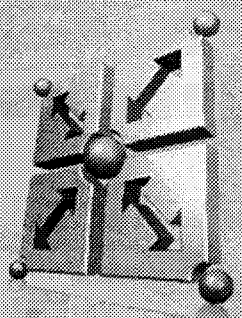
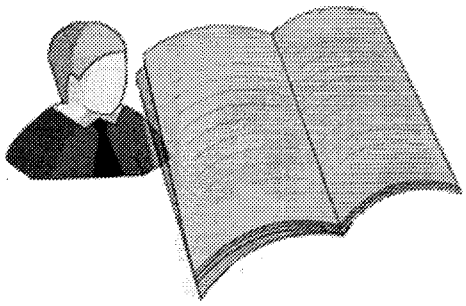




Converter DEM in Studio

Version 1.0

Olivier Fournier
Joan Alvado Cárcel

SCRIPTIE IC 2004 FOUR

How did we share the work?

PRESENTATION OF THE PROJECT	Joan Alvado Cárcel
Foreword	
Management Summary	
Structure of the thesis	
References	
INTRODUCTION	Olivier Fournier
1. Cordys Company	
2. Cordys product	
3. Cordys BCP	
PLAN	Joan Alvado Cárcel Olivier Fournier
FUNCTIONAL DESIGN	
1. Introduction	Joan Alvado Cárcel
2. DEM models in Baan IV	Olivier Fournier
3. BPM models in Studio	Joan Alvado Cárcel
4. Comparison DEM-Studio	Joan Alvado Cárcel Olivier Fournier
TECHNICAL DESIGN	
1. Introduction	Joan Alvado Cárcel
2. DEM data format: ASCII files	Olivier Fournier
3. BPM models in Studio	Joan Alvado Cárcel
4. Mapping	Joan Alvado Cárcel Olivier Fournier
5. UML and Architecture	Joan Alvado Cárcel Olivier Fournier
TECHNICAL REALIZATION	
1. Introduction	Joan Alvado Cárcel
2. Parser Method	Joan Alvado Cárcel
3. Modular Structure	Joan Alvado Cárcel
4. Template: XSLT Version file	Olivier Fournier
CONCLUSION	Olivier Fournier

Foreword

The purpose of this document is to provide a description of the current project to the people inside Cordys Company who are in relationship with the *"DEM Models in Studio"* project, as well as the broad of examiners from the Hogeschool van Utrecht. This description comprises all about our approach and understanding of the project, and is used also to define and approve its scope, planning and organization.

Management Summary

Cordys Enterprise is developing since last years *Cordys Studio,* a user-friendly environment which will be part of a more complete platform to allow collaboration between business partners, *Cordys BCP.* Most of the potential clients of Cordys BCP have already ERP systems, and an important part of them work with DEM models from Baan Sw.

In relation to this, this project provides a new functionality to Cordys Studio allowing clients to transfer their old DEM models into Studio BPM models. It makes Studio more competitive since it can offer to customers both investment protection and a comfortable migration of their models to the new tool (Studio). The new functionality will be integrated in the *"Content Transfer Utility"* from the *"Settings"* menu.

The scope will hold these main topics:

- Development and integration of a software application to convert DEM models into Studio BPM models
- Selection of models to be executed in Studio.
- Completion of these models to make them executable.
- Development of an audiovisual presentation to customers.
- Development of user –gu ides for users.

As an important issue, note that first version of Studio is expected to be on the market after the end of September 2004.

Structure of the thesis

About the thesis document, is important to explain how the document is structured to make it easier to understand. The chapters that we include are placed following the normal development of the project, i.e., chapters show the order in which the work was done. Thus, the major chapters of the thesis are:

I: Introduction: A small introduction to explain a few principles about Cordys company and its products.

II: Plan: A complete document to show all the preliminary study that was done to understand and define clearly the project.

III: Functional Design: In this step is offered a first approach to DEM and BPM graphic components.

IV: Technical Design: This step is composed of two parts: First, we did a further study of DEM and BPM models and the respective data formats used to represent them. Also all the design previous to the software realization is included here.

V: Technical Design: Explanations about the code that we use to implement the project.

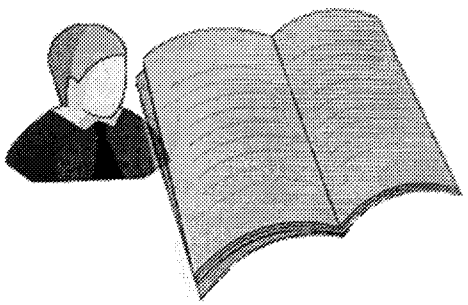
VI: Conclusions.

References

The following references have been used as input for this Release Project Plan. They are referenced in the following text as indicated below.

Reference	Description
Cordys website	www.cordys.com
Cordys White Paper	Help document about Cordys BCP.
Cordys Studio White Paper	Help document about Cordys Studio.
Cordys Framework (Appendix B)	Document about the organization in the company.
Project Management (Appendix C)	Document about the waterfall method that we apply for this project.
Cordys Studio Quick-Start Guide	Trainer guide for beginner about Cordys Studio.
World wide web consortium	www.w3.org

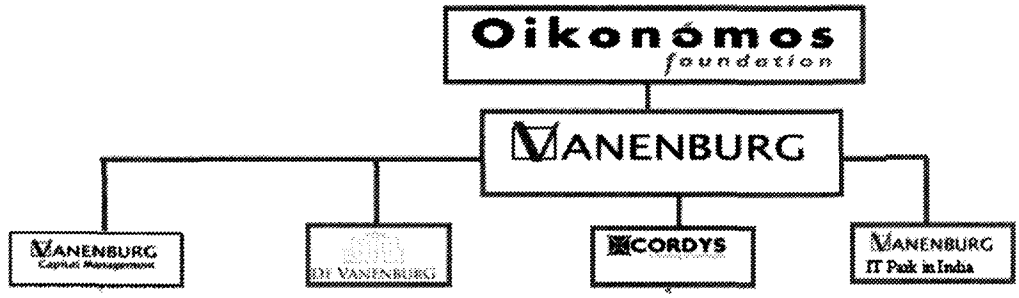
Contents



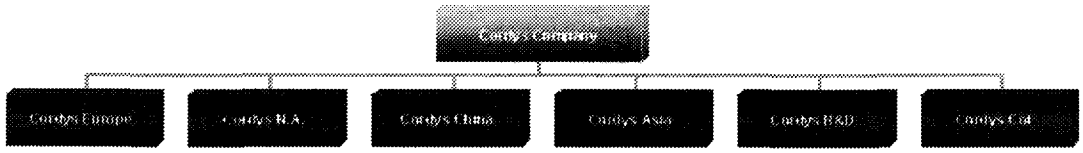
I Introduction

1.1 Cordys Company

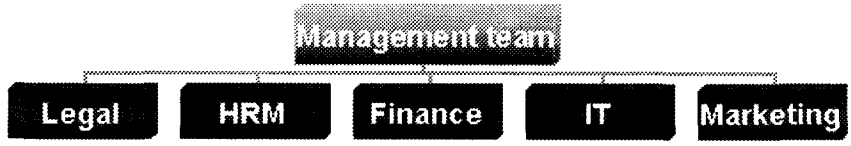
Headed by Jan Baan, Vanenburg Group, earlier called Baan Investments, was a major shareholder in ERP (enterprise resource planning) vendor Baan in the Netherlands, until it sold its stake to Invensys PLC in London in 2000 and later Baan was again sold, this time to SSA Global Technologies Inc. The Vanenburg Group, however, has moved on and has investments in several companies, including Cordys, through Vanenburg Capital Management. The company has more than 150 clients worldwide. The following diagram shows the structure of Vanenburg group.



Cordys Company is implanted in several continents and arms, like the following diagram:



The next diagram represents the structure of Cordys in the Netherlands. There are about 100 employees.



IT department is a part of Cordys Research and Development arm. The other part is situated in India.

During our training period we worked on BCP product, for Research and Development arm in the Netherlands.

1.2 Cordys product

Cordys is an enterprise that provides Real-Time application solutions based on Web Services that allow Enterprises to an easy definition, deploy and manage their business processes. Also collaboration and interactions among different Enterprises are a main focus of Cordys

activity. Cordys Real-Time business solutions are collectively known as the Cordys Business Framework, which comprises two different branches:

- **Technology**: Cordys BCP (Business Collaboration Platform)
- **Application**: Cordys B-Apps (Business Applications)

1.3 Cordys BCP

This project will be in relationship mainly with Cordys BCP technologic solution. Cordys BCP connects applications such as ERP, SCM (Supply Chain Management), CRM and Web services using open standards such as SOAP, XML, WSDL, and LDAP.

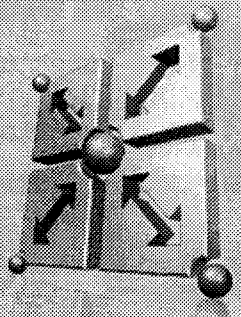
Customers can keep their existing applications, by means of creating an application connector. The platform takes care of infrastructure requirements such as coordinating the interactions between applications and services. It exposes enterprise applications, via the Internet, to external and internal business partners using WSDL, HTTP and SOAP, and can link to external Web services complying with the same standards, according to Cordys.

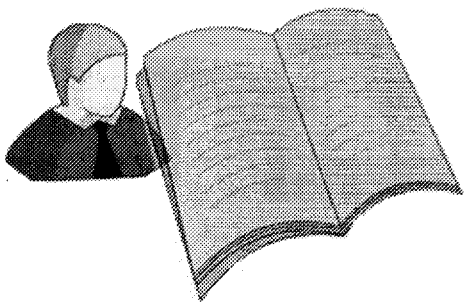
Cordys BCP platform comprises four main components:

- **Cordys Integrator:** Cordys Integrator enables all internal and external enterprise applications to become exposed as Web Services using industry standards (SOAP, XML, LDAP, WSDL). This includes both new systems that have been architected to comply with either a «.NET» or «J2EE» framework or existing (i.e., legacy) applications.
- **Cordys Orchestrator:** Cordys Orchestrator enables deployment, monitoring, execution and management of Real-Time Collaborative Business Processes.
- **Cordys Portal:** With Cordys Portal, users can have an access to performance indicators, analyze business intelligence, and use collaborative communication tools. So these are in effect three products that talk to each other in a Web services mode.
- **Cordys Business Application Connectors:** Cordys Business Application Connectors provide Web Services based access to a wide array of leading enterprise applications such as Baan, SAP, PeopleSoft, etc and enable their business logic, data and information to become part of overall real-time framework deployed on Cordys BCP.
- **Cordys Studio:** Cordys BCP provides a graphical environment, called Studio. It is a user-friendly environment running on the top of BCP that allow users to model, manage, and execute their business processes and collaborative applications. Studio includes a Business Model Repository, with three pre-configured types of business models:
    - Value Chain Model
    - Business Context Model
    - Business Process Model

An important part of the potential market for Cordys BCP is currently working with ERP, SCM, and CRM systems.

One of these ERP systems is Baan software; it uses DEM models to modeling the enterprises. To achieve the most competitive position in today's market, Cordys Studio should be able to allow clients to make an easy migration from their existing DEM models to Studio business models standards without the effort of re-modeling their business processes.





II Plan

II.1 Definitions and Abbreviations

Term	Definition
CA (Controlled Available)	The product is ready to be implemented live at the first customer(s). A special care period starts to support the implementation and ramp up the organization for GA. The MT delegate approves sales in this phase.
GA (general Available)	The product and the organization are ready as agreed. The product is orderable without restrictions other than known technical constraints (porting, backend connectivity, etc).

Abbreviation	Description
API	Application Programming Interface
BCM	Business Context Model
BCP	Business Collaboration Platform
BPM	Business Process Model
BPML	Business Process Modeling Language
BPMN	Business Process Modeling Notation
CRM	Customer Relation Management
DEM	Dynamic Enterprise Modeling
ERP	Enterprise Resources planning
HTTP	Hypertext Transfer Protocol
IT	Information Technology
J2EE	Java 2 Enterprise Edition
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
SCM	Supply Chain Management
SOAP	Simple Object Access Protocol
VCM	Value Chain Model
WSDL	Web Services Description Language
XML	Extensible Mark-up Language
.NET	Microsoft e-commerce platform

II.2 Definition of the problem

Cordys Studio contains an advanced business-modeling component. With this component business processes can be modeled up to an executable model. For new users it is important that they can directly adapt their systems to the Studio environment from an existing library of models. Instead of reinventing business and process models they can make a quick start by adapting existing models instead of defining complete new models. This highly leverages previously made investments, especially for Baan users who are currently using business processes in DEM format.

II.3 Project assignment

The main purpose of this project is to add a concrete functionality to Cordys Studio tool. This functionality allows Studio DEM users to convert their old DEM models into executable Studio models. It makes Studio more competitive since is easier for new customers to adapt their IT systems to Studio environment without re-modeling their business processes. It is an investment protection for the customers.

Also, Knowledge Transfer must take place for the users. For this, user guides will be developed on how to import DEM models in Studio, and make them executable. An audiovisual demo must be done as well, to show clients how the converted models are executed.

The objectives of this project are:

1) The creation of a general program that is able to convert DEM models into Studio BPM models.
2) The adaptation of a library of models in order to obtain executable BPM models.
3) The creation of an audiovisual presentation for customers to show them how their DEM models can be executed in Studio.

Some sources of models are existing models in ERP systems such as Baan. By building a converter these existing DEM models can be converted.

The Studio standard functionality is bigger than the previous DEM functionality. After the DEM models are converted, they need to be completed in Studio before they can be made executable.

II.4 Scope of the project

II.4.1 Included

- Realization of converter software to translate DEM data formats into XML files.
- Execution of different example models under this new software.
- Integration of the new software in Studio environment.
- Enhance the example models in Cordys Studio to make it executable.
- Internal documentation about the project must be presented to the Supervisors of the project, as well as to the people from Cordys Studio team. This documentation consists mainly in Design Documents with details about the source code.
- A user guide about how to import DEM models must be done for customers.

