Lappeenranta University of Technology
Faculty of Technology Management
Degree Program in Information Technology and Communications Software

Candidate's Thesis

Heikki Korkala

DESIGN AND IMPLEMENTATION OF A PRODUCTION SYNCHRONIZATION EXTENSION FOR MICROSOFT DYNAMICS AX

Examiner: Erja Mustonen-Ollila D.Sc. (Tech.)

Supervisor: Erja Mustonen-Ollila D.Sc. (Tech.)

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Degree Program in Information Technology and Communications Software

Heikki Korkala

Design and implementation of a production synchronization extension for Microsoft Dynamics AX

Candidate's Thesis

2012

32 pages, 6 figures, 1 table, and 4 appendices.

Examiner: Erja Mustonen-Ollila D.Sc. (Tech.)

Keywords: axapta, dynamics ax, production, manufacturing, x++, morphx

In a just-in-time, assemble-to-order production environments the scheduling of material requirements and production tasks - even though difficult - is of paramount importance. Different enterprise resource planning solutions with master scheduling functionality have been created to ease this problem and work as expected unless there is a problem in the material flow.

This case-based candidate's thesis introduces a tool for Microsoft Dynamics AX multisite environment, that can be used by site managers and production coordinators to get an overview of the current open sales order base and prioritize production in the event of material shortouts to avoid part-deliveries.

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan koulutusohjelma

Heikki Korkala

Tuotannon synkronointitoiminnun suunnittelu ja toteutus Microsoft Dynamics AX -toiminnanohjausjärjestelmään

Kandidaatintyö

2012

32 sivua, 6 kuvaa, 1 taulukko ja 4 liitettä.

Tarkastaja: Tutkijaopettaja TkT Erja Mustonen-Ollila

Hakusanat: axapta, dynamics ax, tuotanto, valmistus, x++, morphx
Keywords: axapta, dynamics ax, production, manufacturing, x++, morphx

Tilausohjautuvassa, juuri-oikeaan-tarpeeseen -periaatteella toimivassa tuotannossa materiaalitarpeiden ja tuotannon ajoitus on ensiarvoisen tärkeä, mutta haastava tehtävä. Tehtävää helpottamaan on luotu toiminnanohjausjärjestelmiä, joiden pääajoitustoiminto huolehtii normaalitilanteessa aikataulutuksesta. Järjestelmät toimivat pääsääntöisesti odotusten mukaisesti, mutta eivät usein tarjoa vaadittavaa joustavuutta ongelmatilanteiden edessä.

Tämä case-tapaukseen pohjautuva kandidaatintyö esittelee työkalun Microsoft Dynamics AX -toiminnanohjausjärjestelmän multisite-implementaatioon, jonka avulla toimipaikkojen operatiiviset päälliköt pystyvät seuraamaan tuotannon yleistilannetta ja priorisoimaan työjonoaan osatoimitusten välttämiseksi.

PREFACE

I wish to thank my supervisor Erja Mustonen-Ollila for her inspiring attitude torwards scientific work and lighting-fast organizing skills. Writing this thesis has been a breeze thanks to her.

Also, I wish to thank operations manager Saku Antikainen and production coordinator Aleksi Torniainen of Isku Teollisuus Oy for answering tons of logistic and manufacturing related questions and guiding me through the company's processes.

Finally, thank you to production director Mika Tyrväinen of Isku Teollisuus Oy for authorizing my work and trusting me with developer privileges on the company ERP.

Lahti, April 15th, 2012

Heikki Korkala

CONTENTS

ABBREVIATIONS AND SYMBOLS

ActiveX	Microsoft's component object model components.
AOT	Application object tree A structure containing all Dynamics AX application objects.
AOS	Application object server The server containing business logic for Dynamics AX.
ATO	Assemble to Order Final product is assembled only when demand occurs.
BOM	Bill of Materials Materials required for manufacturing a product.
DAX	Dynamics AX Microsoft's implementation of an ERP.
ERP	Enterprise Resource Planning A software solution which takes care of a companys daily operations.
IDE	Integrated Development Environment
JIT	Just-In-Time A manufacturing philosophy where products are produced on order just when needed.
MorphX	Dynamics AX's integrated development environment
MRP	Material Requirements Planning Part of an enterprise resource planning software which schedules different production and logistic operations based on shipping dates.
MSDN	Microsoft Developer Network An informational website provided by Microsoft for various system developers and administrators.
RDBMS	Relational Database Management System A system used to store and retrieve relational data.
SQL	Structured Query Language A language used to query data from a relational database.
UML	Unified Modeling Language
Unit testing	Application module functional testing.
X++	Programming language Dynamics AX is developed with.

1 INTRODUCTION

1.1 The research problem

This candidate's thesis was done for a Finnish furniture manufacturer that has implemented a Microsoft Dynamics AX as an enterprise resource planning software in 2009. Implementation project has been going on to this day as new system features have been taken into active usage and old news have been refined according to business needs.

The implementation has proceeded to a stage where most of the base functionality of the system is being used on daily basis and usage skill is quite good. As the collective understanding of the new system has strengthened, a number of development needs has arisen to provide functionality for modelling the client company's own processes and methods of work. One of these is a need for synchronizing production between different production sites in the factory. Site synchronization in this context means that production sites should always prioritize their work so, that every site is manufacturing items from the same sales order for a customer and only sales orders for which all materials are available are released for manufacturing.

The company policy is that customer orders are delivered only as full deliveries i.e. part deliveries are not allowed. This poses a problem when multiple sites manufacture orders. A materials shortage in one of the sites participating in production means that all the already produced items must wait in the shipping terminal for the last products to be completed. This means that the terminal may easily fill up with partly produced sales orders that cannot be shipped. With the shipping terminal filling up, items must be replaced in other warehouses or the shipping terminal emptied with part deliveries. This of course raises warehousing and logistics costs.

The problem is formulated as follows: How to implement such an extension to the currently used enterprise resource planning system so, that production sites are working synchronously. Synchronization in this context means that from the multitude of sales orders to be delivered for a given date, the ones for which all materials are available for every participating production site are manufactured first.

In practice this means that when a sales order is broken down into production orders, all these production orders are made available for production only when the warehouses in assembly sites report that materials inventory is sufficient for producing every production

order in the current sales order.

1.2 Research methodology

The research methodology of this project consists of a literature review and an implementation documentation. The literature review provides an overview on information available on Dynamics AX, the development of the system and the related business problem.

The focus of this project is on the empirical implementation of the software artefact the objective of which is to provide a tool for production order prioritization and production site synchronization.

1.3 Framework of the thesis

This thesis concerns the software implementation of a tool to prioritize production order processing. Section 2 explains the problem in detail and provides starting points for the solution. Section 2.3 explains the technical environment in which the software is implemented. Section 2.4 regards the solution which has been selected for implementation.

Section 3 goes through the whole implementation process of the software. In section 3.1 the development paradigm of the software is presented and explained and in sections 3.4 through 3.5 the flow from design to implementation is explained. Section 3.4 takes a look on the testing model used in the project and section 3.5 goes through the deployment of the software to the production environment.

Section 4 analyzes the results of the work and section 4.1 lays the groundwork for future development of the extension.

1.4 Thesis goals and contribution

The objective of this candidate's thesis is to extend the client company's Microsoft Dynamics AX -ERP (Enterprise Resource Planning) solution by designing and implementing a production synchronization extension which enables factory work centers to prioritize

production in the case of material shortages [1]. The extension needs to provide transparency between production sites and is designed to be used by production coordinators and site managers. The sites in which this software is to be used are the company's final product assembly sites.

