structure below the top item. The released items must be connected to each other with the relation type “EBOM” or “Manufacturer Equivalent”. The documents and drawings are not sent to SAP.

The Connection and Items classes are created to collect and handle the items and the relationship from xml. The connection class includes a Java map for the relationship attributes and the methods to get and set an attribute, a child item and a connection object id. The Items class includes a Java map for item the attributes and a Java map for the child items. The Item class also includes the methods to set and get attributes and child items. The child items are only added one level above the parent item.

The OrderBomItem and PlantItems classes are the final forms of the objects used in the message. After the data from xml-file has been read to the Connection and Items objects, it is reprocessed to fill the OrderBomItem and PlantItems forms.

The Ebom structure in the xml-file is described inside the node’s tag. Every parent item in the structure has its own tag tagged with letter “n”. Inside the parent item tag is every child items tagged also with letter “n”. The lower “n” tag has attributes d, o, r and a removed attribute. The d attribute represent the direction, the attribute “o” value represents the child object id, the attribute “r” value represents the relationship object id and the removed attribute tells if current line is removed from structure. The direction attribute always has the value “f” as from direction. The removed attribute is only shown if the value is true. The figure 3 below is an example of the structure section in the xml-file.

```
<structure>
  <roots>
    <n o="25388.14427.54880.33306" d="f"></n>
  </roots>
  <nodes>
    <n o="25388.14427.54880.33306">
      <n r="25388.14427.23376.24939" o="25388.14427.57760.52196" d="f"></n>
      <n r="25388.14427.28604.63559" o="25388.14427.7408.6514" d="f"></n>
    </n>
    <n o="25388.14427.57760.52196">
      <n r="25388.14427.49956.49188" o="25388.14427.49956.5543" d="f"></n>
      <n r="25388.14427.8416.17058" o="25388.14427.8416.6905" d="f"></n>
      <n r="25388.14427.36744.60503" o="25388.14427.39496.45631" d="f" removed="true"></n>
      <n r="25388.14427.36744.41142" o="25388.14427.54076.61513" d="f" removed="true"></n>
    </n>
    <n o="25388.14427.7408.6514">
      <n r="25388.14427.31904.28413" o="25388.14427.31904.25562" d="f"></n>
    </n>
    <n o="25388.14427.49956.5543">
      <n r="25388.14427.9420.59462" o="25388.14427.9420.54019" d="f"></n>
    </n>
  </nodes>
</structure>
```

FIGURE 3. An example structure section in the xml-file.

The items are listed under the objects tag in the xml-file. Every item is tagged with the letter “o” and every attribute of the item is its own child tag tagged with the letter “s”. Every child tag has an attribute “s” whose value is the name of the item attribute, and all the child tags have another child tag “v” where the item attribute value is. Figure 4 below shows an example of an item in the xml-file.

```
<objects>
  <o mx="25388.14427.49956.5543">
    <s s="current.access[modify]" t="1"><v>TRUE</v></s>
    <s s="attribute[Critical Spare]" t="0"><v></v></s>
    <s s="attribute[Description SE]" t="16"><v></v></s>
    <s s="type" t="16"><v>Proprietary Item</v></s>
    <s s="attribute[Serial no]" t="16"><v></v></s>
    <s s="attribute[Production Make Buy Code]" t="16"><v>Unassigned</v></s>
    <s s="attribute[ERP Material Number]" t="16"><v>N00000235</v></s>
    <s s="id" t="16"><v>25388.14427.49956.5543</v></s>
    <s s="attribute[One Year Operation Quantity]"t="0"><v></v></s>
    <s s="attribute[MTBF Percent]" t="0"><v></v></s>
    <s s="revision" t="16"><v></v></s>
    <s s="attribute[Spare Part]" t="16"><v></v></s>
    <s s="to[Manufacturing Responsibility].from.name" t="16"><v></v></s>
    <s s="description" t="16"><v></v></s>
    <s s="attribute[Network]" t="16"><v></v></s>
    <s s="name" t="16"><v>N00000235</v></s>
    <s s="attribute[Activity]" t="16"><v></v></s>
    <s s="current" t="16"><v>Release</v></s>
    <s s="attribute[Manufacturer Code]" t="0"><v></v></s>
    <s s="attribute[Delivery Date]" t="2"><v></v></s>
    <s s="attribute[Startup Quantity]" t="0"><v></v></s>
    <s s="attribute[Description ES]" t="16"><v></v></s>
    <s s="attribute[Description RU]" t="16"><v></v></s>
    <s s="to[Classified Item].from[Part Family].name" t="16"><v>METAL PRODUCTS</v></s>
    <s s="attribute[Material Provider]" t="16"><v></v></s>
    <s s="attribute[Description FI]" t="16"><v></v></s>
    <s s="attribute[Spare Part]" t="16"><v>No</v></s>
    <s s="attribute[Unit of Measure]" t="16"><v>EA (each)</v></s>
    <s s="modified" t="2"><v>6/16/2011 11:17:20 AM</v></s>
    <s s="originated" t="2"><v>9/30/2010 2:38:49 PM</v></s>
    <s s="attribute[Description DE]" t="16"><v></v></s>
    <s s="policy" t="16"><v>Design Item</v></s>
    <s s="attribute[Weight]" t="8"><v>0.0</v></s>
    <s s="attribute[Description PT]" t="16"><v></v></s>
  </o>
```