- A user guide about how to make imported DEM models executable must be done for customers.
- A visual demonstration for customers must be developed.
- Final thesis report to be delivered to the Hogeschool van Utrecht.

II.4.2 Not included

- Creating the DEM models examples to be converted, since the company provides a repository of DEM models.
- Development of a user guide for the converter software is not necessary since the program has almost no user interface. So what we shall develop is the normal documentation about the project (internal), but no any external quick-start guide for customers in this first phase. Only a small description of the new functionality for Studio will be develop to be integrated in the Studio Documentation.

II.5 Results to be delivered

- Converter Software to convert DEM models to Studio BPMN models.
- Executable Reference Models of Baan on Cordys Studio.
- Reference Model Documentation: Presentation, Internal Documentation and users guide.

II.6 Essential pre-conditions

II.6.1 Technical

- The converter software developed does not need to be Multi-byte compatible for the moment, because this aspect is not being considered for the Studio implementation. It can be added afterwards without incompatibilities, but is outside the scope of this project.
- The software has to be developed according to the CORDYS code standards.
- The software must be compatible with both Windows and Unix Systems, since DEM models are available for both of them. Then, the Converter Program should be able to recognize Windows and Unix return line character from the ASCII files that are representing the DEM models.
- Developed code needs to be clear and unambiguous.

II.6.2 Equipment

- Two laptops will be provided for the IT department inside the company for the realization of the project.
- Internet and internal network (LAN) have to be available for these laptops during the project, as well as the required software (Baan IV, Cordys BCP, etc...). Help in this aspect will come from Cordys Studio team as well as from the IT department.
- A server to use Workflow API and Baan must be also provided.

II.6.3 Human resources

- The project team is composed of two members.
- Personal training on different product or technology such as Cordys BCP, Cordys Studio, Workflow API, SOAP, etc.

II.6.4 Policy

- The project should be developed in six months (starting date: 02/02/2004).
- The realization of the converter software in Cordys Studio is planned to be ready in May. This is related with the first step of our project (Development of a Converter Program).

II.6.5 Software version

- For the graphical components conversion the studied version of Baan Sw is Enterprise Modeler Baan IVc.
- After the conversion, in the third phase of the project, the study will be focused on Baan 5.0c, to take into consideration the addition of extra functionality to the DEM models of Baan 5.0c.
- The version of Cordys Studio for this project is Studio CA

II.6.6 Methodical

- The project will be developed according to the principles of Waterfall Method (See *"Methods, Techniques and Standard"* point, as well as *"References"*).
- The system will be developed according to UML design.

II.7 Success and Risk factor

II.7.1 Success factor

- The project members must be in a close collaboration with the members of Cordys Studio team, especially for the first phase of the project
- The project members must be in collaboration with Henk Ten Voorde for the second and third phases of the project.

II.7.2 Risk factor

Risk Level	Details	Actions to avoid or minimize risk
Loss of information from the original models.	Keeping significant DEM components out of the scope as a wrong decision can imply a significant lose of information from the original model. It can be really detrimental for the customer.	Include Annotations in Studio for any of these components to explain how each functionality can be added.
License of reference models	To be able to work with DEM models in Cordys Studio a Copyright is necessary. So Studio can offer customers the possibility to transfer their already existing DEM models into Studio BPMN models, but cannot offer new models based on DEM to customers without buying a license. Legal problems can be found here	The company has shared Baan licenses with SSA/Baan.
Workflow API Baan finished in time	The project needs Workflow API to execute the third part (make models executable). A delay in the development of Workflow API can imply a delay for the project.	Be always informed about the Workflow API project advance.
DEM models are not designed for execution	The objective is executed DEM model after the conversion on Cordys studio, but DEM models are not designed for execution.	Taking into consideration every component and annotation of DEM models to complete manually the BPM models before execution.

II.8 Activities to be done

The project is structured in six main stages. First, a complete investigation about Studio and Baan, and how it fits in Cordys environment has to be done in order to obtain a good project understanding. After, a program to convert DEM models, stored as ASCII files, into BPM models, stored as XML files, must be developed. Once the program will be internally tested and accepted, the next phase of the project consists in the selection of a few models from an existing library to run them under the converter software. Finally, after the conversion of these models, the obtained BPM models have to be completed with some functionality from Studio to make them executable.

1) Project Investigation
2) Development of a Data Conversion Program
3) Integration of the software in Cordys Studio
4) Content Preparation
5) Completing the Models
6) Developing Education Part

II.9 Methods, techniques, standards

In a general view, this project will be developed according to the classic Waterfall method, which is usually the methodology used in Cordys (Appendix A). The method will be applied in different stages of the project, depending on each stage. Picture below shows how Cordys is used to apply the standard phases of Waterfall:

- **PI:** Project Investigation
- **TR:** Technical Realization
- **IT:** Internal Test
- **ST:** System Test
- **AT:** Acceptance Test

Of course some deviations are going to be made over this general steps.

UML techniques will be applied during the technical design to develop Java Classes and present a group of models.

The technical realization in Java will be developed according to the Cordys standard packages.

About testing, in this project we will do two different types of tests: internal tests and acceptance tests.

II.9.1 Internal test

The way to do the internal tests usually starts by sending a description or plan to the correspondent supervisors describing how the test is going to be done. After receiving an approval, the two developers of the project as specified will do the tests, and the results will be sent to the adequate responsible person in each case, to obtain approval.

II.9.2 Acceptance test

After receipt of the approval for the internal test, new tests will be done in sessions by the two developers of the project, and supervisors. A document with description and plan of the tests is not necessary in this case, since the supervisors will be provided directly the new issues to test. After the acceptance test has been approved, a meeting with more people from the company can be organized as well to show the results of these tests, if requested.

II.10 Quality management

At this point, it is important to explain that there are two main outputs for this project. One is the program converter, called *Import DEM Model,* which is developed in the first phase of the project as part of the *"content transfer utility"*. The other is an audiovisual presentation for customers to show how their DEM models can be executed in Studio. The rest of the phases complete the project with demonstrations about how the converted models are executable. The converter program is what is going to be implemented in the end tool, while the audiovisual presentation is highly necessary to make the product attractive for customers.

The quality of the converter program will be based in two factors. First, a correct translation of each imported component from DEM models to BPM models must be achieved. But also a strong and consistent integration of the software as part of Cordys Studio is important as quality indicator, which will be able to do an efficient task importing DEM models for customers.

This quality will be guaranteed with internal and acceptance tests, but with the steps about select and enhance the models in Studio to make them executable. These steps are a study to demonstrate that the new models we obtain after the conversion can be executed in Cordys Studio successfully.

Internal test for the first phase will be done with members of Cordys Studio team (Research and Development Department).

The main tests that will be done during the project are the following:

- **1st test**: After the technical realization of the program converter. Internal and acceptance test will be done as explained in the previous point. Quality here is mainly based in the right correspondences between input and output models. Expected to start on April the 28th, lasting 64 hours. (H. Rietveld).
- **2nd test**: Internal and Acceptance test that will be done after integration of program converter. Expected to start on May the 25th, lasting 24 hours. (Henk Rietveld).

- **3rd test**: After make the converted models executable. In this case, workflows obtained determine the quality of the execution. Expected to start on July the 8th, lasting 24 hours.

II.11 Project organization

Next table offer a view about which persons are related to the project, and which functions do they have:

Name	Role in project	Function	Organization
Olivier Fournier	Developer	Student Project	Development (Cordys Studio Team)
Joan Alvado Cárcel	Developer	Student Project	Development (Cordys Studio Team)
Wilfried Rijsemus	Supervisor	Program Manager	Program Management (Cordys)
Gerrit Spronk	Student Supervisor	Responsible of Students	Hogeschool van Utrecht
Henk Rietveld	Technical Support	Product Manager	Development (Cordys Studio Team)
Erwin Nooteboom	Technical Support	Software Architect	Development (Cordys Studio Team)
Henk ten Voorde	Project Owner	Program Manager	Program Management (Cordys)

II.12 Relationship with other projects

This software is going to be integrated inside Cordys Studio 1.0 GA version. This project adds functionality to Cordys Studio. It should be compatible in all the aspects with the existing tool.

The integration in Studio 1.0 is also part of the project. It will be tentatively a new option in the **"Content Transfer Utility"** window. This option could be in the browser UI to allow the user to select a DEM file to import into Cordys Studio.

Another issue to take into consideration is the fact that during the third phase of the project we will use a new tool, which is currently being developed in Cordys, **Workflow API.** This tool provides Studio an API to access Baan systems and methods to communicate with a remote Baan system. So, the project has a dependency of the final realization of **Workflow API.** Nevertheless, this dependency is not expected to have a delay for the current project, since **Workflow API** is not going to be used until the third phase of the project, which will be as much early on May. The **Workflow API** is planned to be ready by the end of March.

II.13 Communication

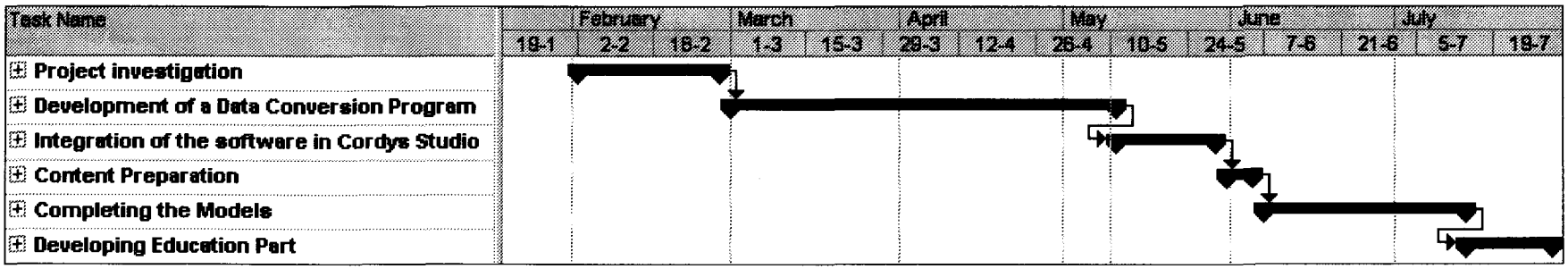
- Progress Report will be sent to the project supervisors every two weeks (Tuesday).
- Meeting with People from Cordys Studio team each time it is necessary. Reports of these meetings will be realized and send to them when required, in order to keep an open communication.
- Internal network with the current development of each project inside the company will be available. Here is a link to our project that we have to fill each two weeks to keep it updated. This information will be used as progress report for the project, and everybody from the company can consult this information.
- Individual phones, e-mail and msn messenger will be available for the communication between the members of the project and any other necessary person of the company.

II.14 Estimate of the project cost

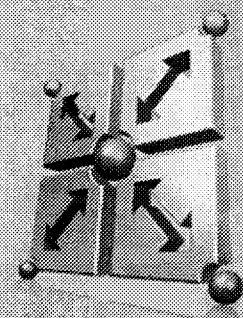
- 1000 hours will be needed for the two members of the group for the realization of this project.
- From other persons related to the project, like the technical support people from Cordys Studio or the Supervisors, between one and two weekly hours can be needed. That means a total cost of 36 (average between 24 and 48) hours per each of them.

II.15 Planning

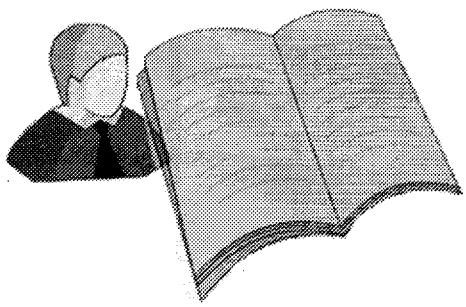
- **Project Start Date:** Thu 26-2-04
- **Project Finish Date:** Fri 30-7-04



Notice: In appendices D, you can get more details about the planning (It is a chart planning).



III Functional design



III.1 Introduction

This was the first phase of the project after the understanding of the project and the realization and approval of the project plan. To be able to do a good converter software, first we start by study in detail both systems, DEM models in Baan and BPM models in Studio, from a high level view.

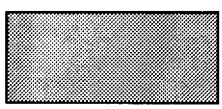
Thus, the technical design is structured in three main steps. First, a study of all the graphic components contained in DEM models and their properties. After, the same study has been done for BPM models in Studio, analyzing all the graphical components, as well as some special structures, which are the groups. Finally, we finish the functional design by establishing a comparison between the components of both systems based on what we have studied in the two previous steps.

III.2 DEM model in Baan IV

III.2.1 Introduction

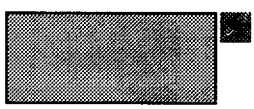
In this document is offered a general overview of all the representative graphic components from Baan DEM software, concretely Baan IVc version, although some of them will not be taken into consideration for the translation into Studio BPM models.

III.2.2 Types of activity



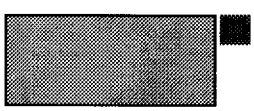
Activity

An activity box represents a manual or automated process applied to an item.



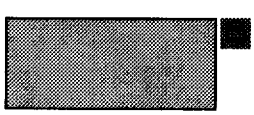
Baan Session Activity

This draw show that the activity linked requires calling a Baan session.



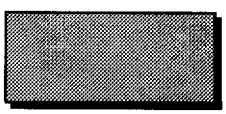
Manual Activity

The icon linked to this activity requires a manual action for its execution.



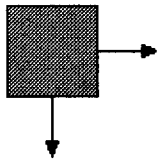
Other application

This type of activity is used to determine a link with another application (other vendors). This component will be out of the scope for the conversion.



Business Process

This icon represents a nested business process for the activity shaded. That means that this activity is developed in another diagram with sub-processes on a lower level.



Control Activity

This component represents workflow points when a decision has to be made. An arrow is split up in two or more arrows representing the different flows to be followed depending on the taken decision (Xor, And, Or).

III.2.3 Types of state

In each diagram you have to define two of the states as the **Begin State** and **End State**. Thus, the first state of a DEM model is always the **Begin State** and the last must be the **End State.** The following pictures show the icons for these states.

Begin State

This state represents the start of a (sub) process.