Even though the objective is to solve problems related to methods of production process - specifically, how production orders are prioritized in general - in the client company, this thesis focuses on the development and implementation of a software artefact [2]. The developed extension is restricted to view and logic development only. Data-layer implementation is not needed due to the possibility of using Dynamics AX:s system data objects and system class objects for this extension.

This thesis is a case-based applied research project with emphasis on designing and implementing a software artefact [2]. As the project focuses on the practicalities of the implementation of the software the impact on business is not analysed in great detail.

The project contributes to the coordination of the client company's production by creating a software tool which automatically prioritizes manufacturing of specific production orders based on the overall state of the production environment.

2 THEORETICAL BACKGROUND AND CONTEXT OF THE STUDY

2.1 Context of the study

The client's production environment consists of multiple sites all of which are responsible for producing certain product groups or materials needed by products. When the client's ERP receives a sales order, it is broken down into a production order tree where the root node is the final product, branch nodes are semi-finished products and leaf nodes are purchased materials which are converted into purchase orders as shown by figure 1. The MRP (Master Requirements Planning) functionality is then automatically activated to schedule all purchasing and production so that when production is started all materials purchases have been finished.

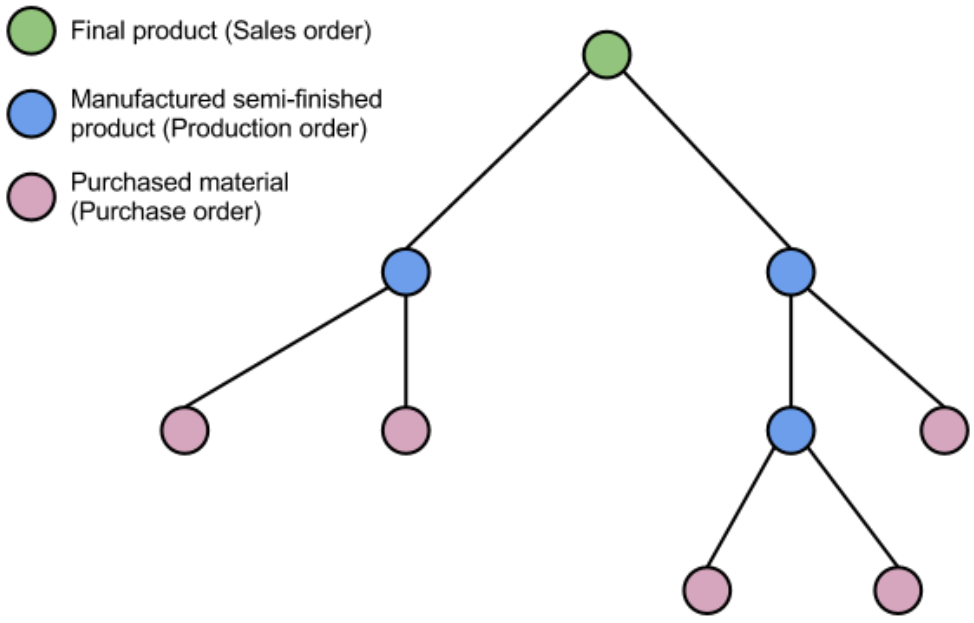

Figure 1. Sales order production and purchasing breakdown

The client company follows mostly ATO (Assemble to Order) JIT (Just-In-Time) philosophy where products are assembled just before customer demand [3]. Manufacturing may

be performed beforehand and varies depending on the type and size of a sales order. Production sites are independent production units and manufacture their production orders based on the MRP scheduling just before customer demand date to avoid unnecessary warehousing of finished products. The problem this thesis concerns with occurs when there is a material shortage in one of the sites participating in the production of a sales order as show by figure 2.

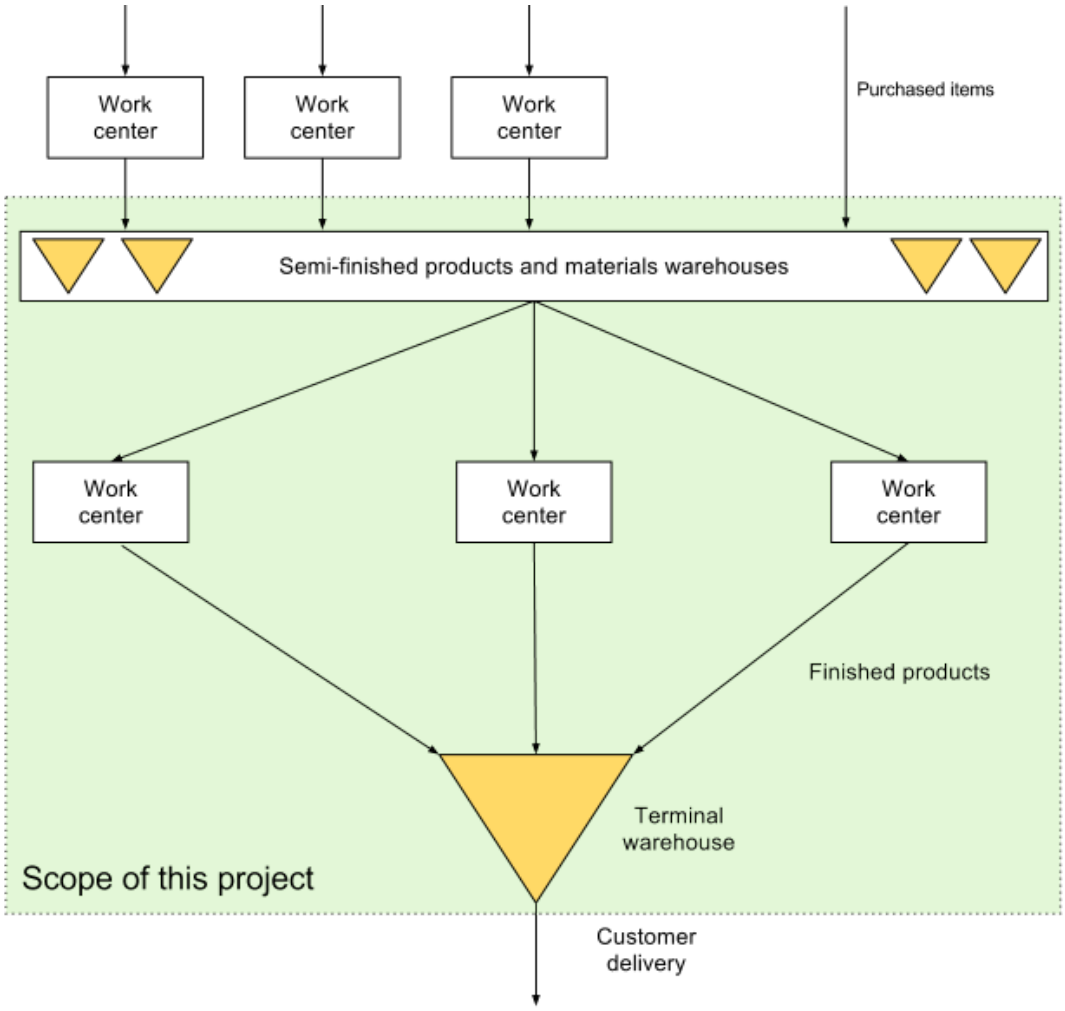

Figure 2. Material flow in the production environment and the scope of this project

The manufacturing and assembling of items in a single sales order is usually performed by multiple production sites. Let's say a customer orders a chair, a table and a coach. The sales order has a single demand date which must be satisfied by all participating sites, so all the items the customer ordered must be ready at the same time. Suppose one of these sites suffers from material shortage. The consequence is that all other sites produce items independent of one another i.e. don't prioritize orders based on the situation of other sites.