FIGURE 4. An example item in the xml-file.

The relations are listed under the relationship tag in the xml-file. Every relationship is tagged with the letter “r” and every relationship attribute is its own tag tagged with letter “s”. Every child tag has an attribute “s” whose value is the name of the item attribute, and all the child tags have another child tag “v” where the item attribute value is. Figure 5 below shows an example of a relationship object in the xml-file.

```
<r mx="25388.14427.36744.41142">
  <s s="id" t="16"><v>25388.14427.36744.41142</v></s>
  <s s="attribute[Reference Designator].value" t="16"><v></v></s>
  <s s="attribute[Find Number].value" t="16"><v>40</v></s>
  <s s="attribute[Spare Part Quantity].value" t="4"><v>0</v></s>
  <s s="from.id" t="16"><v>25388.14427.57760.52196</v></s>
  <s s="to.id" t="16"><v>25388.14427.54076.61513</v></s>
  <s s="attribute[Quantity].value" t="8"><v>1.0</v></s>
  <s s="type" t="16"><v>EBOM</v></s>
  <s s="attribute[Reference Point].value" t="16"><v>43</v></s>
  <s s="attribute[Spare Part BOM].value" t="0"><v></v></s>
</r>
```

FIGURE 5. An example relationship object in the xml-file.

With the DOM-parser it is possible to take all the items and relationship into the node list and then again create a new node list of the item/relationship attributes. When the node list of attributes is done, all the required attributes can be collected. The Iitem attributes of one item object are collected to an instance of items class and then the instance is put to the Java map. The relationship attributes of one relationship object are collected to an instance of connection class and then the instance is put to a Java map. The nodes section is parsed only to collect all the objects that are not removed. All the connections in the structure are collected to the relationsInStructure list. If an object is removed and the user wants to add it to the structure again it is possible because the relationship object changes every time.

When a new instance of connection is ready, it has among other attributes “id”, “to.id” and “from.id” attributes. “to.id” represents the child item id and “from.id” represents the parent item id. The Items class method provides the addchild-method which can be used to add the child item to the child map of the parent item. A child item is not added if the child item state is not released. The connection type must be either “EBOM” or “Manufacturer Equivalent”, and the connection object id must exist in the relationsInStructure list. If the item type equals “Bulk Item” or ”Specification Item” it is not added as a child because the customer wants to remove them from the Obom message.

After all the relationships are parsed and handled, the top item has to be parsed from the xml-file. The top item is inside the structure tag. It is separated with tag “roots”. Roots tag has a child tag “n” and the top item id is the value of “n” tags “o” attribute.