End State

This icon represents the end of a (sub) process.

State

A state represents a static point in which a concrete item is. The flow of a DEM model is always state-activity-state, so states represent static characteristics for an object that has to be processed or has been processed. Here is important to remark that DEM models are constructed according to the petri-net algorithm.

III.2.4 Types of link

Relationship

Arrows represent Relationships between the different components of a diagram. These are directed arrows that show not only the connections between components, but also the direction of the flows within a process. A relationship can have a condition.

Join

Sometimes two relationships can converge in a same component. Two flows become one component. This is called a join.

III.2.5 Type of event

In other way, sometimes is necessary to define and associate an event to an activity or a state. Thus, there are four types of events, represented for the following icons:

User Event

The actor assigned to the activity in the process must perform an action.

External Event

An external input is required for the linked activity.

Timer Event

The activity is executed each time the timer goes off.

Automatic Event

An automatic activity is executed for a set of workflow objects, that is, a container.

Nevertheless, events are considered outside the scope for the conversion of graphic DEM model components, because they are manually added in DEM models.

III.2.6 Other components

Finally, more information can be visualized from a DEM model with another three icons, as is described below:

Text

A description has been added to this activity.

Utility

A utility has been linked to this activity. A utility is a group of display and/or print sessions. This component is considered out of the scope for the conversion.

Role

One or more roles have been linked to the related activity (by project model).

<free text> **Free text**

Allows adding some description about the business process model.

Another interesting point to take into account for the conversion is the fact that different diagrams can be grouped in a **Project,** to define a higher unit of work. Common issues to both of them such as roles, properties, etc... can be assigned directly to projects (i.e. to all the components of a project).

III.2.7 List of Properties

Here is presented an overview about which concrete properties are going to be taken into consideration for the conversion. The properties are structured in six groups, which are **Business Process Activity, State, Relationship, Business Process, Version** and **Project,** as is showed below:

III.2.7.1 Business process activity

Below you can see the properties that will be used in the conversion.

- **Activity type**: With this property is possible to fix the kind of the selected activity, choosing one of four the following five types, which are already explained:
  - Baan Session
  - Manual Activity
  - Business Process
  - Control Activity
  - Other Application
- **Control type**: This property fixes the logic for decisions in a control activity.
  - Xor
  - Or
  - And
- **Code**: External identifier of the activity.
- **Activity description**: General description of the activity.
- **Work instruction**: Property to show is there's any comment to include any additional instruction to the activity, as text.

III.2.7.2 State

- **Description**: General Description of the state.
- **State text**: Additional text that can be added to a state in order to provide more information.
- **Begin and End**: These properties establish if the selected state is the Begin State or End State, respectively.

III.2.7.3 Relationship

- **Condition**: Only if the selected relationship is placed after a control activity, you can define a linked condition for this flow.

- **Description**: General Description about the condition. It can not be referred to a relationship without condition.

III.2.7.4 Business process

- **Business process name**: Name or code of the business process
- **Version**: Version linked to the business process.
- **Description**: General description.
- **Generation date**: Creation date of the business process.
- **Owner**: User that create the business process
- **Last change**: Date of the last update.
- **User last change**: User that made the last update:
- **Work instruction**: As with an activity, a comment can be added to a business process in order to provide more information.

III.2.7.5 Version

- **Code**: Identifier for a version.
- **Description**: General description of the version.
- **Derived from version**: With this property you can indicate that the current version comes from an older version.
- **Text**: Additional text added to provide more information about the version.
- **Effective date**: Creation date of the version.

III.2.7.6 Project

- **Project model**: Name or code of the project.
- **Description**: General description about the project.
- **Version**: Version linked to the project.
- **Text**: Additional text that can be added to provide more information about the version.
- **Generation Date**: Creation date of the project.
- **Owner**: User that create the project.

III.3 BPM in Cordys Studio

III.3.1 BPM Components

The main difference between DEM models and Business Process Models in Studio is that the state components from DEM diagrams are not present in the BPM diagrams. This is because while DEM models are based on the petri-net algorithm, BPM models are based on the BPMN standard language. BPMN (**Business Process Modeling Notation**) specification provides a graphical notation for expressing business processes in a Business Process Diagram (BPD). The BPMN specification also provides a binding between the notation's graphical elements and the constructs of block-structured process execution languages, including BPML and BPEL4WS.

Thus, below are shown the five main graphic components in BPM diagrams.

Start State

Every process diagram contains a start state - the place where the execution begins. There must be exactly one start state.

Activity

The activity represents an elementary work - it can be an invocation of a web service (from the CAS repository) to retrieve information from the back-end system or write them back, or a manual task invoking a URL. If the activity is manual job then also a role has to be assigned to it.

Decision

A Decision in a business process is a way of resolving a split. Every Decision has at least one incoming and at least two outgoing connectors. Each outgoing connector has a description and condition associated with it. At runtime, that outgoing Connector is followed which associated condition is true. Only the default outgoing Connector can be left with no condition associated, and it is followed at runtime if all other outgoing Connectors' conditions evaluate to false.

Connector

Arrows represent Relationships between the different components of a diagram. These are directed arrows that show not only the connections between components, but also the direction of the flows within a process. Connector placed after a decision can define special properties, such as condition.

End State

The execution of the process stops in this End state. There can be more than one end state in the diagram if the flow is branching at some point to two or more flows.

It is possible to include links between different diagrams as well. This is achieved with a special component, sub-process.

Sub-process

Sub-process diagram refers to another diagram, it is also an invoking, but not as the web service - the invoking is made as a function or procedure. If we have complex process diagrams there maybe some parts are the same so we model them only once and then refer to these parts as sub-processes.

III.3.2 Events

In BPM there are five different types of events, which are the following:

Intermediate message

If an intermediate event is waiting for an input message, the trigger type is 'Message'. If an intermediate event with trigger type 'Message' is drawn after a Decision, that Decision is translated to the BPML choice. This means, at runtime, the outgoing Connector is selected depending on the message type.

Delay

If an intermediate event includes a delay, e.g. wait 7 days after a quotation is sent, then its type is 'Delay'.

Exception

It is possible to define one or more Exception events for an Activity or sub-process. The Exception event is fired if the Activity generates an exception (example a SOAP processor error). You can also specify the error code (the error code that is thrown up by the SOAP fault message) for which the Exception event should be fired. If no error code is specified, the Exception event will be fired for all errors.

Time Out

It is possible to define a Timeout event for an Activity or sub-process. The Timeout event is fired when the Activity or sub-process is not executed within the specified time interval.

Compensate

It is possible to define a Compensate event for an Activity or sub-process. The Compensate event is fired if the Activity is rolled back. An Activity is rolled back if the total process needs to be rolled back.

III.3.3 Annotation

It is possible as well to add extra information to a BPM component by linking an annotation. In Studio there are two kinds of annotations:

Annotation

Annotation, with background color and border that may be of help to the user. An Annotation is a descriptive comment placed on the business model diagram.

Transparent annotation

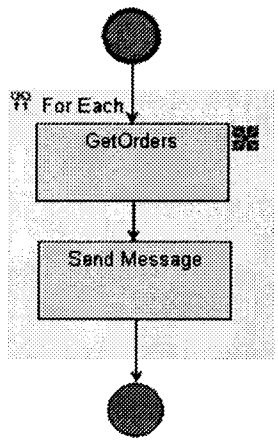
Annotation, in plain text and no border that may be of help to the user. An Annotation is a descriptive comment placed on the business model diagram.

III.3.4 Groups

Finally, different BPM components can be placed in a group in order to execute actions for all of them, or fix a common context for these components. In studio there are the four following kind of groups:

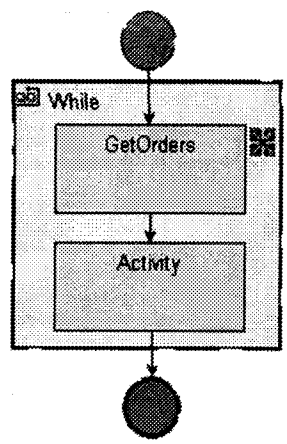
III.3.4.1 For each

All Activities or sub-Processes of the group are executed for each sub-part of a message.



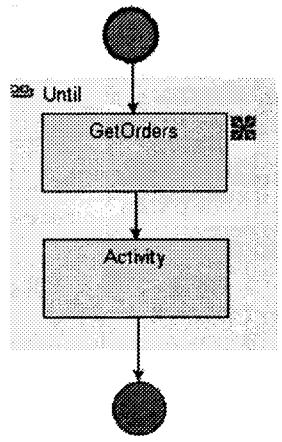
III.3.4.2 While

All Activities or sub-Processes of the group are executed as long as the condition is true.



III.3.4.3 Until

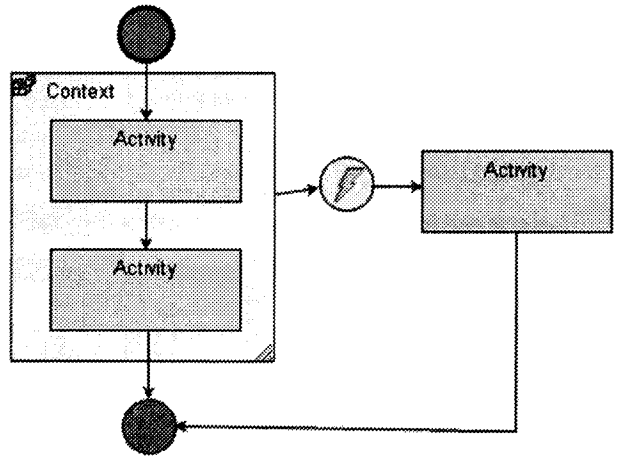
All Activities or sub-Processes of the group are executed till the condition is false. Unlike 'While', here the Activities or sub-Processes are executed at least once.



Graphically, as you can see above, there's no difference between these three groups.

III.3.4.4 Context

Sometimes is necessary to group different components without specifying any condition for an action, but just associating events or workflows to all of the context components.



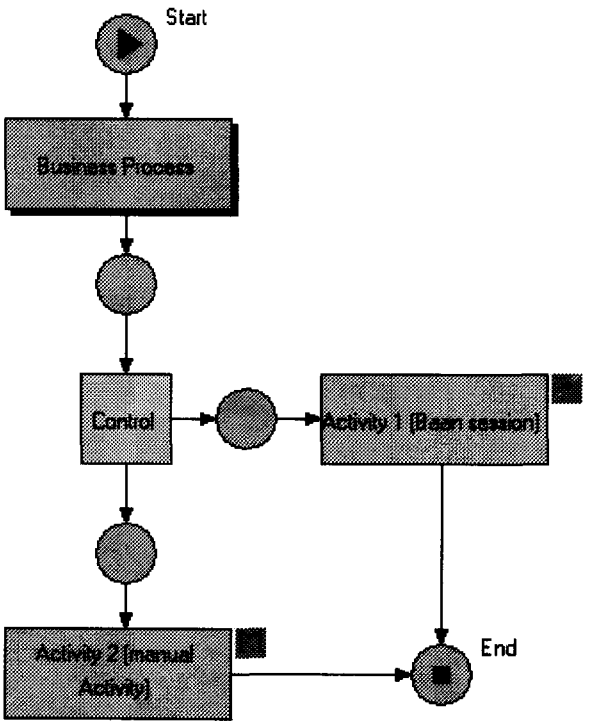
III.4 Comparison DEM to Studio

Here we include a comparison between DEM models and Studio BPM models, showing the correspondences between the graphic components and some properties of both systems.

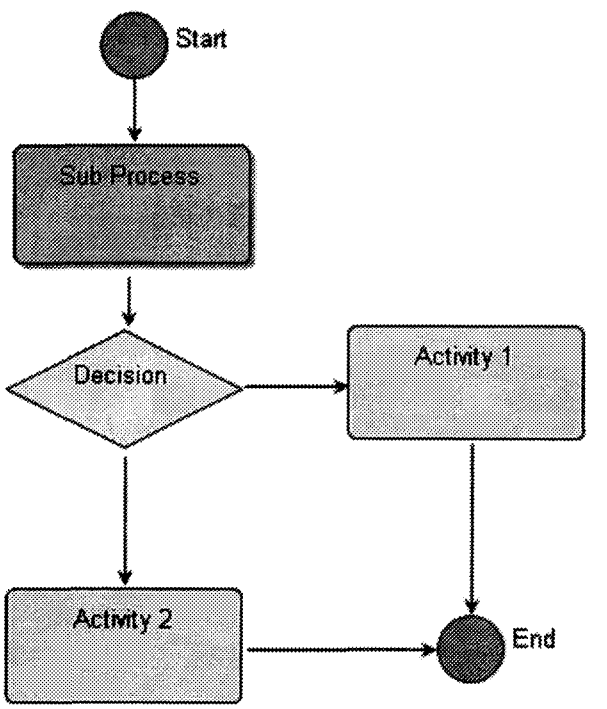
III.4.1 Example of DEM and BPM diagram

First, to give an easy and intuitive approach, we include an example of a model being represented in Baan System and in Cordys Studio. It is easy to see the differences and similarities between the two models displayed in the table below:

DEM:



Studio:



III.4.2 Graphical comparison

DEM COMPONENTS:	STUDIO COMPONENTS:
Begin State	Start State
Activity	Activity
Control Activity	Decision
Relationship	Connector
End State	End State
Business Process (shaded box)	Sub-process
Text	Annotation
Free Text	Transparent Annotation
Role	*Not converted (Comment will be added as annotation to show how it can be added in Cordys Studio).*
Utility	*Not converted (Comment will be added as annotation to show how it can be represented in Cordys Studio).*
Join	*Not converted*

About events, note that in DEM models an event is linked to a concrete activity, but in Studio one event is represented as between two activities. You can see an event more as a transition between these activities than as a strict action linked to one of them.

For instance, a timer event associated to an activity X in DEM is represented in Studio as an event between a previous activity and X.