2.2 Literature review

This section provides an overview on the literature surrounding the subject from describing Dynamics AX ERP in general to development of Dynamics AX and logistics literature on the problem at hand.

2.2.1 Dynamics AX overview

For a generalized description of the system and the different modules the system provides *Dynamics AX: A Guide to Microsoft Axapta* describes Dynamics AX as Microsoft's take on the package business application market. One of the major advantages in Dynamics AX is thought to be the layering system which is described in this reports section 2.3.4. Dynamics AX is depicted as an easy to use ERP with seamless integration to other Microsoft products such as Microsoft Office. The book starts by explaining the installation and implementation of the system and goes on to describe the usage of the basic modules and their usage. The book also covers Dynamics AX's reporting and analysis systems, integration API's and Enterprise Portal which is a SharePoint system used to access business data and reporting. Also a short explanation on the development and available development tools of the system is done [1].

2.2.2 Development

Development books or other techincal manuals are quite scarce on Dynamics AX considering how big the actual system is. However a few books on the system have been published:

Microsoft Dynamics AX 2009 Programming: Getting Started is an entry level development book which begins by explaining the different application objects that are available or possible to create in the system. From there the book moves on to explain the X++ language features and available development tools in detail. Of the X++ language all the primitive data types, syntactic elements, control flow instructions, class objects and such are explained. After this the book discusses how data is stored and used in the system, how the user can interact with it using application objects and how to search for data programmatically. Data searching in Dynamics AX development is done in three ways: either by creating a query application object, creating a view application object or writing a select statement [4].

- Query application objects are structures created in the application object tree which define all the necessary data sources and filters used to output a dataset relevant to the search.
- Views are combinations of different data sources (tables) which are used to speed up querying of certain data.
- Select statements are the programming syntax for querying data in X++ code. Select statements resemble SQL (Structured Query Language) statements but support IntelliSense and syntax checking.

Dynamics AX Development cookbook collects a list of usable programming patterns and examples from different areas of Dynamics AX. For the purposes of this project, specifically chapters "Processing data", "Working with forms" and "Working with data in forms" provide useful examples on how to implement different features required by this project. The book is divided into recipes for creating generic feature and provides code examples with explanations and comments [5].

Inside Dynamics AX 2009 goes into the system architecture and development in detail. The book covers development from the architecture and performance point of view and creates an in-depth view on the system from a developer standpoint. This book explains the core concepts of developing the system with examples of different advanced development tasks. Also the book covers how the security and licensing frameworks of the systems work and how to use them. As an interesting side-note, the book covers the usage of Dynamics AX Business connector which is a way for using Dynamics AX application objects from a .NET based program [6].

Finally the book explains how to go through a code upgrade. When a project is in active development, each iteration of the development process outputs a deployable object. Code upgrade has to do with how the new deployable is actually deployed in a production system [6].

Best Practices for Microsoft Dynamics AX 2009 Development is a whitepaper listing all coding practises which are to be followed when developing an Dynamics AX 2009 X++ project. This project tries to conform to these practices [7].

Testing Guidance for Microsoft Dynamics AX 2009 is a whitepaper containing information on how Dynamics AX development projects should be tested. Testing in this context means unit testing and functional testing. This project implements both testing types.

A separate unit testing suite is developed and functional testing is performed as a user test [8].

Dynamics AX Developer Center is website part of MSDN (Microsoft Developer Network) which contains up-to-date information on how the system is to be developed. As an important feature, the website contains the whole system refrence manual from X++ language structures to system classes that can be used in system development. This project relies heavily on information contained in this site [9].

Agile & Iterative Development: A Manager's Guide explains the development model used in this project. Iterative development is a circular development type with short development phases which are handled as independent mini-projects. These mini-projects are deployed to the working system after each iteration of the development process. This model makes it easy to handle software projects where requirements' priority varies greatly depending on the current state of business [10].

2.2.3 Business problem

As for the business problem this project handles, a number of literature has been written but as this project is based on a industry specific case problem, the exact match in literature is hard to find.

As the explanation of the production model used in the client company *JIT production* provides an overview of the methods and processes in the company. JIT - also known as Toyota Production System - is a production philosophy orginally created by car manufacturer Toyota. JIT production philosophy's objective is to lower processing inventory and carrying costs. In practice this means, that products are manufactured just in time for the customer demand i.e. the production is started at such time that the product is to be delivered to the customer just as it is completed in production [3].

2.3 Technical background

The client's environment consists of Microsoft Dynamics AX enterprise resource planning solution in three tier installation [4]. The three tiers are:

1. Data Tier

2. Application Logic Tier

3. Presentation Tier

Data tier is the system where all application data is stored. In practice it is a cluster of servers running Microsoft SQL Server which is Microsoft's primary RDBMS (Relational Database Management System) for storing relational data. Application Logic Tier is a Windows Server System running Dynamics AX application object service. This service provides all business logic functionality and handles communication between clients and data tier. Finally the presentation layer consists of the actual user client installed for all user workstations [6].

Client installations provide a developer mode in which software developers can access and modify all application objects. Modification can be done declaratively by creating objects in to the AOT (Application Object Tree) by using the graphical interface or programmatically by creating a class object and programming the necessary logic into class methods. Class objects are programmed using the integrated MorphX development environment in Dynamics AX and using a programming language called X++ [4].

2.3.1 Application object tree

The AOT provides a number of different application objects for different purposes. Figure 3 presents how the tree is viewed in Dynamics AX. Application objects range from objects created for data storage to documentation for each of these object.

Data dictionary contains all the object necessary for Dynamics AX to store business data. Most important type of data dictionary object is table which represent an object type accessor to underlying SQL data. Tables defined in data dictionary are automatically created in the SQL database the system uses and can contain fields, relations to other tables and methods for e.g. input validation.

Macros are code modules which contain helper code for repetitive generic tasks. They can be used in any other code module to speed up development by not having to write boilerplate code.

Classes contains class object definitions. Class objects are objects as in object oriented programming paradigm which behave as blueprints for instantiated program objects in the

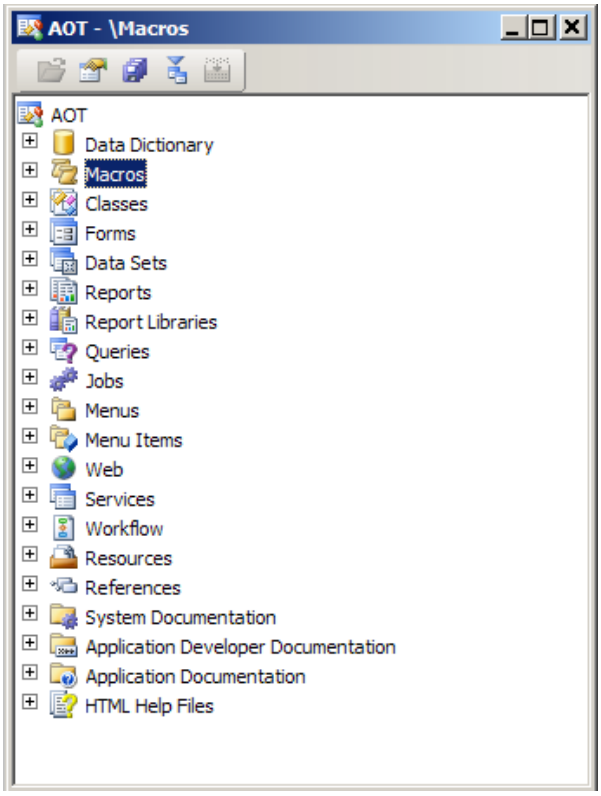

Figure 3. Application Object Tree viewer

system. The developer can decide whether to instantiate the objects on the client software or the application object server. When the class object must process large amounts of data, the object should be instantiated on the server for performance reasons. This reduces the amount of client-server roundtrips as the object can perform it's tasks by it's own and only return the data to the client when it is finished processing. Likewise a UI component class object should only be instantiated on the client to minimize network communication to the server, because the component is drawn and utilized solely on the client program [6].