4.4.4 Temporary data format to final form

After all the required data from the xml-file is parsed and loaded to temporary stores, the final formation of the PlantItems and OrderBomItems objects can be started. The formation is done by using the method createStructure.

CreateStructure is a recursive method meaning that it calls itself until the whole structure is handled. The method has the following parameters: Item, list of Items objects, OrderBomItems list, PlantItems map and plant attribute. The Item as a parameter defines the parent item. The items object map is made available in this method by setting it as a parameter. The OrderBomItems list and PlantItems map are created beforehand and they are populated over the process of the recursive structure creation. Plant information is attribute information from the baseline object and it is needed to create OrderBomItems and PlantItems. The item parameter is the only parameter changed every time before method is called.

When the CreateStructre method is called for the first time, the top item is set to the item. Every time method is called, the parent item is added to the PlantItems map. If the parent item has child items, the CreateOBOMItems method is called. CreateOBOMItems creates a new instance of the OrderBomItem class, populates the parent item and child items and returns the OrderBom-object. The returned OrderBom object is then added to the OrderBomItems list. After the OrderBomItem object is added the program continues to handle the child items. The child item is also added to the PlantItems map. An item can be both a child item and a parent item and an item may appear multiple times in the structure. In the PlantItems list every item should appear only once, and therefore the plant item objects are added to a map with their id as a key value. If the program adds the item several times, the map automatically replaces the old item

with the new one. The plantItems object consist of material number, material type, name and plant elements, and these elements are always identical wherever in the structure the current item exists. When the child item is added to the PlantItems map, the current child item is set to Item and the CreateStructure method is called again.

When the program has gone through the whole structure, the OrderBOMItems list is ready to be returned. The PlantItems map has to be converted to a list before it is returned. The convert is simply done by looping through the map and adding the current PlantItem to the PlantItemsList.

4.5 Return message

The integration is specified to be asynchronous, and to be sure of that the integration action has been successful, a return message is needed. The integration message is generated in SAP regarding if the message transport has been successful or not. Both the item integration and Obom integration use the same message form. The return messages are handled regarding the Interface operation. Below there is an example of the return message. The Obom integration in the example has been unsuccessful.

```
<MessageHeader>
    <SourceSystem>R10CLNT310</SourceSystem>
    <TargetSystem>BS_D_ENOVIA</TargetSystem>
    <InterfaceOperation>CreateUpdateObom</InterfaceOperation>
    <DateTime>2011-09-23T13:35:08</DateTime>
    <ControlObject>62048.22990.49284.30235</ControlObject>
    <ControlObjectOwner/>
</MessageHeader>
<Response>
    <MaterialNumber>48151874</MaterialNumber>
    <IDocNumber>19628700</IDocNumber>
    <ReturnCode>E</ReturnCode>
    <Messages>Error when saving BOM. Please see the application log.</Messages>
</Response>
```

FIGURE 6. An example return message.

In the previous release the return message generated both successful and unsuccessful messages, but only the unsuccessful messages were handled in AWE. AWE sent an email message to a specified group of users when the

transfer was not successful, and the user who sent the item to SAP might not have been notified at all if the transfer had failed. In this release the return message is always handled and object attributes are set regarding to the transfer success.

The return message has the Awe object id as an attribute. The Awe object has an attribute “args”. “args” includes the basic information of the Enovia object. Regarding to the Awe object “args” attribute, the return message can be connected to the Enovia object sent with the current Awe object. When the current Enovia object is known the object attributes can be updated regarding the transfer success. If the transfer has not been successful, Awe still sends an email to the specified group.

The Obom integration requires multiple return messages. Every OrderBomItem transfer is handled separately, and SAP generates a separate return message every time. OrderBomItems are collected to a list, and when a return message arrives, the list cell corresponding to the messages OrderBomItem is updated regarding to the success of the integration. Once all the return messages of the current OrderBom have arrived, AWE updates the baseline object attributes regarding if the OrderBom is transferred successfully or not. If even one OrderBomItem is not transferred successfully, the attribute “ERP Transfer Successful Status” value of the baseline object is set to “No”.