III.4.3 Taking Decisions in DEM and Studio

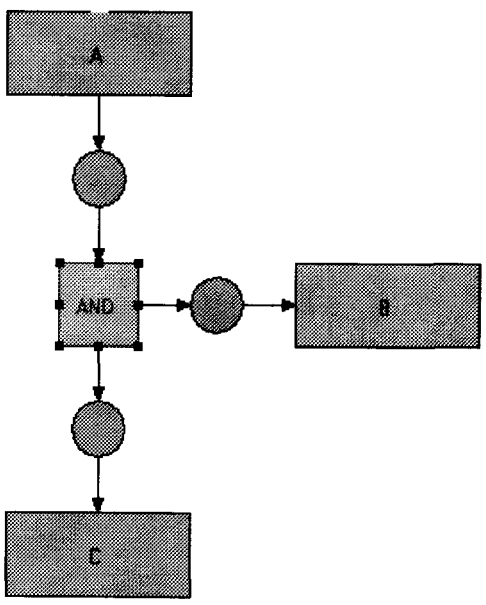
Sometimes in the workflow of a business process it is necessary to include a decision, which is depending on a condition. Graphically, an arrow is split in two or more arrows representing the different workflows to follow depending on the taken decision, and both DEM models and Studio BPM models have a defined component to represent this. These components are **Control Activity** and **Decision**, respectively.

But also DEM and Studio have different graphic structure, mainly because of the absence of intermediary states in Studio. Therefore is useful to detail here the equivalences and differences of both models when they try to represent logical conditions, i. e., show how each logic construction from DEM is represented in Studio.

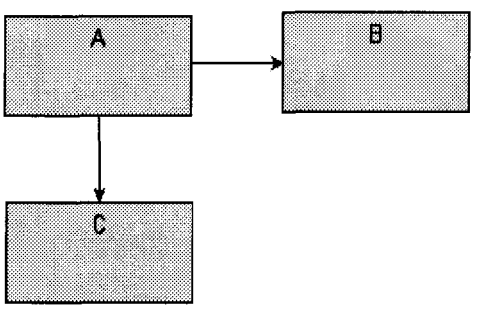
III.4.3.1 Operator And

If the type of control function is **And**, an explicit representation of the logic can be displayed is Studio, even without a decision component. Looking at the example below, we can see that this is because the only needed is to show in the diagram than an activity can access both of the following activity. When we delete in Studio intermediary states, this information can be represented without decision component, as follow:

DEM:



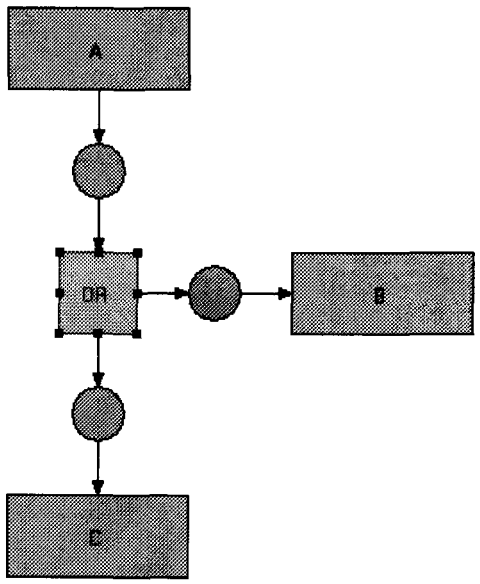
Studio:



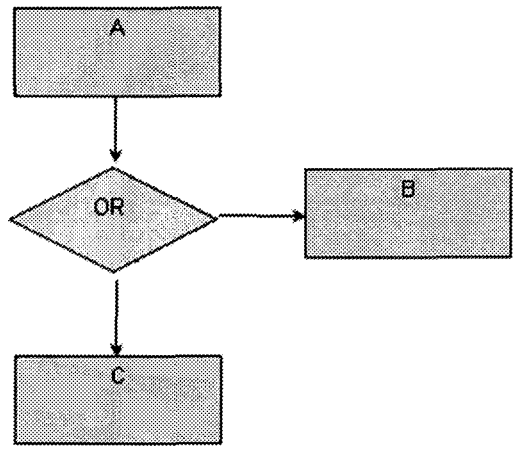
III.4.3.2 Operator Or and Xor

For the other two kinds of control function, **Or** and **Xor**, there are no significant differences between them. Here, a decision component is necessary in Studio to represent that you can access a different activity depending on the condition defined on this component. In more detail, is possible to define if you want to apply an **Or** condition or a **Xor** condition in the properties of the decision component. The general schema is showed in the diagrams below:

DEM:



Studio:

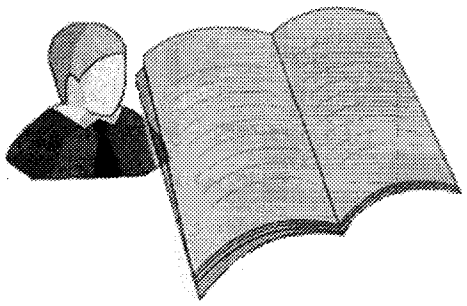


III.4.4 Comparison of DEM and Studio properties

DEM PROPERTIES	STUDIO PROPERTIESS
Business Process Activity	
Activity type	
Baan Session	*Component type*\Activity
Manual Activity	*Component type*\Activity
Business Process	*Component type*\Sub process
Control Activity	*Component type*\Decision
Other Application	*Properties*\Annotation (not converted)
Control type	We skip this for the moment
Code	*Properties**Application*\Code
External Code	*Not converted*
Argument	*Not converted*
Activity description	*Properties**General*\Description
Activity Category	*Not converted*
AO Document	*Not converted*
Work instruction	*Properties*\Annotation
Utility	*Not converted*
User event	*Not converted*
External event	*Not converted*
Timer event	*Not converted*
Automatic event	*Not converted*
State	
Description	*Properties**General*\Description
State text	*Properties*\Annotation
Not	*Not converted*
Condition type	*Not converted*
Condition	*Not converted*
Link to Business Process	*Not converted*
Begin State	*Component type*\Start event

■ End State	■ *Component type*\End event
Relationship	
■ Condition	■ We skip this for the moment.
■ Not	■ *Not converted*
■ Description	■ *Properties**General*\Description
Business Process	
■ Business process name	■ *Properties**General*\Description
■ Version	■ *Version**General*\Version
■ Description	■ *Version**General*\Description
■ Generation date	■ *Version**History**Creation*\Date
■ Owner	■ *Version**History**Creation*\User
■ Last change	■ *Version**History**Last Modification*\Date
■ User last change	■ *Version**History**Last Modification*\User
■ Business Process Category	■ *Not converted*
■ Utility	■ *Not converted*
■ Expired	■ *Not converted*
■ Work instruction	■ *Properties*\Annotation
Version	
■ Code	■ *General**Settings*\Version
■ Description	■ *General**Settings*\Description
■ Derived from version	■ *General**Settings*\Derived from version
■ Status	■ *Not converted*
■ Effective date	■ *History**Creation*\Date
■ Text	■ *Version**General*\Annotation
■ Finish date	■ *Not converted*
■ Expiry date	■ *Not converted*
■ Owner	■ *History**Creation*\User
■ Standard	■ *Not converted*
■ Description on printout	■ *Not converted*
Project	
■ Project model	■ *Properties**General*\Code
■ Description	■ *Properties**General*\Description

Version	*Properties\General*\Version
Text	*Properties\General*\Annotation
Generation Date	*Properties\History\Creation*\Date
Owner	*Properties\History\Creation*\User
Expired	*Not converted*
Copied from type	*Not converted*
Copied from model	*Not converted*
Copied from model version	*Not converted*
Name I	*Not converted*
Address I	*Not converted*
City I	*Not converted*
Name II	*Not converted*
Address II	*Not converted*
City II	*Not converted*
Internal Project Manager	*Not converted*
External Project Manager	*Not converted*



IV Technical Design

IV.1 Introduction

After Functional Design and before starting with the design and realization of the software, we completed our previous study of DEM and Studio Systems with the Technical Design. In this phase, what we did was enter a bit more into technical details about how DEM and BPM models and represented in Baan and CORDYS systems, respectively.

Then, the technical design is also structured in three big main steps. The first one is a document about the technical representation of DEM models in Baan, which is done by ASCII files. After that, a second document explains how BPM models are represented in Studio by XML files following some CORDYS standards. And finally we include the mapping between properties of both systems that was used after for the technical realization.

In other way, in the Technical Design we include the UML models to clarify how to realize the software, as well as a study about the structure and methods to use in our software.

IV.2 DEM data format: ASCII file

IV.2.1 Introduction

In this chapter we will offer a study in detail about how DEM models are stored internally in Baan IV systems in relational tables and how they can be exported in ASCII code files. First, is important to remark that the information contained in the DEM processes in structured in tables. Each table represents a concrete component of the processes, while each property or attribute of these components is stored as a column of the correspondent table.

Thus, all the information contained in one process can be stored in Baan IV within different tables, corresponding to the different components of the process diagram. Working with Baan IV DEM models is possible to distinguish between six main types of tables, which are **Versions, Processes, Relations, Free Text, Activities** and **States.**

These tables are structured according to a hierarchical structure showed in ***Figure 1.*** In the figure is easy to appreciate that every process has a version, and a version can be linked to different processes. In the concrete case of this project, all the DEM models are maintained in the same package, ***"tg"*** (package for the Organizer Baan IV), and the module is also fixed, ***"brg"*** (Module related to the Enterprise Modeler). Thus, all the DEM models are stored in ***tgbrgxxx***. So this study will be only concerned to these package and module ***"tgbrg"***. Note also from the diagram that every process can be composed of several relation, activities, states and annotations (free text), while each one of these components can belong only to one process.

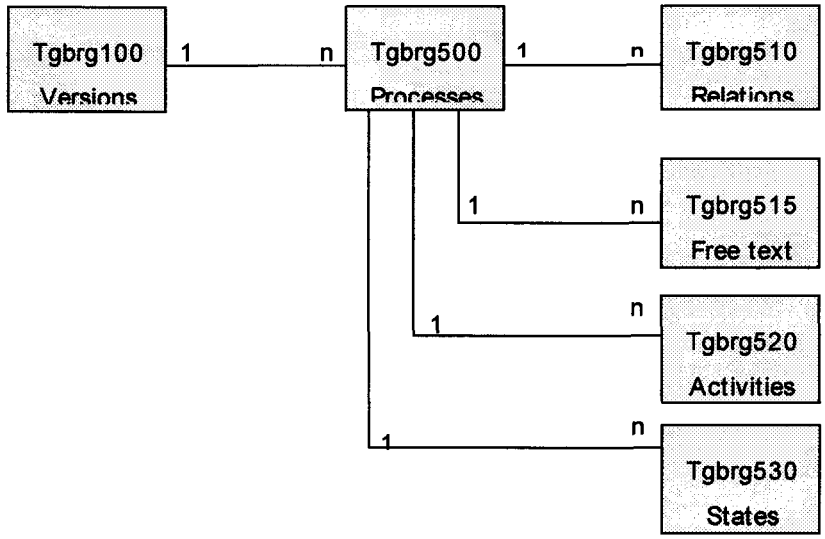
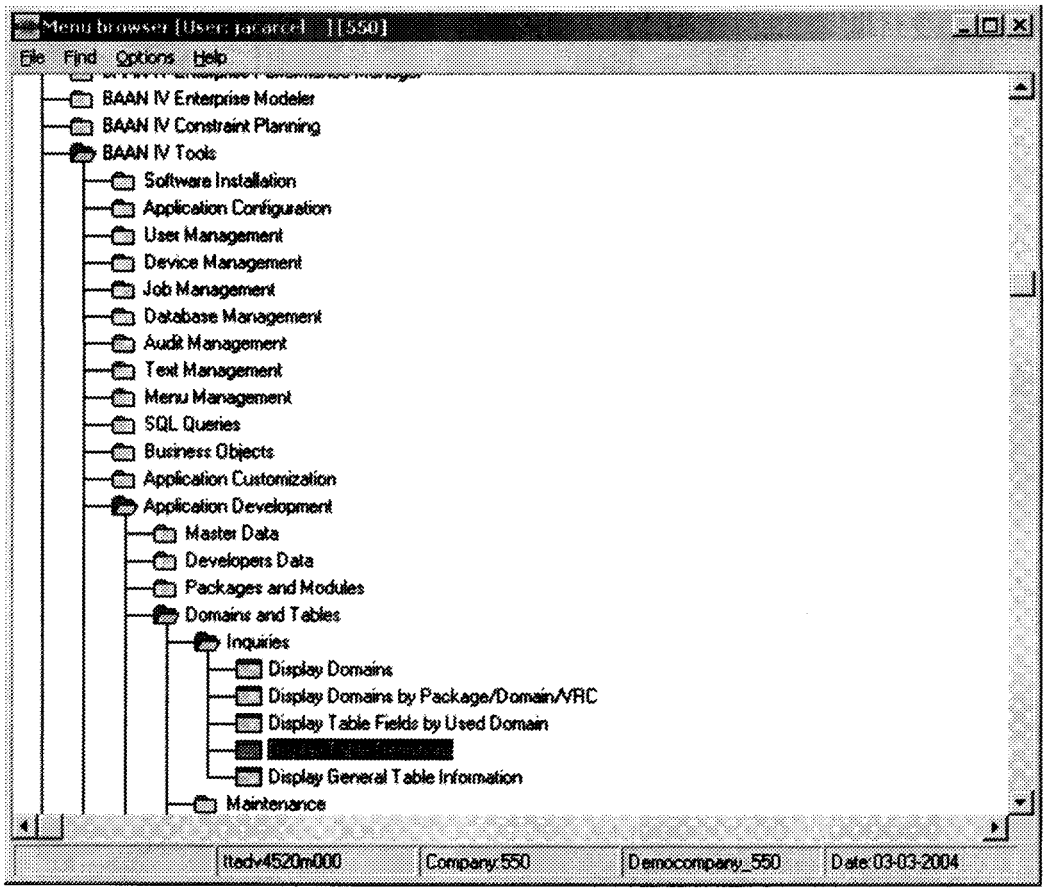


Figure 1: Structure of tables in Baan Sw.