For the use of this thesis we are interested in three different types of objects: Tables which are part of application data dictionary, Classes and Forms.

Data Dictionary tables are data objects that let the application access the underlying SQL database. This database contains all business data that is used by the system. We will be mainly interested in sales order and production order data. Classes are program object classes that implement application logic. We will use classes for implementing the functionality of the extension. Forms are objects which the ERP users can access to view and manipulate data. Forms usually present data in a datagrid and provide different command buttons, which after being invoked by the user call a method in a class object.

2.3.2 MorphX IDE

The MorphX development environment (figure 4) is activated by opening a class object or any other code object for editing. MorphX consists of a sidebar in which all the methods in scope are listed and a code window in which application code is written. MorphX uses supports code coloring and IntelliSense-like autocompletion functionality which automatically proposes methods based on context and scope. After a code module has been implemented MorphX automatically compiles the module on save and reports any errors or best practises deviations in a compilation log window [7].

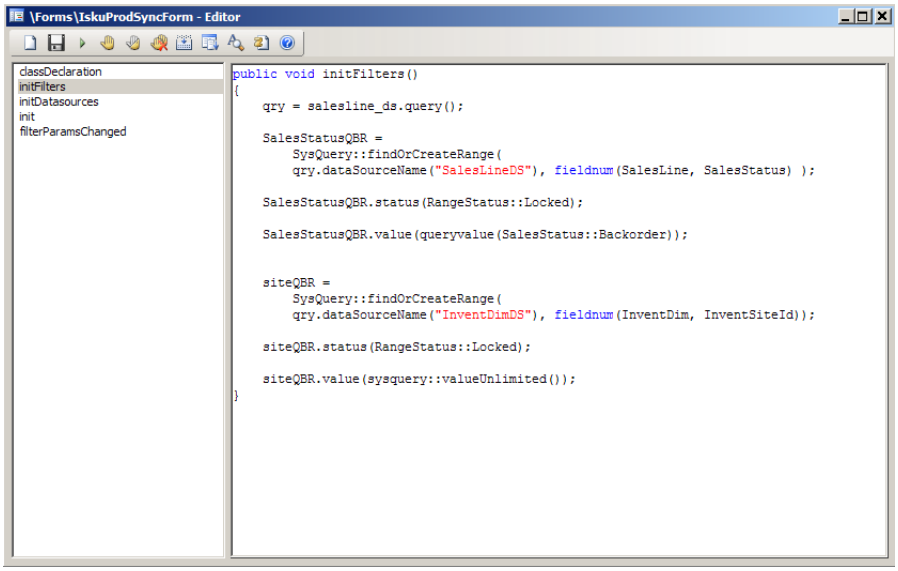

Figure 4. MorphX development environment

MorphX includes a project system in which all application objects are gathered or created for a development project. This way the developer does not have to search for different project application objects from the whole application object tree. Project system also provides an easy way to export the different objects to a transportation file for moving code between different deployments of Dynamics AX.

2.3.3 X++ language

X++ is an object oriented programming language developed for and used in Dynamics AX application development which syntactically resembles C++ or C#. The language provides all standard programming language flow control structures such as while and for loops, if-then-else constructs, object scope modifiers (public, private, etc.) and such.

The language is data-aware which means that it provides a mechanism for language integrated data querying and keywords for describing how the business data is to be fetched or updated. The compilation process is similar to .NET languages or Java becaus the programming language is first compiled into an intermediate bytecode format and stored separately. Dynamics AX runtime executes this bytecode when called from the highest application layer it is implemented. [6].

Differing from other "curly braces" type languages, X++ is case-insensitive, but development best practices dictate some rules on how upper- and lowercase nomination is to be used. For example camelCasing should be used for variable names and PascalCasing for type names [7] [6].

The type system of X++ contains value types which can be primitive or extended. Extended types are specializations of primitive types or enumerations which contain metadata for the type e.g. for presentation purposes. All variable declarations in an X++ method must appear before any actual code in methods and best practices dictate that a semicolon should follow the variable declaration block [6].

2.3.4 Layering system

Because ERP systems are usually developed by a number of different stakeholders, Dynamics AX provides a layering system for the application object tree modifications for different groups listed in table 1.

Table 1. Dynamics AX application customization layers

SYS	Standard application implementation
GLS	Region-specific modifications
HFX	Microsoft's patch and update layer
SL1/2/3	Local partner solutions
BUS	Business partner solutions
VAR	Value added reseller modifications
CUS	Customer administrator modifications
USR	End user modifications

Going through the layers from bottom-up the SYS layer provides the base functionality of the whole system. This layer cannot be modified in any way by users and it cannot be deleted. GLS layer is created for modifications to match legal demands in a specific

country where the system is to be implemented. HFX is a layer for Microsoft to provide system updates and patches. Layers SL1 to SL3 allow distributors to implement vertical solutions for the system. BUS layer is for business partners to create their own customizations. VAR layer is a layer for value added resellers who specialize in creating industry specific solutions. Finally the layers in which this project is implemented are CUS and USR layers. The actual development is done in USR layer which is the customization layer for end users. All system user can create their own customizations in this layer which can then be used by other users too. After development is complete, the product is moved on to CUS layer. CUS layer is for client administrators to provide in-house modifications to the system that are used by all system users [4].

Layering system works so that for every object a user tries to access in the application object tree the implementation which is highest on the layer stack. Say that a user tries to access a form in which no customizations have been made. The application would present the user with the form implementation of the SYS layer. Then, say a business partner makes a modification to this form. The modification is based on the implementation in SYS layer, but the modified version would be saved in BUS layer. Now when the user tries to access this form she is presented with the modified implementation in the BUS layer, which is - as previously stated - based on the SYS layer implementation of the object.

This layering system provides a way for making customizations in a safe manner. If a malicious user decides to delete all application objects in the system, he can probably access only the USR layer. Now the USR layer is wiped out but the functionality from CUS or lower application layers is used until the USR layer can be recovered. This of course eases development. If during development phase something goes terribly wrong in the application object tree, the modifications are only made in the USR layer. The layering system however doesn't protect data. If actual business data is deleted from the system, it would need to be recovered from backup databases.

2.4 Solution method

The client has requested that an extension to the ERP is to be designed and implemented which would enable production sites to synchronize their production in such way, that all sites would manufacture items belonging in the same salesorder. A list of extension requirements is viewable in appendix 1.