4.6 Testing

Testing in this Technia project was divided into 4 different parts: functional testing, internal integration testing, integration testing with the customer’s test group and user acceptance testing. The main track of the project followed the typical test plan. Internal integration testing in the PLM-ERP integration would have required an internal ERP system and therefore the typical test plan was not possible to be followed and the internal integration step was passed. The Data validation makes an exception because it is implemented on the Enovia side and does not require any communications of the system.

4.6.1 Functional testing

A developer must test every change and new functionality before any changes are committed to the subversion and installed to the internal test environment. The functional test is performed by a developer in developer's local environment. The PLM-ERP integration functional testing is performed by the author of this thesis.

In the PLM-ERP integration, the functional testing was performed with separate testing classes. Every method was tested both separately and latterly as an entity. The xml parse was first tested with an example xml-file. After the file check-out and file extraction had been successfully tested, the file check-out from the local environment was used in the testing.

The return message functionally could not be tested with real return messages. The return message functionality was implemented in earlier releases and was only updated in this release. Because the return message functionality was working already, only the updated parts were tested. The return message sets values to the object attributes regarding if the transfer is successful or not. In the functional test the test class connects to internal test environment and set the attribute values to the default item.

The functional test started immediately when the first issues were ready and lasted until the last bug was fixed. Every time a bug was found and the issue was returned to the developer the testing started with a functionally test and followed the test plan.

4.6.2 Internal integration test

The internal integration test was operated in Technia's internal test environment. The internal test is performed by Technia's test person. Awe is not installed in Technia's internal test environment, and therefore the internal integration test only covered the changes done in the Enovia side. The data validation

was implemented on the Enovia side and it was the only part of the PLM-ERP integration tested in the internal integration test.

The internal integration test started a month later than the development had started. The internal integration test lasted until the end of the project.

4.6.3 Integration test

The integration test was performed in the client's test environment. The test is lead by Technia's project manager and performed by the customer's PLM specialized test group. The bugs and changed business request found in this test where returned immediately to the make it possible for the developers to start solving the problems as soon as possible. Even the bugs and changed business request were returned to the developers immediately as they appeared, the updates to this environment were done in two-week cycles.

In the integration test the Enovia and SAP sides of the integration were tested together for the first time. Therefore the bugs in message formats and the communications between the systems were seen in this late stage. With the PLM-ERP integration related issues the update cycle was specified to be once a week.

The integration test lasted five weeks and included three core updates and five PLM-ERP integration updates. The package used in the last update was delivered to the customer user acceptance test. Over the integration test approximately 20 bugs were fixed. In addition 10 new/changed business requests appeared. All the high priority bugs and business requests were fixed immediately and some with the lower priority were left for the future.

4.6.4 User acceptance test

The user acceptance test was performed by the real end users of the systems. The test was performed in North America because the system will be introduced first in Canada. The testers followed the pre-specified use cases. The use cases were created to describe the testers' everyday use of the system.

The bugs and change proposals found in the user acceptance test were collected together and returned to the project management group. The project management group then decided if the bugs had to be fixed or changes had do be made before the release was delivered to the customer's production environment. Only major bugs were fixed and others were moved to the next release. The user acceptance test lasted two weeks, and one week was reserved for bug fixes.

4.7 Installation

The installation to the production environment is planned to happen a one and half month after the user acceptance test began. Before the release is delivered to the production environment, the installation is practiced with test test environment. The test environment is identical with the production environment.

The installation of this release requires extra work because the customer wants to harmonize and reduce the operating systems. As harmonizing and reducing of the operating systems requires multiple data migrations.

The Enovia and AWE updates are done by using Ant scripts. The first installation packages are created from the last revision of the code with the Ant scripts, and then the packages are moved to the destination environment. In the destination environment the packages are installed separately with the Ant scripts. After AWE is installed, the connection configurations need to be set to point to the correct SAP-PI.

The actual work in the installation is divided for two persons. The author of this thesis is responsible of hte AWE installation and another developer from Technia's project group is responsible for the Enovia installation.