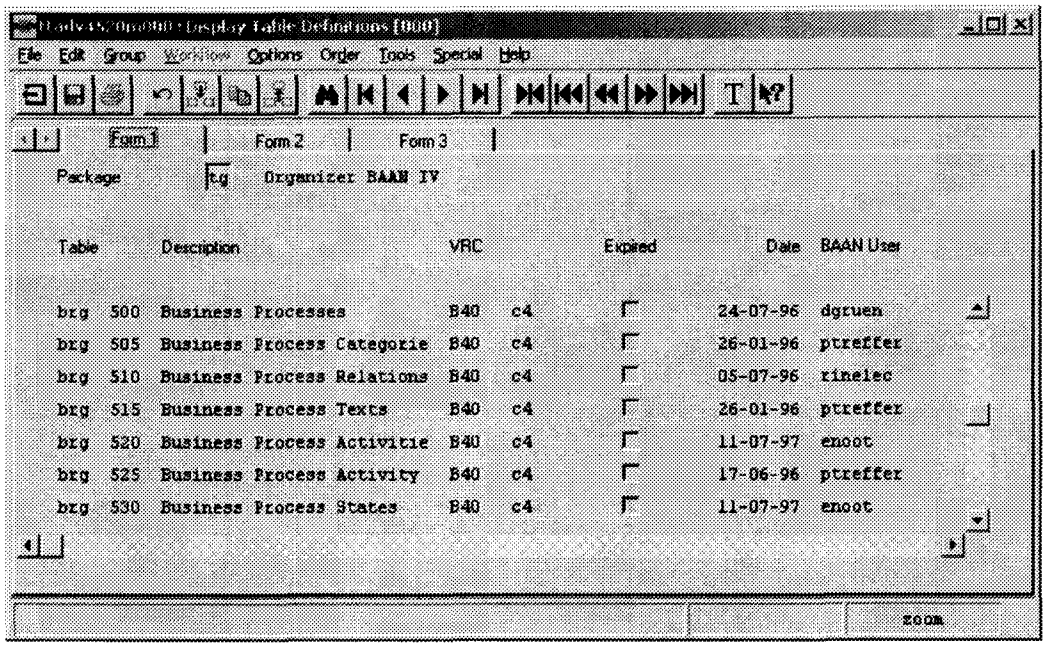
IV.2.2 Accessing Tables in Baan IV

To see all the table definitions in Baan IV, we choose the following options from the Menu Browser:

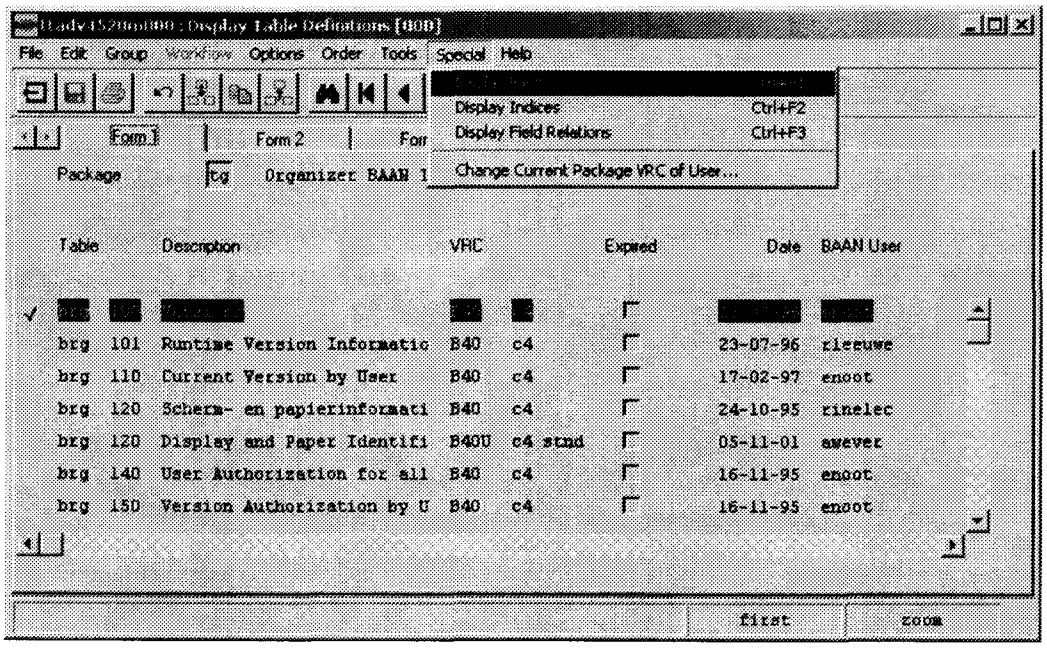
"**Baan IV Tools**" → "**Application Development**" → "**Domains and Tables**" → "**Inquiries**" → "**Display Table Definitions**".



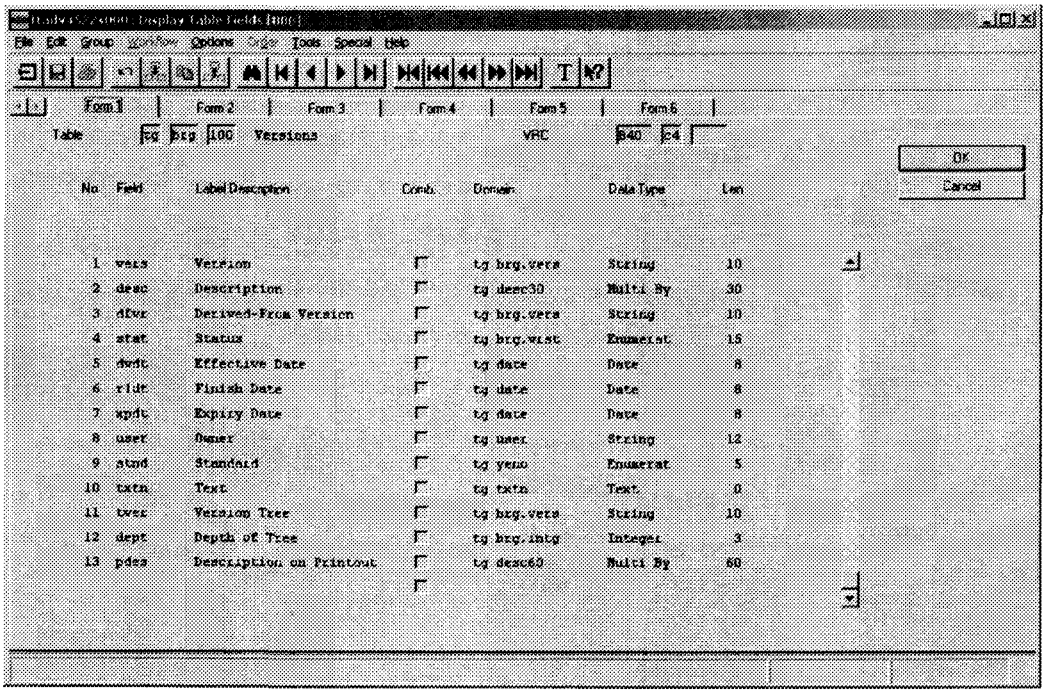
After selecting this option, we obtain the window displayed below, in which you can browse within the different DEM Data tables, and their attributes. Also is possible to select the package and module for the tables we want to display. To do that, we click on the **"Search"** icon (binocular symbol) from the toolbar. The picture shows the correspondent window for packet ***"tg"*** and module ***"brg"***.



To see in more detail the fields of one of the DEM Data tables, first is necessary to select one of the tables, and after we choose in the window menu the options **"Special" → "Display Fields"**, as you can see in the picture.



A new window **"Display Table Fields"** show all the attributes contained in the selected table. These attributes are stored as columns of the correspondent table. Below is displayed the window with the **Version** attributes (tgbrg100).



IV.2.3 DEM Tables and Fields

Now we will detail the different fields for each kind of table, enumerating all the fields for each type of table, and looking at their meaning, with a small description of each attribute. How to access the fields of each table is already explained in the previous point.

IV.2.3.1 Versions

Next Window shows the attributes of the **Versions** (tgbrg100):

Version: Code of the version.

Description: Description of the version.

Derived-From Version: With this property you can indicate that the current version comes from a previous version.

Status: Status of the current version. It can be **"Developing", "Finished"** or **"Expired".**

Effective Date: Creation Date of the Version.

Finish Date: Date when the version status if fixed to **"Finished".**

Expiry Date: Date when the version status if fixed to **"Expired".**

Owner: User who create the version.

Standard: Boolean to show if the current version is standard or not.

Text: Explanation or annotation referred to the version.

Version Tree: List of previous versions.

Depth of the Tree: Depth of list of previous versions.

Description on Printout: Description to be displayed and printed at the bottom of the Version diagrams.

IV.2.3.2 Processes

Next Window shows the attributes of the **Business Processes** (tgbrg500):

Business Processes: Name or code of the business process.

Version: Code of the version linked to the process.

Description: General Description.

Business Process Category: Processes can be grouped in a higher level, **Categories**. To do that, categories must be created, defined and linked to the processes.

Work Instruction: Comment that can be added to a process to add more information.

Owner: User that create the Process.

Generation Date: Creation date of the process.

Expired: Boolean value to show if the current process is expired or not.

Utility: Field that allow to link a utility to the process.

Proc-vers: Combined field, which is used as a foreign key.

Help Text Code: Runtime help code. Runtime help is generated from work instruction.

Last Change: Date of the last modification in the process.

User Last Change: User that made the last modification in the process.

IV.2.3.3 Relations

Attributes of the **Business Process Relations** (tgbrg510):

Business Process: Name or code of the Business Process of the relation.

Version: Version of the Business Process of the relation.

Position Number From: Position of the relation source component.

Position Number To: Position of the destination source component.

Component Type From: Type of the relation source component.

Component Type To: Type of the relation destination component.

X-Position: List of X coordinates used to place the relation (arrow).

Y-Position: :List of Y coordinates used to place the relation. About the length of this list, note that each coordinate is a point in the line, and a line has at least always two points.

Condition Type: With this field is possible to specify the kind of condition that can be linked to an input status. There are two types, **"Dynamic Condition"** and **"Static Condition"**.

Not: The logical function "not" can be applied to the static condition linked to the relationship with this field.

Condition: Reference for a possible static condition linked to the relation.

Description: General description of the linked static condition.

Proc-vers: Combined field for foreign key.

Proc-vers-posf-post: Combined field for foreign key..

Dynamic Condition: Reference for a possible dynamic condition linked to the relation.

Not: The logical function "not" can be applied to the relationship with this field.

Description: Description of the dynamic condition.

IV.2.3.4 Free Texts

Attributes of the **Business Process Texts** (tgbrg515):

Business Process: Code or name of the business process of the text.

Version: Version of the business process of the text.

Position Number: Identifier for the text component.

Text: Text to display as a comment in the editor.

Font: Type of letter selected to display the comment.

Position Number (xpos): X coordinate used to place the free text (left-top corner of the text box).

Position Number (ypos): Y coordinate used to place the free text.

Proc-vers: Combined foreign key field.

IV.2.3.5 Activities

Attributes of the **Business Process Activities** (tgbrg520):

Business Process: Name or code of the business process of the activity.

Version: Version code.

Position Number: Internal Identifier for the activity component.

External Code: External identifier for the activity, that is, the code that will be displayed to identify the activity in diagrams and reports.

Activity Description: General description of the activity.

Activity Type: Kind of activity (Baan Session, Other application program, Control Activity, Manual Activity or Business Process).

Control Type: Logical function that can be applied to a control activity (OR, XOR, AND, JOIN).

Program Code: Session code.

Argument: This field can be only used if the type of the activity is Baan Session or Other Application Program. With this argument you specify the activity functionality.

Nested Business Process: Sub process code.

AO (Administrative Organization) Document: This field show if there is any administrative form linked to the current activity.

Activity Category: Activities can be grouped in a higher level, **Activity Categories,** according to their common properties.

Utility: Field that allows to link a utility to the correspondent activity.

Work Instruction: Comment that can be added to an activity to provide more information.

Position Number (xpos): X coordinate used to place the activity (left corner on the top on the box).

Position Number (ypos): Y coordinate used to place the activity (left corner on the top on the box).

Width: Size for the width dimension of the graphic activity (Box).

Height: Size for the height dimension of the graphic activity (Box).

Help Text Code: Runtime help code, which is generated from work intruction.

User Event: Boolean to indicate if the activity has linked a user event.

External Event: Boolean to indicate if the activity has linked an external event.

Timer Event: Boolean to indicate if the activity has linked a timer event.

Automatic Event: Boolean to indicate if the activity has linked an automatic event.

Proc-vers: Combined foreign key field.

Proc-vers-pono: Combined foreign key field.

IV.2.3.6 States

Attributes of the **Business Process States** (tgbrg530):

Business Process: Name or code of the business process of the state.

Version: Version code.

Position Number: Internal Identifier of the state.

External Code: External Identifier of the state, that is, the code that will be displayed to identify the activity in diagrams and reports.

State Description: Description of the state.

Link to Business Process: Linked Process Code.

Condition Type: With this field is possible to specify the kind of condition that can be linked to an input status. There are two types, **"Dynamic Condition"** and **"Static Condition"**.

Not: The logical function "not" can be applied to the state with this field.

Condition: Condition, which is linked to the state.

State Text: Text displayed inside the state.

Position Number (xpos): X coordinate used to place the state. (left coordinate).

Position Number (ypos): Y coordinate used to place the state. (top coordinate).

Width (width): Size for the width dimension of the graphic state.

Width (height): Size for the height dimension of the graphic state.

X-Position: X coordinate used to place the text of the state.

Y-Position: Y coordinate used to place text of the state.

Help Text Code: Runtime help code reference.

Proc-vers: Combined foreign key field.

Begin State: Boolean to show if the state is the Begin State of the process.

End State: Boolean to show if the state is the End State of the process.

Proc-vers-pono: Combined foreign key field.