This is accomplished by designing and implementing a customization - an extension to the client's ERP system which enables production managers and coordinators to select which production orders should be processed next in the production schedule. The extension's target users are production managers of sites manufacturing finished products to be delivered to customers. Sites producing semi-finished products are not in the scope of this project.

3 DESIGN AND IMPLEMENTATION

This section documents the actual design and implementation phases of the project. We start by defining the development model to be used in the implementation of the project.

3.1 Selection of development paradigm

For this project a client-driven iterative development model was chosen as the development paradigm. In this development model the lifecycle of the extension is divided into iterations which are treated as projects of their own. The importance of implementing different requirements is based on the current insights of the client and may change considerably for the next iteration of the software. Iterative development allows small subsets of the required features to be implemented in a short timeframe to start the usage of the software as soon as possible [10]. Figure 5 visualizes the phases of the iterative model.

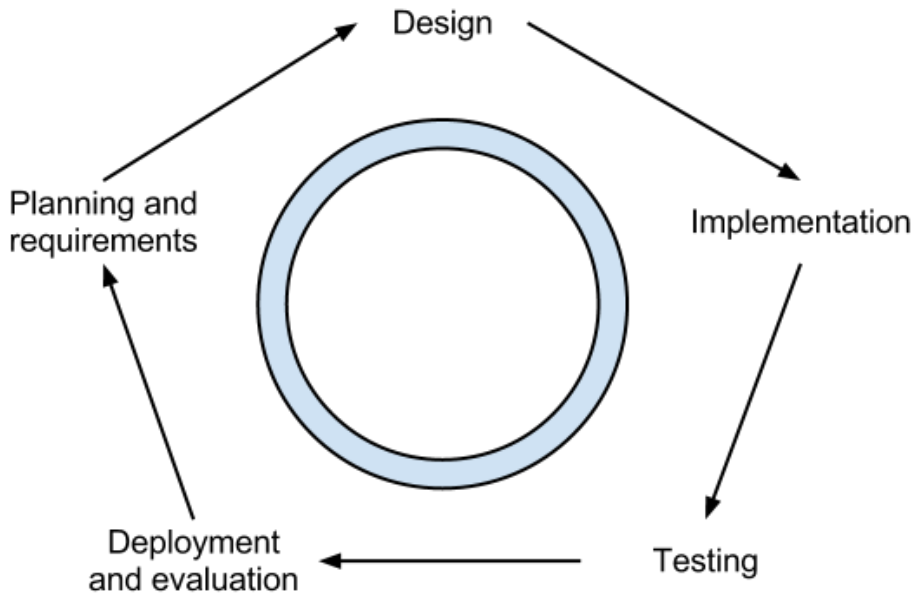

Figure 5. Iterative development model

First phase of the model or the entrypoint of the process is initial planning where the

concept of the software artefact is sketched. This consists of forming the concept of the tool which would solve a particular business problem - in this case how to synchronize production sites according to project objectives in section 1. The output of this phase is a list of requirements for the software artefact.

Second phase is analysis and design of the software according to previously defined requirements. In a small development project such as this we define the necessary application objects to be defined in the application object tree and UML (Unified Modeling Language) diagrams for a day-to-day use case and application concept.

Third phase is the actual implementation of the software which is documented in section 3.3. Implementation is to be done for a subset of requirements which the client has pointed out as most important.

Fourth phase is testing the software so that it behaves precisely as the implemented requirements dictate. This can be done with numerous different methods but in this project unit testing and user testing is utilized.

Final phase before the next development iteration is the evaluation of the software. In practice, for this project this phase means moving the solution from a test environment to the production environment and giving end-users the ability to use the software with real business data. Evaluation is done by the users and feedback is taken into account in the future iterations of the development model by adapting requirements to conform to the evaluation given.

3.2 Design

Design was started with a meeting with the different users of the extension. In this meeting a list of requirements for the extension was composed (appendix 1). Requirements were then prioritized based on time available and the necessity of the required functionality for the application. Requirements were then encapsulated and formed into implementation features.

3.3 Implementation

The first iteration of this project includes a business logic class object for processing the necessary data and a Dynamics AX form definition for presenting the data to the user as a grid of salesorders. This section explains how the user interface and application business logic was implemented in the project.

3.3.1 User interface

The user interface design was started by sketching an early prototype of the UI to paper with the client's representatives (Appendix 4). The client requested that the user interface should initially be as simple as possible with only the absolutely required functionality shown to ease the introduction of the software to users who are not technically oriented. More advanced users should be able to customize the UI similarly as other UI forms Dynamics AX provides. The user interface is depicted in figure 6.

The client's preference on the UI was a data grid -based form which would resemble other forms in the system as closely as possible. The form consists of basically three parts:

- A ribbon-style toolbar for presenting commands the user can give to the software such as printing a delivery card or releasing an order to produciton.
- A field for site filtering.
- A grid for data representation with tabs for different grid views.

A ribbon style command bar was selected due to the fact that the ribbon provides a way to provide a user with a multitude of commands but at the same time partition the command buttons in to sections for easy understanding.

The site filtering field is provided to allow to user to view the situation from the viewpoint of every assembly site on the factory and to show only data that is relevant to the selected site. The filtering behaves so, that if the selected site does not participate in the manufacturing of a certain open sales order, the sales order is not shown in the data grid. The filtering could have been automated determining the site from the user information in the system, but as production coordinators and site managers often manage multiple sites, a manual approach was selected.