5 RESULTS AND FUTURE DEVELOPMENT POSSIBILITIES

This chapter describes the results of this project and how the specified requirements were achieved. Future possibilities and improvement ideas are also reviewed in this chapter.

5.1 Results

Even though the schedule was challenging, the requested functionalities were delivered to the customer mostly as planned. The first PLM-ERP integration package was delivered to the customer test a day later than planned. The first package did not include all the functionalities as planned but the missing functionalities were delivered in the next deliveries.

The biggest challenges in this project were the schedule and the lack of knowledge. The schedule was tight and the summer holidays during the project made it even more challenging. Integrations require much understanding of the used systems and Awe requires a proper orientation before it can be used. The Awe knowledge transfer was arranged in the beginning of the project and integration business understanding grew as the project progressed. The handling of the customer's changed requirements in the given time also became a big challenge.

Some programming problems were encountered over the project. The biggest problem in the programming was parsing the xml-file. The parsing was first tried to do with the digester but due to the lack of digesting skills the parsing had to be done using more familiar ways. Some smaller programming issues were also encountered but they were solved quickly with the help of senior colleagues. The classes created during this project can be re-used in the future integrations.

Facing and resolving the issues are the best way to learn new skills. This project gave great base knowledge of integrations and led to the improved the programming skills. Over the project the knowledge of the PLM and ERP were improved too. Communication with the customer and with the customer's sub-

contractor was a great example of the communications in this business area and it improved the business communication skills. There is still a much to learn in the integration business and the learning process continues in the future releases.

In the next releases the integration is extended to include Ebom (Equipment bill of material)Mbom (Manufacturing bill of material) and Sbom (Service bill of material). The integration between Enovia and Sibel is also specified but the exact time is not defined yet.

The project followed the SCRUM framework for project management. Weekly meetings with the project group and almost daily conferences with the integration group (the author and a consultant) kept the whole project on track and in schedule. In addition the faced problems and changed requirements were handled faster and the outcome of the project met the customer's needs better.

5.2 Future possibilities

The greatest future possibilities are related to the AWE productization. The number of integrations between Enovia and other systems increase exponentially at the same time when Enovia spreads to other enterprises.

At the moment AWE is only used in Technia's projects. AWE is always heavily customized and once AWE has been installed the maintenance usually requires the same developer work that originally did the configuration. The documentation of customization and the improved error handling could be the first steps in the AWE productization. In addition, improvements in testing are necessary.

6 DISCUSSION

The aim of this thesis was to implement an integration between the Enovia PLM system and SAP ERP system. The integration was specified to include the item and Obom transfers to SAP and return message handling. The specified requirements were achieved. Even though all the development was not done as first planned, the functionality works as it was planned. The work outcomes have passed the customer integration test and are now ready for user acceptance test.

During the process of this thesis, the authors' knowledge of Enovia, PLM, ERP and SAP increased enormously. The programming skills were also improved and the company's product development became more familiar.

Technia's objective for the software development is to create classes which can be reused later. The work outcomes can and will be reused later in this project and in other integrations.

In all, the project was really successful. The objectives were reached and the customer is satisfied with the outcome. The development continues immediately after the next release specification has been finalized.

LIST OF REFERENCES

1. Kemppainen, Tapani – Kropsu-Vehkaperä, Hanna – Haapasalo, Harri. 2011. Product data ownership network. Research report. University of Oulu.

2. Technia Homepage, Technia in short. Available at: http://www.technia.com/About-Technia1/About-Technia/Technia-in-short/ Date of data acquisition: 16 October 2011.

3. Sääksvuori, Antti – Immonen, Anselmi 2008. Product Lifecycle Management. Springer.

4. Business Dictionary. Available at: http://www.businessdictionary.com/definition/enterprise-resource-planning-ERP.html Date of data acquisition: 16 October 2011.

5. Räty, Jani. 2009. Asynchronous Work Environment. Technia.

6. Hurskainen, Tero. 2011. Spearhead Design Specification. Technia.

7. Haime, Rauno. 2011. Spearhead Integration Message Mapping. Technia.