IV.2.4 Analyzing the ASCII Files

As is already explained, DEM models are exported in Baan IV as ASCII code files. This point is a detailed analysis of these files, considering the structure of the files as well as the ASCII notation used to represent the different tables and files in the export file. For the first technical approach in this project, we only will take into consideration as relevant for this project the six tables explained above, from the package ***tg,*** module ***brg***.

IV.2.4.1 Structure of the ASCII Files

Every ASCII file used for the representation of a DEM model starts with a Header part. This part can offer both general information about the file and many comments to explain details about the used notation or any useful explanation considered necessary to make easier the understanding of the file.

After the Header Part, the Contents Part present all the information contained in the source DEM model organized in tables. In our concrete case, the six main tables of the package ***tg,*** module ***brg,*** appears in the ASCII file following the same hierarchical order that is showed in *Figure 1*. For each table, every component is stored as an entry containing all its fields. After all the fields of a component, a new component entry is placed, and after all the components of a table, a new table is placed in the file. Next figure show the general structure of an export dump of DEM models. Since is only an example, we only put here a few tables to show how table entries and field entries are structured in the file.

```
================================================================
================================================================
```

HEADER PART

CONTENTS PART

- *Version table*
  - First version entry.
- *Process table*
  - First Process entry
    - First entry field
    - Second entry field
    - Third entry field
  - Second Process entry
    - First entry field
    - Second entry field
    - Third entry field
- *Free text Table*
  - First Text entry
    - First entry field
    - Second entry field
    - Third entry field
    - .
    - .
    - etc.....

```
================================================================
================================================================
```

IV.2.4.2 ASCII Notation for Tables and Fields

An explanation in detail about the ASCII data format is necessary to understand how the information contained in the files should be interpreted. A few preliminary concepts must be introduced here before enter into the code.

Every record in a table is represented by a line with the symbol **"#"**, and a concrete tag. After this line, the fields of this entry are also represented by a new line with the symbol **"#"** followed by a specific tag. Some information is also displayed for each field in the same line to make the file clearer, although this additional information is finally not considered as ASCII code. Most usually, a line representing a field will be preceding a line without the symbol **"#"**, which represents the concrete contents filled for that field.

Finally, it is important to note that the type of each field is specified by means of a prefix in the beginning of the tag. The existing prefixes in Baan Sw for the representation of DEM models are the following:

```
The prefix #TB represents a table.
The prefix #TD represents a table field of type date.
The prefix #TE represents a table field of type enume.
The prefix #TL represents a table field of type long.
The prefix #TF represents a table field of type float/double.
The prefix #TS represents a table field of type string.
The prefix #TX represents a table field of type text.
The prefix #KW1 represents a keyword 1 of text.
The prefix #KW2 represents a keyword 2 of text.
The prefix #KW3 represents a keyword 3 of text.
The prefix #KW4 represents a keyword 4 of text.
The prefix #TXG represents a text group of text.
The prefix #EOP represents edit options of text.
```

IV.2.4.3 ASCII Example

As an example, now we include an entry for an activity and one of its fields to comment the ASCII notation in more detail and show how to read the fields in the ASCII files.

```
#TBtgbrg520 -- (Business Process Activities) --
#TStgbrg520.proc S6 (Business Process) #string 6 index part#
proc3
```

The First line is representing the entry of a new Business Process Activity. The line starts with the symbol # followed for the basic tag to represent a Business Process Activity Table, ***TBtgbrg520***. This tag is composed of the different parts:

```
#TBtgbrg520 -- (Business Process Activities) --
```

TB: This prefix show that the tag represents a table.

Tg: Code of the package.

Brg: Code of the module.

520: Number that refers to the internal tgbrg table **"Business Process Activities"**.

After this, next lines introduce all the fields related to this table. These lines usually contain a representation of field, which is the basic tag of the table followed by a dot and a field code, plus other information displayed, which is not really considered as ASCII code. Thus, second line is composed of the next parts:

```
#TStgbrg520.proc S6 (Business Process) #string 6 index part#
```

#TStgbrg520.proc: Basic representing the **"Business Process"** field inside the **"Business Process Activities"** table.

S6: Information displayed about the type of the file (String of length 6).

(Business Process): Information displayed about the name of the file.

#string 6 index part#: Comment about the field.

The third lines do not start with the symbol **"#"**, so is a content line. It offers the content of the previous field. In this case we just find the Business Process name **"proc3"**.

IV.2.4.4 Example of Version and Business process table:

```
Baan Enterprise Modeler Dump (tg B40O c4 stnd)
Version : Test (Test)
DATE    : 04-02-17 [13:16]
Login   : ofournie (Olvier Fournier)
System  : poseidon
$BSE    : /usr4/b40c4c/bse
Company : 550   (Democompany_550)
Language: 2     (English)
-------------------------------------------------------------------------
#No Previous Versions
-------------------------------------------------------------------------
#TBtgbrg100 ---------- (Versions) ----------
#TStgbrg100.vers S10 (Version) #string 10 index part#
Test
#TStgbrg100.desc S30 (Description) #string 30#
Test
#TStgbrg100.dfvr S10 (Derived-From Version) #string 10#

#TEtgbrg100.stat (Status) #enume tgbrg.vrst#
tgbrg.vrst.developing
#TDtgbrg100.dvdt (Effective Date) #date(yyyy,mm,dd)#
date(2004,02,02)
#TDtgbrg100.rldt (Finish Date) #date(yyyy,mm,dd)#

#TDtgbrg100.xpdt (Expiry Date) #date(yyyy,mm,dd)#

#TStgbrg100.user S12 (Owner) #string 12#
ofournie
#TEtgbrg100.stnd (Standard) #enume tgyeno#
tgyeno.no
#TXtgbrg100.txtn (Text) #text#
#TStgbrg100.tver A20 S10 (Version Tree) #string 10#
Test
```

```
#TLtgbrg100.dept (Depth of Tree) #integer#
1
#TStgbrg100.pdes S60 (Description on Printout) #string 60#

#TBtgbrg500 ---------- (Business Processes) ----------
#TStgbrg500.proc S6 (Business Process) #string 6 index part#
  joan
#TStgbrg500.vers S10 (Version) #string 10 index part#
Test
#TStgbrg500.desc S60 (Description) #string 60#
dggdg
#TStgbrg500.bpca S6 (Business Process Category) #string 6#

#TXtgbrg500.txtn (Work Instruction) #text#
#TStgbrg500.user S12 (Owner) #string 12#
jacarcel
#TDtgbrg500.date (Generation Date) #date(yyyy,mm,dd)#
date(2004,02,03)
#TEtgbrg500.expi (Expired) #enume tgyeno#
tgyeno.no
#TStgbrg500.cuti S10 (Utility) #string 10#

#TStgbrg500.txtc S32 (Help Text Code) #string 32#

#TDtgbrg500.cdat (Last Change) #date(yyyy,mm,dd)#
date(2004,02,03)
#TStgbrg500.upus S12 (User Last Change) #string 12#
jacarcel
```

IV.2.5 Properties converted

Next tables show a complete view about all the mentioned aspects, with all the fields from the considered tables, the correspondent DEM ASCII tags for each one, and a column to show if the related field will be take into consideration for the conversion or not.

Version (tgbrg 100)					
Fields	Label Description	Domain	Data type	DEM ASCII format	Convert
vers	Version	Tg brg.vers	String	#TStgbrg100.vers S10	Yes
desc	Description	Tg desc30	Multi By	#TStgbrg100.desc S30	Yes
dfvr	Derived from Version	Tg brg.vers	String	#TStgbrg100.dfvr S10	Yes
stat	Status	Tg brg.vrst	Enumerat	#TEtgbrg100.stat	No
dvdt	Effective date	Tg date	Date	#TDtgbrg100.dvdt	Yes
rldt	Finish date	Tg date	Date	#TDtgbrg100.rldt	No
xpdt	Expiry date	Tg date	Date	#TDtgbrg100.xpdt	No
user	Owner	Tg user	String	#TStgbrg100.user S12	Yes
stnd	Standard	Tg yeno	Enumerat	#TEtgbrg100.stnd	No
txtn	Text	Tg txtn	Text	#TXtgbrg100.txtn	Yes
tver	Version tree	Tg brg.vers	String	#TStgbrg100.tver A20 S10	No
dept	Depth of tree	Tg brg.intg	Integer	#TLtgbrg100.dept	No
pdes	Description on printout	Tg desc60	Multi By	#TStgbrg100.pdes S60	No
Processes (tgbrg 500)					
Fields	Label Description	Domain	Data type	DEM ASCII format	Convert
proc	Business Process	Tg brg.proc	String	#TStgbrg500.proc S6	Yes
vers	Version	Tg brg.vers	String	#TStgbrg500.vers S10	Yes
desc	Description	Tg desc60	Multi By	#TStgbrg500.desc S60	Yes
bpca	Business process category	Tg brg.bpca	String	#TStgbrg500.bpca S6	No
txtn	Work instruction	Tg txtn	Text	#TXtgbrg500.txtn	Yes
user	Owner	Tg user	String	#TStgbrg500.user S12	Yes
date	Generation date	Tg date	Date	#TDtgbrg500.date	Yes

expi	Expired	Tg yeno	Enumerat	#TEtgbrg500.expi	No
cuti	Utility	Tg brg.cuti	String	#TStgbrg500.cuti S10	No
cmba	Proc-vers				No
txtc	Help text code	Tg st32	String	#TStgbrg500.txtc S32	No
cdat	Last change	Tg date	Date	#TDtgbrg500.cdat	Yes
upus	User last change	Tg user	String	#TStgbrg500.upus S12	Yes
Relations (tgbrg 510)					
Fields	**Label Description**	**Domain**	**Data type**	**DEM ASCII format**	**Convert**
proc	Business process	Tg brg.proc	String	#TStgbrg510.proc S6	No
prov	Version	Tg brg.vers	String	#TStgbrg510.prov S10	No
posf	Position number from	Tg brg.pono	Integer	#TLtgbrg510.posf (array of length 10)	Yes
post	Position number to	Tg brg.pono	Integer	#TLtgbrg510.post	Yes
ctpf	Component type from	Tg brg.dctp	Enumerat	#TEtgbrg510.ctpf	Yes
ctpt	Component type to	Tg brg.dctp	Enumerat	#TEtgbrg510.ctpt	Yes
posx	x-position	Tg brg.coord	Double	#TFtgbrg510.posx	Yes
posy	y-position	Tg brg.coord	Double	#TFtgbrg510.posy A10	Yes
cdtp	Condition type	Tg brg.cdtp	Enumerat	#TEtgbrg510.cdtp	No
noty	Not	Tg yenox	Enumerat	#TEtgbrg510.noty	No
cond	Condition	Tg brg.cond	String	#TStgbrg510.cond S6	No
desc	Description	Tg desc60	Multi by	#TStgbrg510.desc S60	Yes
cmba	Proc-vers				No
cmbb	Proc-vers-posf-post				No
dcon	Dynamic condition	Tg brg.cond	String	#TStgbrg510.dcon S6	No
dnot	Not	Tg yenox	Enumerat	#TEtgbrg510.dnot	No
ddes	Description	Tg desc60	Multi by	#TStgbrg510.ddes S60	No
Free text (tgbrg 515)					
Fields	**Label Description**	**Domain**	**Data type**	**DEM ASCII format**	**Convert**
proc	Business process	Tg brg.proc	String	#TStgbrg515.proc S6	No
vers	Version	Tg brg.vers	String	#TStgbrg515.vers S10	No
pono	Position number	Tg brg.pono	Integer	#TLtgbrg515.pono	Yes

text	Text	Tg desc60	Multi by	#TStgbrg515.text	Yes
font	Font	Tg st40	String	#TStgbrg515.font	Yes
xpos	X position	Tg brg.pono	Integer	#TLtgbrg515.xpos	Yes
ypos	Y position	Tg brg.pono	Integer	#TLtgbrg515.ypos	Yes
cmba	Erid-vers				No

Activities (tgbrg 520)

Fields	Label Description	Domain	Data type	DEM ASCII format	Convert
proc	Business processs	Tg brg.proc	String	#TStgbrg520.proc S6	No
prov	Version	Tg brg.vers	String	#TStgbrg520.prov S10	No
pono	Position number	Tg brg.pono	Integer	#TLtgbrg520.pono	Yes
extc	External code	Tg brg.extcode	String	#TStgbrg520.extc S32	No
desc	Activity description	Tg desc60	Multi by	#TStgbrg520.desc S60	Yes
actp	Activity type	Tg brg.actp	Enumerat	#TEtgbrg520.actp	Yes
cotp	Control type	Tg brg.cotp	Enumerat	#TEtgbrg520.cotp	No
prgm	Program code	Tg brg.prgm	String	#TStgbrg520.prgm S15	Yes
argu	Argument	Tg brg.argu	String	#TStgbrg520.argu S50	No
nbpr	Nested business process	Tg brg.proc	String	#TStgbrg520.nbpr S6	Yes
aodo	AO document	Tg brg.aodo	String	#TStgbrg520.aodo S6	No
catg	Activity category	Tg brg.catg	String	#TStgbrg520.catg S6	No
cuti	Utility	Tg brg.cuti	String	#TStgbrg520.cuti S10	No
wkin	Work instruction	Tg txtn	Text	#TXtgbrg520.wkin	Yes
xpos	Position number	Tg brg.coord	Double	#TFtgbrg520.xpos	Yes
ypos	Position number	Tg brg.coord	Double	#TFtgbrg520.ypos	Yes
widt	Width	Tg brg.coord	Double	#TFtgbrg520.widt	No
hght	Hight	Tg brg.coord	Double	#TFtgbrg520.hght	No
txtc	Help text code	Tg st32	String	#TStgbrg520.txtc S32	No
eusr	User event	Tg yenox	Enumerat	#TEtgbrg520.eusr	No
eext	External event	Tg yenox	Enumerat	#TEtgbrg520.eext	No
etim	Timer event	Tg yenox	Enumerat	#TEtgbrg520.etim	No
eaut	Automatic event	Tg yenox	Enumerat	#TEtgbrg520.eaut	No

cmba	Proc-vers				No
cmbb	Proc-vers-pono				No
States (tgbrg 530)					
Fields	**Label Description**	**Domain**	**Data type**	**DEM ASCII format**	**Convert**
proc	Business process	Tg brg.proc	String	#TStgbrg530.proc S6	No
prov	Version	Tg brg.vers	String	#TStgbrg530.prov S10	No
pono	Position number	Tg brg.pono	Integer	#TLtgbrg530.pono	Yes
extc	External code	Tg brg.extcode	String	#TStgbrg530.extc S32	No
desc	State description	Tg desc60	Multi by	#TStgbrg530.desc S60	Yes
nbpr	Link to business process	Tg brg.proc	String	#TStgbrg530.nbpr S6	No
cdtp	Condition type	Tg brg.cdtp	Enumerat	#TEtgbrg530.cdtp	No
noty	Not	Tg yenox	Enumerat	#TEtgbrg530.noty	No
cond	Condition	Tg brg.cond	String	#TStgbrg530.cond S6	No
txta	State text	Tg txtn	Text	#TXtgbrg530.txta	Yes
xpos	Position number	Tg brg.coord	Double	#TFtgbrg530.xpos	Yes
ypos	Position number	Tg brg.coord	Double	#TFtgbrg530.ypos	Yes
widt	Width	Tg brg.coord	Double	#TFtgbrg530.widt	No
hght	Width	Tg brg.coord	Double	#TFtgbrg530.hght	No
xtxt	X position	Tg brg.coord	Double	#TFtgbrg530.xtxt A3	No
ytxt	Y position	Tg brg.coord	Double	#TFtgbrg530.ytxt A3	No
txtc	Help text code	Tg st32	String	#TStgbrg530.txtc S32	No
cmba	Proc-vers				No
inpt	Begin state	Tg yenox	Enumerat	#TEtgbrg530.inpt	Yes
outp	End state	Tg yenox	Enumerat	#TEtgbrg530.outp	Yes
cmbb	Proc-vers-pono				No