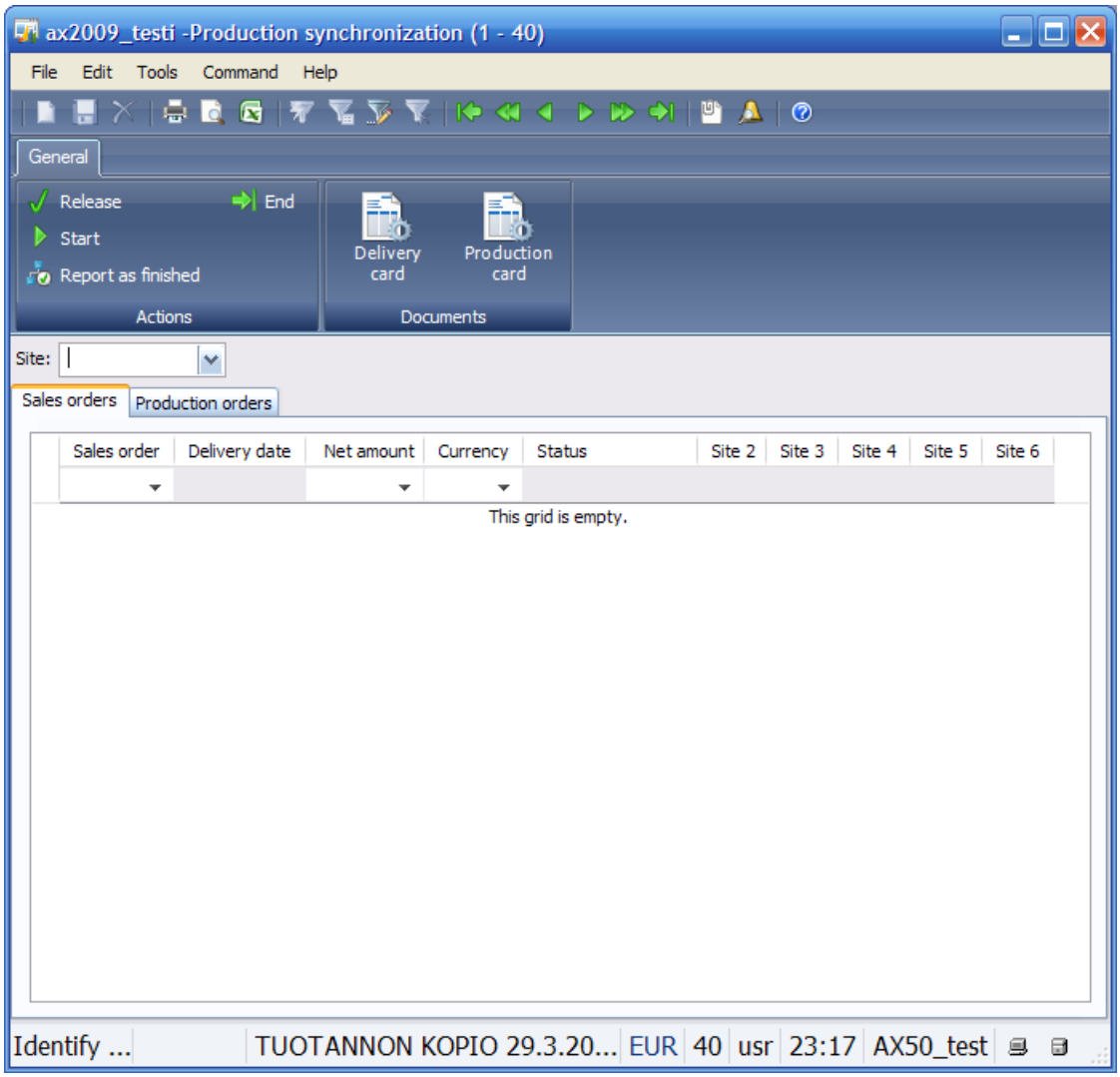

Figure 6. User interface of the software

The data grid contains the actual business data representation of the software. The main grid view displays all the open sales orders and their material availability on the factory. Sales orders are sorted by delivery date and the net amount in currency is displayed to give the user an idea of the criticality of the order. Status field gives a textual description on what action should be taken with the production order. If all materials are available on all sites the field contains "Ready for production". If materials are not available the field shows the text "Waiting for materials". The site fields are displayed to represent the status of individual assembly sites. The status is represented as different icons in the grid field. If the site does not participate the field will show as empty. Material shortage is represented as a red X. If the site is currently processing the order, a yellow exclamation mark is shown. Finally if the site has completed production on the sales order a green OK icon is displayed.

When a user selects a sales order from the grid by clicking it with the mouse, the system automatically queries the database for all production orders related to the sales order. When the user switches to the production orders tab, a grid containing the respective production orders are displayed. Production order grid consists of the production order number, item and quantity to be produced and status of production.

Note that on figure 6 the commands of the ribbon toolbar are not implemented on the first phase deployment of the software.

3.3.2 Application logic

The application logic is wrapped in a single class object which is accessed from the user interface form (class diagram in appendix 3). The objective of this class is to handle all time complex data processing tasks for the application. This helps separate the application presentation from the user interface and enables different views to be built for presenting the data to the user.

The class has two public methods:

- public bool CheckSiteParticipation(SiteID siteid_, SalesOrder salesorder_)
- public bool CheckSalesorderProdMaterials(SalesOrder salesorder_, SiteID siteid_)

The first method's (CheckSiteParticipation()) function is to check whether or not a pro-

duction site is participating into the production of a given sales order. This is done by checking all open sales orders in the system i.e. sales orders that have not been delivered yet. Each sales order has rows for items the customer has ordered and each of these sales order rows has a corresponding production order in the production orders list if the item concerned is manufactured in-house i.e. not a purchased item. The production order record defines which one of the production sites is responsible for it's manufacturing. If it is the site relayed as a parameter to the method, the site is considered to be partly responsible of the successful delivery of the sales order and the sales order is presented in the user interface.

The second method (CheckSalesorderProdMaterials()) checks whether or not a sales order is ready to be released into manufacturing. This is determined by retrieving all related sales order rows (the rows containing the actual sold items) from the system and retrieving their respective production orders. A production order links to a BOM (Bill of Materials) structure which lists all the materials and semi-finished products needed for completing the production order. For each item and it's required quantity in the BOM a corresponding amount is searched from the materials warehouse inventory. If a sufficient quantity of materials and semi-finished products are found the sales order is marked as ready for production for the current site. This check is performed for each assembly site in the factory and if all are marked as ready, then the user interfaces informs the user of this.

Due to heavy data processing, the class object for these checks is instantiated on the application object server to maximize the performance of the application. The data flow diagram is modelled in appendix 2.

3.4 Testing

Alongside actual functionality and business logic development a separate unit testing project was developed alongside the application. Dynamics AX's MorphX IDE provides an integrated testing facility for unit testing projects and suites. Testing is initiated by creating necessary testing classes for each tested business logic class and initiating unit testing from the system developer menu [11]. Tests are then run and if successful the system reports this in a testing log.

As the implemented application wasn't very large in scale, a separate testing project would not have been necessary for the first iteration of the project. However keeping in mind the future development needs and requirements changes, a unit testing modules were

developed alongside the actual application logic class objects.

Implemented unit tests are basically data-driven unit tests which verify application logic module output to the actual data in the database. Although this is discouraged in unit testing principles [8] it was a necessity because Dynamics AX's development environment lacks a proper mock data creation framework. After each modification to the application, the unit tests are rerun and if they are successful the application is ready for user testing.

For this project, integration testing was not needed because the functionality provided by the extension is encapsulated so that it does not currently modify any business data in the system. This means that the application does not interfere in any way with the standard behaviour of the system.

The extension also went through user testing where different users of the extension used it in the test environment. When all users reported they were happy with the performance and overall functionality, the deployment phase could be started.

3.5 Deployment

As the development work is completed on the client's test environment, which is a Dynamics AX system with a snapshot of the actual business data, and necessary unit and user tests have been passed successfully, the project containing all the application objects is moved to the production environment.

In practice this means using MorphX IDE's exporting tool to output the whole project as an .xpo file which is the imported in the production environment. Export and import tools have a way for defining on what application layer the objects should be deployed to. In this project the application objects were inserted into the USR layer of the production environment. After deployment it is necessary to create a way for the user to invoke the software. This is done by manipulating the application object tree by creating a menu item type application object which is then connected to the form object of the extension. This creates a menu choice which presents the application form to the user who selects the item. A separate menu entry is also created to the application object tree which ties the menu item created before to an already existing menu or a view in the system.

Thus the deployment is complete and the user is able to launch the software inside Dynamics AX. Afterwards an email message will be sent to all users to inform them of the

possibility of using the new functionality.

4 CONCLUSIONS

Extension projects to a modern ERP system is a straightforward task when implementing a software artefact that is properly encapsulated and separate from other functionality of the system. In Dynamics AX environment we can implement new functionality even though

Problems during design have to do with requirements specification in a project with a small time frame for phase completion. Requirements and priorities tend to change a lot during design and development and therefore finding a suitable middle ground of features to be implemented in a single phase can be a daunting task.

On the technical side, problems in Dynamics AX environment are mostly related to the help available on the development and customer customization of the system. Most literature on the system tend to be management and usage manuals, but actual development guidebooks or manuals are quite sparse. In-depth development training is provided by system providers but because this is costly and takes time, it wasn't considered in this project.

For some users of the system a separate training session for the extension may be necessary. This will be discussed with the client when the core user group is done with testing of the software.

At this phase, the software is only limited to a set of report-like functionalities and provides a view on the business data. The user is still required to use other constructs on the system to actually manipulate data. On following phases the software is required to make changes into the business data.

4.1 Future Work

After completion of the first development iteration of the project the software artefact has been deployed in production environment for user testing and comments. After a suitable amount of time these comments are then formed into new requirements and the requirements list is modified so that it better mirrors the actual business needs of the company.

Development is to be continued with follow-up development iterations in which new func-

tionality is once again added to the system according to the selected development model. As the extension currently works as a report-type solution for the current status of production and sales, it is probable that the next iteration will include functionality to actually modify business data based on user input. In practice this means that when a production manager selects a suitable production order set for processing, the extension provides functionality for him to free the orders for manufacturing personnel. This way the extension not only provides new information transforms but also works as a primary tool for production coordination.

Secondly, a manufacturing of the client depends on a number of paper documents which are transmitted alongside the product to be manufactured from raw materials to finished products. In the following development iterations the extension is required to provide a functionality for production staff to print these documents. Documents to be printed are delivery card and production card. Delivery card is a document that attached to the final product so that it will be sent into the right address. Production card lists the product's BOM and manufacturing steps required for a single manufacturing phase of a product. Printing these documents with the extension would save time on system usage and easy production management tasks.

The actual new features to be implemented in following iterations are defined by project management and is determined by the state of business needs with the feedback from this first phase evaluation taken into account.

REFERENCES

[1] Luis X. B. Mourão and David Weiner. *Dynamics AX: A Guide to Microsoft Axapta*. Apress, 2006. ISBN: 1-59059-489-4.

[2] Pertti Järvinen. *On Research Methods*. Opinpaja, 2004. ISBN: 951-97113-9-2.

[3] Cheng T. and Podolsky S. *Just-In-Time Manufacturing: An Introduction*. Chapman & Hall, 1996. ISBN: 978-0412735400.

[4] *Microsoft Dynamics AX Programming: Getting Started*. Packt Publishing, 2009. ISBN: 978-1-847197-30-6.

[5] *AX 2009 Development Cookbook*. Packt Publishing, 2009. ISBN: 978-1-847199-42-3.

[6] The Microsoft Dynamics AX Team. *Inside Microsoft Dynamics AX 2009*. Microsoft Press, 2nd edition, 2009. ISBN: 9780735626454.

[7] Best practices for microsoft dynamics ax 2009 development. Microsoft, 2010.

[8] Testing guidance for microsoft dynamics ax 2009. Microsoft, 2008.

[9] Dynamics ax 2009 developer center. WWW-page, URL: `http://msdn.microsoft.com/en-us/dynamics/ax`. Cited 22.4.2012.

[10] Craig Larman. *Agile & Iterative Development: A Manager's Guide*. Addison-Wesley Professional, 2004. ISBN: 0-13-111155-8.

[11] Dynamics ax 2009 unit test framework. WWW-page, URL: `http://msdn.microsoft.com/en-us/library/aa874515(v=ax.50).aspx`. Cited 22.4.2012.

Appendix 1. Requirements

Table A1.1. List of requirements

Requirement type	Date	Description	Development phase
Business	27.2.2012	Decrease warehousing and delivery costs by creating a tool for production order prioritization and production site synchronization	Phase 1
Business	27.2.2012	Shorten time needed for the selection of valid production orders to be released into production	Phase 1
Non-functional	27.2.2012	Tool is created in Dynamics AX ERP	Phase 1
Non-functional	27.2.2012	Real time calculation, no batch processing	Phase 1
Non-functional	27.2.2012	Available for production coordinators and site managers	Phase 1
Functional	27.2.2012	Provide a list of sales orders and the status of production and materials in all participating sites	Phase 1
Functional	27.2.2012	Provide a way to manually release all production orders of a sales order to manufacturing	Phase 2
Functional	27.2.2012	List production orders of a sales order	Phase 1
Functional	27.2.2012	Provide site filtering functionality for the sales order list	Phase 1
Functional	27.2.2012	Provide a functionality to automatically release production orders which are ready for production	Phase N
Functional	27.2.2012	Sort sales orders by delivery date	Phase 1
Functional	27.2.2012	Filter out large projects which are managed by hand	Phase 2
Functional	27.2.2012	Provide a list of missing materials of a sales/production order and estimates on delivery	Phase 2
Functional	27.2.2012	Provide a way to print production and delivery documents of a selected sales/production order	Phase 2
Functional	27.2.2012	Represent data in a grid	Phase 1