IV.3 BPM data format: XML file

IV.3.1 Introduction

Business Process Models in Studio are stored internally in a CoBOC (Collaborative Business Object Cache) database. CoBOC is an internal tool provided by Cordys Integrator used in CORDYS for creating and manipulating business objects. Thus, CoBOC acts as a central repository and a container for XML objects (also called Business Objects), allowing the management of these objects. These models are represented internally as XML files, containing all the information that allows displaying a graphic diagram. In this document we will offer a list of all the relevant XML tags for the DEM models conversion, and also a resume of the structure of a XML file representing a BPM model.

To obtain these files you need to export contents from Cordys Studio. This functionality is provided by option *"Content Transfer Utility"*. Once you have exported your BPM models, Studio creates a zip file, which contains different kind of files. The type of the files to be exported can be decided, but usually the next four types are obtained:

Versions: General information about version.
File Types: Information about the file types supported by Cordys Studio. Not considered for this project.
Export Configuration: General information about the setting and parameters used for the export.
Models: One file per each model selected is generated, with all the information necessary to represent this model in XML.

The most important of these files for our purposes are the files representing the BPM models. Next point offers an overview about the structure of these files and the XML tags used for the notation.

IV.3.2 Structure of XML files in Studio

XML files representing BPM models are structured in a simple hierarchy showed by tabulations, as HTML files. Main tags of this hierarchy are explained below:

```
-<tuple>
  -<old>
    -<bpm>
      + <documentProperties>
      -<content>
        + <bpml>
        -<model>
          -<object>
            + <dataObject>
          </object>
        + <svg>
        + <tasks>
        + <modelProperties>
        <messageMapping>
```

```
    -</content>
   </bpm>
  </old>
</tuple>
```

Tag	Comments
<tuple>	Tag with properties about the storage of the file in CoboC.
<old>	Existing records. Fixed tag.
<bpm>	Type of the record. Fixed tag.
<documentProperties>	General properties about all the imported models.
<content>	Fixed tag, representing the start of the model contents definition inside the file.
<bpml>	Contains generated BPML.
<model>	Information directly related to the BPM components, including next tags:
<object>	Generic properties of an object
<dataObject>	Specific properties for a kind of object.
<svg>:	Used for printing
<tasks>:	Used for storing model warnings
<modelProperties>:	Properties about the current model.
<messageMapping>	Used to store message mappings

IV.3.3 XML Tags in BPM Data Format

In this paragraph we will describe in detail all the tags contained inside the interesting tags for our project. We consider interesting tags those which are less general, but more concretely linked with the representation of BPM models, and their components. Tags to be further developed in this aspect are **<documentProperties>**, **<modelProperties>**, **<model>**, and specially **<object>** and **<dataObject>**, which describe the properties of the BPM components. The others are not developed since are not relevant for the project. Each of these five significant tags will be further analyzed in this point.

IV.3.3.1 Document properties

Tag	Comments
<name>:	Name for the file.
<description>	Description of the model.
<mimetype>	Type of the process. For DEM conversion always bpm.

<notes>	Possible annotations linked to the business process.
<version>:	Version of the file.
<revision>:	Number of the revision of the file in studio(not yet implemented).
<createdBy>:	User that has made the model.
<creationDate>:	Date of the creation of the model. Is expressed in milliseconds (Java format).
<lastModifiedBy>:	User that made the last modification
<lastModificationDate>:	Date of the last modification. Is expressed in Java format.

IV.3.3.2 Model properties

Tag	Comments
<dataobject name>	
<documentproperties>	
<name>	Name of the model.
<description>	Description of the model.
<caption>	Description.
<mimeType>	Type of the process. It can be *"vcm"*, *"bcm"* or *"bpm"*. (Here it is bpm)
<notes>	Possible annotations linked to the business process.
<version>	Version of the model.
<revision>	Number of the revision of the file (still not implemented).
<createdBy>	User that has created the model.
<creationDate>	Date of the creation of the model.
<lastModifiedBy>	User that made the last modification on the model.
<lastModificationDate>	Date of the last modification of the model.
<validbpmi>	Boolean that shows if the process has no warnings.
<published>	Boolean that shows if the process has been published.

IV.3.3.3 Model

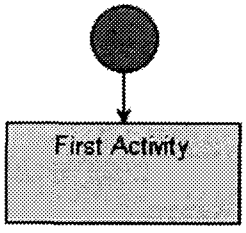
Properties	Comments
ModelFormatVersion	This tag is fixed to "1" which means that the model is represented by XML format of studio 1.0.
FreeObjectId	Identifier of the next object. This is the first identifier not yet used.

Zoom	Default zooms to display the model in the editor. Is usually fixed at 100.
NrObjects	Number of components included in the model.

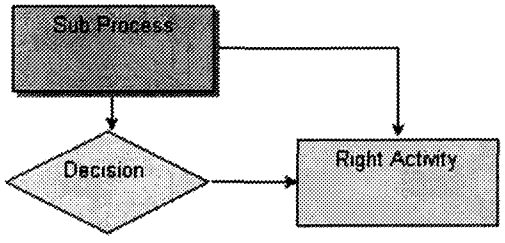
IV.3.3.4 Object

For this tag is important to note that properties of < **object>** tag change depending on the type of component. Most part of the specific properties in this tag are related with the way to draw relationships in Studio, so a explanation about is required.

The graphic representation of a relationship in Studio is an arrow. To represent this in Studio, we differentiate between arrows represented with only one straight line, and arrows represented with more than one line, as is shown in the picture below:

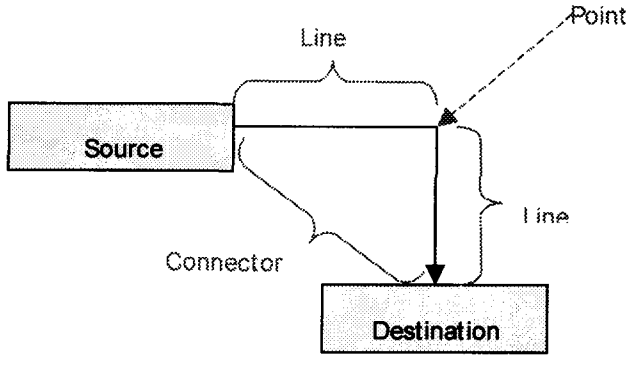


Simple connector between Start State and First Activity.



Simple connector between Subprocess and Decision. Composed connector between Subprocess and Right Activity.

If the relationship is displayed in the BPM diagram as a simple arrow, the correspondent XML components are a *"connector"* and a *"line"*. Otherwise, if the relationship is displayed in the BPM diagram as a composed arrow, the correspondent XML format contains a *"connector"* component, *"line"* components, and *"point"* components, to link these lines. The picture below offers a clear view about this:



Thus, these component types to represent relationships are the ones that present specific properties in the **< object>** tag. Note that *left* and *top* properties are not included in lines and connectors since its position is determined directly from the position of the *source* and *destination* components. But *left* and *top* properties must be included

in *point* components. So we include here an activity as an example of the four standard properties that are common to most of the components, and any other component with its common and specific properties:

Start:

Properties	Comments
Object Id	Internal identifier of the component
ObjectType	Type of the component
Left	Percentage value of an available horizontal space.
Top	Percentage value of an available vertical space.

End:

Properties	Comments
Object Id	Internal identifier of the component
ObjectType	Type of the component
Left	Percentage value of an available horizontal space.
Top	Percentage value of an available vertical space.

Activity:

Properties	Comments
Object Id	Internal identifier of the component
ObjectType	Type of the component
Left	Percentage value of an available horizontal space.
Top	Percentage value of an available vertical space.

Sub Process:

Properties	Comments
Object Id	Internal identifier of the component
ObjectType	Type of the component
Left	Percentage value of an available horizontal space.
Top	Percentage value of an available vertical space.

Decision:

Properties	Comments
Object Id	Internal identifier of the component
ObjectType	Type of the component
Left	Percentage value of an available horizontal space.
Top	Percentage value of an available vertical space.

Line:

Properties	Comments
Object id	Internal identifier of the component.
ObjectType	Type of the component.
ConnectorId	Internal identifier of the relation component.
Source:	Identifier of the source component
Destination	Identifier of the destination component. Note that source and destination for a line could be a point, if the connector is composed

Point:

Properties	Comments
Object id	Internal identifier of the component
ObjectType	Type of the component
ConnectorId	Internal identifier of the relation component, which is linked with the point.
Left	Percentage value of an available horizontal space
Top	Percentage value of an available vertical space

Connector:

Properties	Comments
Object id	Internal identifier of the component
ObjectType	Type of the component.
Connector Type	Fixed for BPM
Source	Internal identifier of the source component
Destination	Internal identifier of the destination component
Direction	Fixed for bpm to *"one-way connector"*.

Documentation:

Properties	Comments
Object id	Internal identifier of the component
ObjectType	Type of the component
Left	Percentage value of an available horizontal space
Top	Percentage value of an available vertical space
Connector Type	Fixed for BPM
Width	Percentage value of an available horizontal space
Height	Percentage value of an available vertical space

IV.3.3.5 Data Object

In this tag, properties inside the **<dataObject>** tag change as well depending on the type of component. Following the standard schema, what we first find in this tag is a group of general properties. After that, each object has different properties structured mainly according to the tags **<tab>** and **<group>**. Referred to the contents referred to the DEM conversion, we consider as relevant the tag **<property>** from the standard schema document, and specially the property *name* inside this tag. With this, we will

Then, here is detailed all the standard properties for the main components, with the common properties in each component, and also all its specific characteristics. The property *name,* from the standard schema document is analysed as well, defining all the different values that it can have in each case. Note that components with type *"line",* or *"point"* don't include **<dataObject>** tag inside the **<object>** tag.

Activity:

Properties & Tag	Comments
Name	Name of the object.
<name>	Name of the component.
<dummyActivity>	Boolean showing if the component is a dummy activity (task which is not executed at runtime).
<type>	Type of application service for activities. Two possible values: - *Application:* The activity needs user interaction. It can be an application or a manual task. - *Soap Service:* The activity needs no user interaction, but it can be executed by linking a SOAP service.
<url>	If the type of the activity is *"Application"*, this tag shows the URL to be run with the activity.
<messageType>	Message type from the scheduling properties panel. This tag is related to the *Scheduling* tab, which only appears if the *Application Service Task* of the activity is *"Application"*. When a Business Process is instantiated, the Activities that are of type Application and have a role linked to them are sent to the inbox or e-mail of all the users linked to the role. There are two possible values: - *Task:* The instantiated Business Process will wait till the task in the inbox is finished before continuing. - *Info:* The instantiated Business Process will continue uninterrupted.
<workDispatchType>	Property from the *Scheduling* tab that indicates to whom should be send the message. It can be **"All linked users"** or **"Users with least work"**.
<allUsersExecuteTask>	Boolean representing the property with the same name from the scheduling panel in the activity properties

<inboxCategory>	Runtime inbox category
<notificationSubject>	Runtime subjects of the task (email, for instance).
<annotation>	Tag to show possible comments or text added to this component.
<application>	Code for a possible application linked to the activity
<applicationType>	Type of the application. It can be *"Application"* or *"Soap Service"*.
<applicationDescription>	Description of the application
<isvpackage>:	Next four properties are referred to the SOAP methods from the repository. This one is the standard package in which these methods are based.
<namespace>	
<method>	Name of a SOAP method linked to the activity
<methodset>	Name of a set of SOAP methods.
<documents>	Documents can be linked to an activity, for example the work instruction for an Activity in PDF format. This tag shows the list of the linked documents.
<dynamicAssignment>	Boolean used to show if the user to execute the activity is dynamically assigned (true), or pre-defined (false).
<assignedUserFromMessage>	If the assignment is dynamic, the user or role that needs to execute the Activity is read from the message that is defined in the *"Read assigned user from message"* field in Studio. This tag shows also that user.
<inputmessage>	Message to be run when the process workflow arrives to the activity.
<outputmessage>	Message to be run when the process workflow arrives to the activity.
<roles>	This tag shows the roles that are allowed to execute a manual activity, if the assignment is static

Start State:

Properties & Tag	Comments
name	Name of the object.
<name>	Name of the component.
<triggerType>	There are three possible values for this tag, which specifies the action to be taken in the Start Event. - *Message:* At runtime, the process should be started with the defined message type. This can be used to start the process as a sub process from a main process.