Appendix 3. Requirements

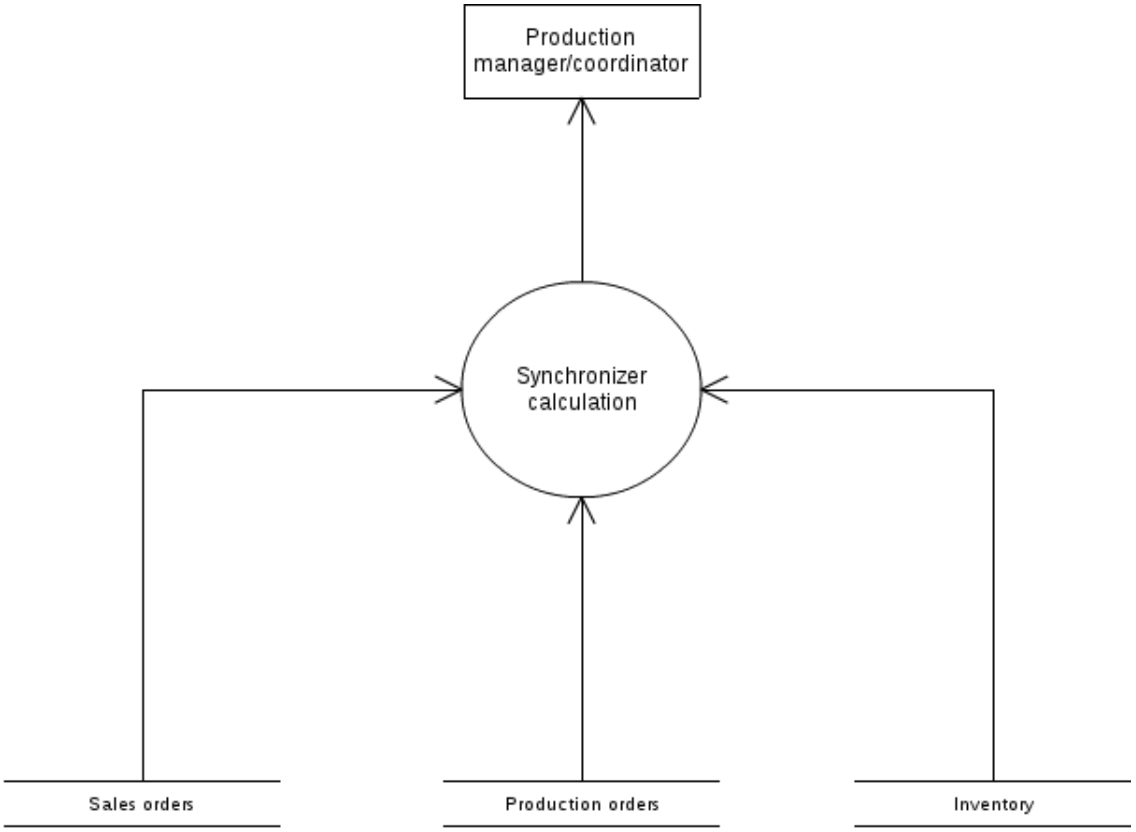

Figure A2.1. Data Flow Diagram

Appendix 3. Class diagram

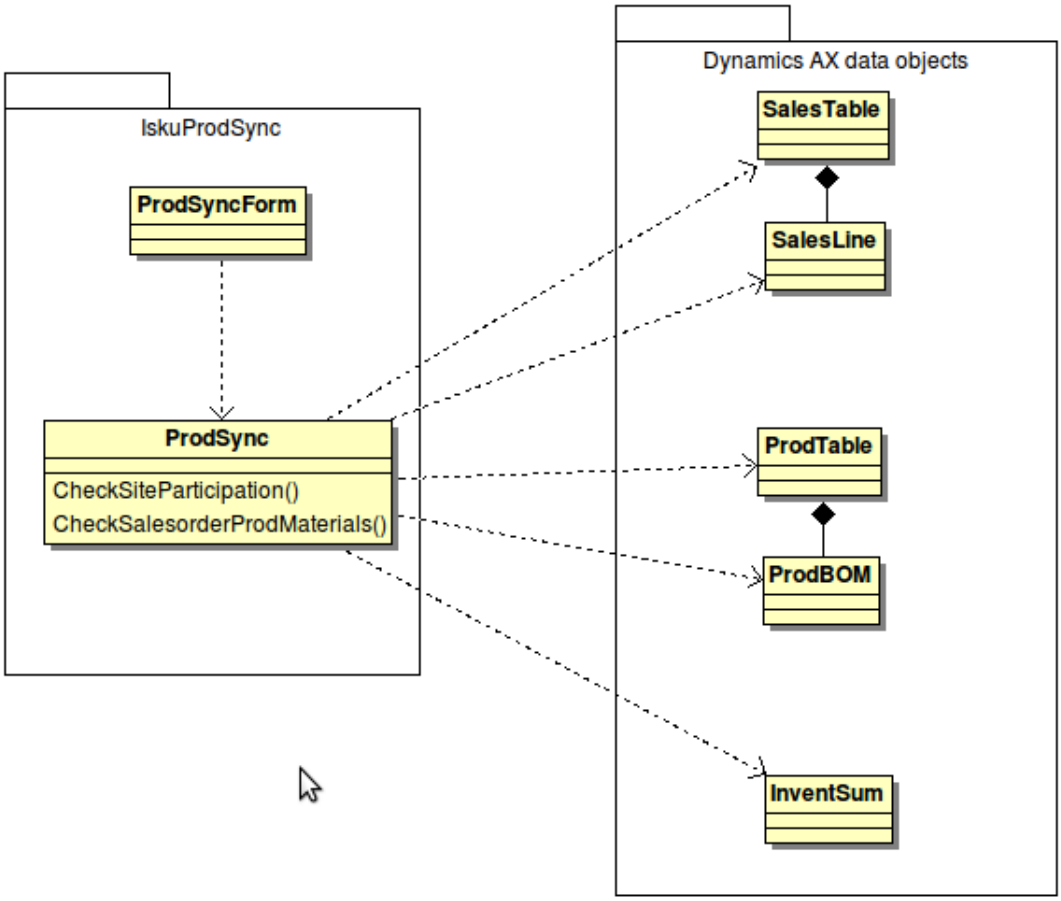

Figure A3.1. Simplified class diagram

Appendix 4. UI-sketch

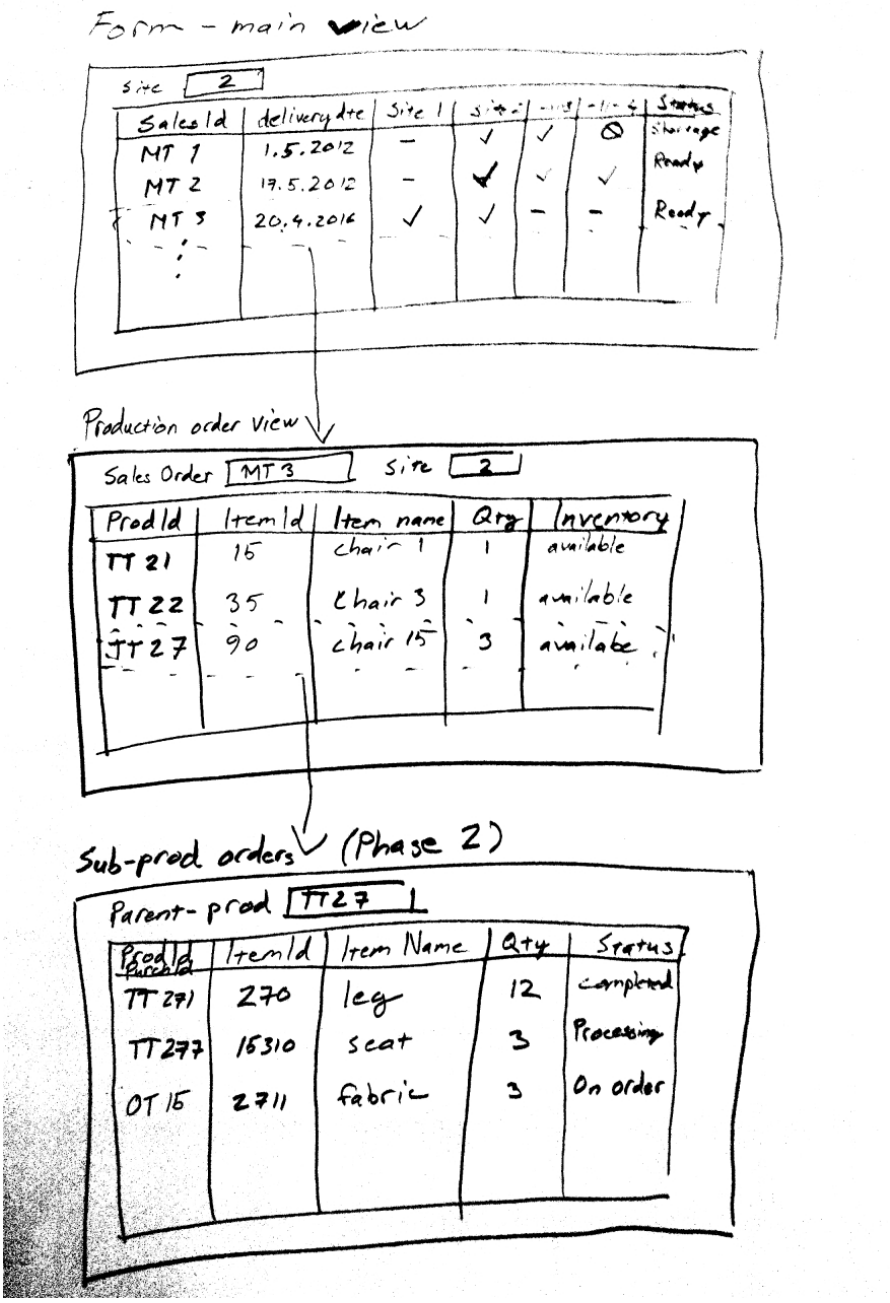

Figure A4.1. User interface sketch