	- *Timer:* The frequency as well as the time unit should be specified. Schedule templates are created at runtime to ensure that the processes are triggered at the appropriate time. *No Message or Timer:* Both previous techniques are not applied to the process.
<inputMessage>	Message to be run when the process start, if the tag **<triggerType>** is fixed to *Message*.
<timer>	Time unit used, if the tag **<triggerType>** is fixed to *Timer*.
<frequency>	Frequency used if the tag **<triggerType>** is fixed to *Timer*.
<annotation>	Tag to show possible comments or text added to this component

End State:

Properties & Tag	Comments
name	Name of the object.
<name>	Name of the component.
<endType>	Tag to show what kind of end state is applied in the model. There are four possible values: - *Message:* The end event can be an output message when the business process is used as a sub-Process of another process. - *Error:* The end event will be a 'process error' if a known error occurs in the process. - *Rollback:* A 'rollback' end event is reached if the entire process is to be rolled back. - *None:* None of the previous statements are applied.
<outputMessage>	Message to be send once the process is finished, if the tag **<endType>** is fixed to *Message*.
<processError>	Message to the send if the tag **<endType>** is fixed to *Error*.
<annotation>	Tag to show possible comments or text added to this component.

Decision:

Properties & Tag	Comments
name:	Name of the object.
<name>:	Name of the component.
<annotation>:	Annotation is the text or explanatory comments that can be included as a descriptive aid.

Sub Process:

Properties & Tag	Comments
name	Name of the object
<name>	Name of the component.
<process>	Name of the parent process
<businessChannel>	A subprocess can be started in different web servers from the standard. These alternatives servers are defined in Orchestrator. Only if the server to be run is not the Studio server, we show the alternative server with this tag.
<synchonize>	Boolean to show if the subprocess execution is synchronized with the parent process execution.
<secureCommunication>	This option can be only be used if the sub-process is called on a Web service via a Business Channel i.e., over HTTPS (secure HTTP).

<mimeType>	Type of the process. It can be *"vcm"*, *"bcm"* or *"bpm"*. (Here, it is bpm)
<inputMessage>	The message that is received from another BPM component (Often an Activity).
<outputMessage>	The message that is sent to another BPM component (Often an Activity).
<annotation>	Annotation is the text or explanatory comments that can be included as a descriptive aid

Connector:

Properties & Tag	Comments
name	Name of the object.
<isDefault>	If the connector is placed after a decision, this tag is a Boolean to show if the connector represents the default condition.
<*condition*>	Description of the condition for the connector.
<conditionDescription>	This tag contains the possible description for a relationship.

Documentation:

Properties & Tag	Comments
name	Name of the object.
<name>	Name of the component.
<border>	Boolean to show if the graphic border of the comment is displayed or not.
<transparent>	Boolean to show if the note appear in the editor diagram or not.
<fontSize>	Size and font for the annotation.

IV.3.4 Default Values in Cordys Studio

For the default values, you can see the appendix B.

IV.4 Mapping DEM to Studio:

IV.4.1 Introduction

Now we will describe how is converted each field from DEM ASCII files into a tag from Studio XML files. Firstly, is important to remember that the aim of the converter program is to obtain a zip file with different kind of files in Studio from a DEM export file. The zip file obtained contains the following documents:

- 1 export configuration file, with general information about the setting and parameters used for the export.
- 1 Versions file, with general information about versions.
- 1 file per exported model containing the description of that model.

The files representing exported models are the most important for the conversion, although sometimes we will use tags contained in another files. That is specified in the next point.

IV.4.2 Mapping

Version: All the converted fields of the *Version* table from Baan IV, which are mapped to tags from the *"Versions"* XML file obtained after the export of BPM models in Studio.

IV.4.2.1 Version

Version (tgbrg 100)	Studio BPMN format	
Label Description	Tag	In studio environment
Version	<documentProperties> <name>	*Version**General**Settings*\Version
Description	<documentProperties> <description>	*Version**General**Settings*\Description
Derived from Version	<content> <parentVersion>	*Version**General**Settings*\Derived from version
Effective date	<documentProperties> <creationDate>	*Version**History**Creation*\Date
Owner	<documentProperties> <createdBy>	*Version**History**Creation*\User
Text	<documentProperties> <notes>	*Version**General*\Annotation
	<tuple> Object Id	
	<documentProperties> <version>	

IV.4.2.2 Process

All the converted fields of the *Process* table from Baan IV are mapped to tags from the XML file related to the concrete model obtained after the export of BPM models in Studio. This file can be accessed after the exportation following the next route:

"Business model repository" → "Business models"

Process (tgbrg 500)	Studio BPMN format	
Label Description	Tag	In studio environment
Business Process	<documentProperties> <name>	*BPM Properties**General**General*\Code
Version	<documentProperties> <version>	*BPM Properties**General**General*\Version
Description	<documentProperties> <description>	*BPM Properties**General**General*\Description
Work instruction	<documentProperties> <notes>	*BPM Properties*\Annotation
Owner	<documentProperties> <createdBy>	*BPM Properties**History**Creation*\User
Generation date	<documentProperties> <creationDate>	*BPM Properties**History**Creation*\Date
Last change	<documentProperties> <lastModificationDate>	*BPM Properties**History**Last Modification*\Date
User last change	<documentProperties> <lastModifiedBy>	*BPM Properties**History**Last Modification*\User

IV.4.2.3 Relation

In Studio, connectors have properties inside the <object> tag with the identifiers of the source and destination components. For the mapping of several fields from the table *"Relations"*, the way is take the identifier of these components, and look for specific tags after a research of the concrete component with that identifier. Thus, as a notation for this method, we will put *Searchsource* when the mapped tag is a field from the <object> or <dataobject> tags, representing that the mapped tag is accessed after searching for an object with the same identifier as the source of the relationship. On the other hand, we will put *Searchdest* when the mapped tag come after searching for a component with the same identifier as the destination of the relationship.

Relation (tgbrg 510)		
Label Description	Tag	Comments
Position number from	<object> → Source	
Position number to	<object> → Destination	
Component type from	**Searchsource** → <object> → Object type	
Component type to	**Searchdest** → <object> → Object type	

x-position	*Connectors*: Is represented directly according to the source and destination positions. *Lines*: Is represented directly according to the source and destination positions. *Points*: <objectid> → left	Note that first and last points to represent relations (arrows) in DEM are shipped.
y-position	*Connectors*: Is represented directly according to the source and destination positions. *Lines*: Is represented directly according to the source and destination positions. *Points*: <objectid> → *left*	
Description	<dataobject> → conditiondescription	

IV.4.2.4 Free text

The object type from XML file used to represent free text from Baan is *"documentation"*.
Tags are mapped in the files corresponding to the exported models as follow:

Free text (tgbrg 515)		
Label Description	Tag	Comments
Position number	<object> → id	Each object needs to be identified by an number, to be able to build a link with another object.
Text	<dataobject> → Name	
Font	<dataobject> → fontSize	It is the font and the size for the character
X position	<object> → left	
Y position	<object> → top	

IV.4.2.5 Activity

Activity (tgbrg 520)		
Label Description	Tag	Comments
Position number	<object> → id	
Activity description	<dataobject> → name	
Activity type	<object> → objectType	
Program code	<dataobject> → url + argument	Still not decided.
Nested business process	<object> → objectType = Sub Process	Tag process of object type subprocess.
Work instruction	<dataobject> → annotation	
Position number	<object> → left	

(xpos)		
Position number (y pos)	<object> → top	

IV.4.2.6 State (Begin and End)

State (tgbrg 530)		
Label Description	Tag	Comments
Position number	<object> → id	
State description	<dataobject> → name	
State text	<dataobject> → annotation	
Position number (X)	<object> → left	
Position number (Y)	<object> → top	
Begin state	<object> → objectType = start Note that the field in Baan tables is a Boolean while here we have the name of the object type.	Tag process of object type start.
End state	<object> → objectType = end Note that the field in Baan tables is a Boolean while here we have the name of the object type.	Tag process of object type end.

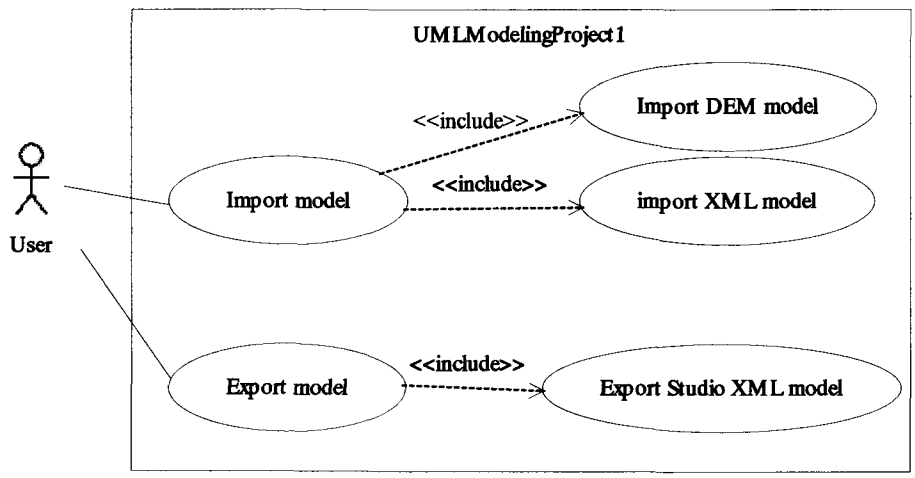
IV.5 UML and Architecture

IV.5.1 Introduction

In this chapter are included all the UML models that we developed to design our software converter. We respect the normal UML order to develop the design, i.e. first we have modeled the *Uses Case Diagram,* followed by the *Sequence Diagram*, *States Diagram* and *Classes Diagram.* It also includes a model about the architecture and methods used to implement our software.

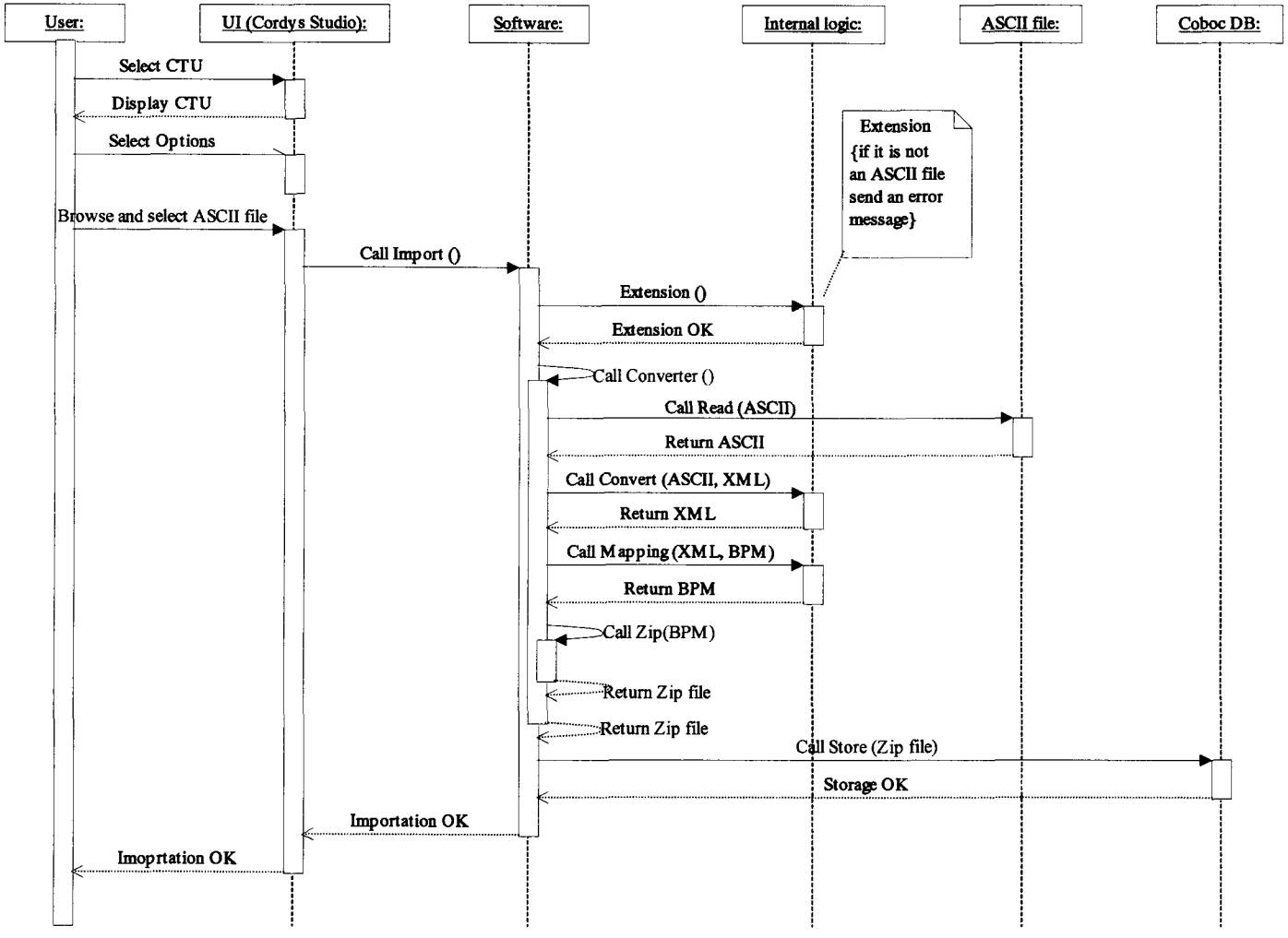
IV.5.2 Uses cases diagram

This is the most generic UML diagram that we have developed for the project. It shows how the project adds functionality to Cordys Studio. The main scenario for this diagram is the *Content Transfer Utility*, in which we already have two possible user cases, *Import Model* and *Export Model.* With the current project a new functionality is added to the *Import Model* option. The new functionality is *"Import DEM model"*.

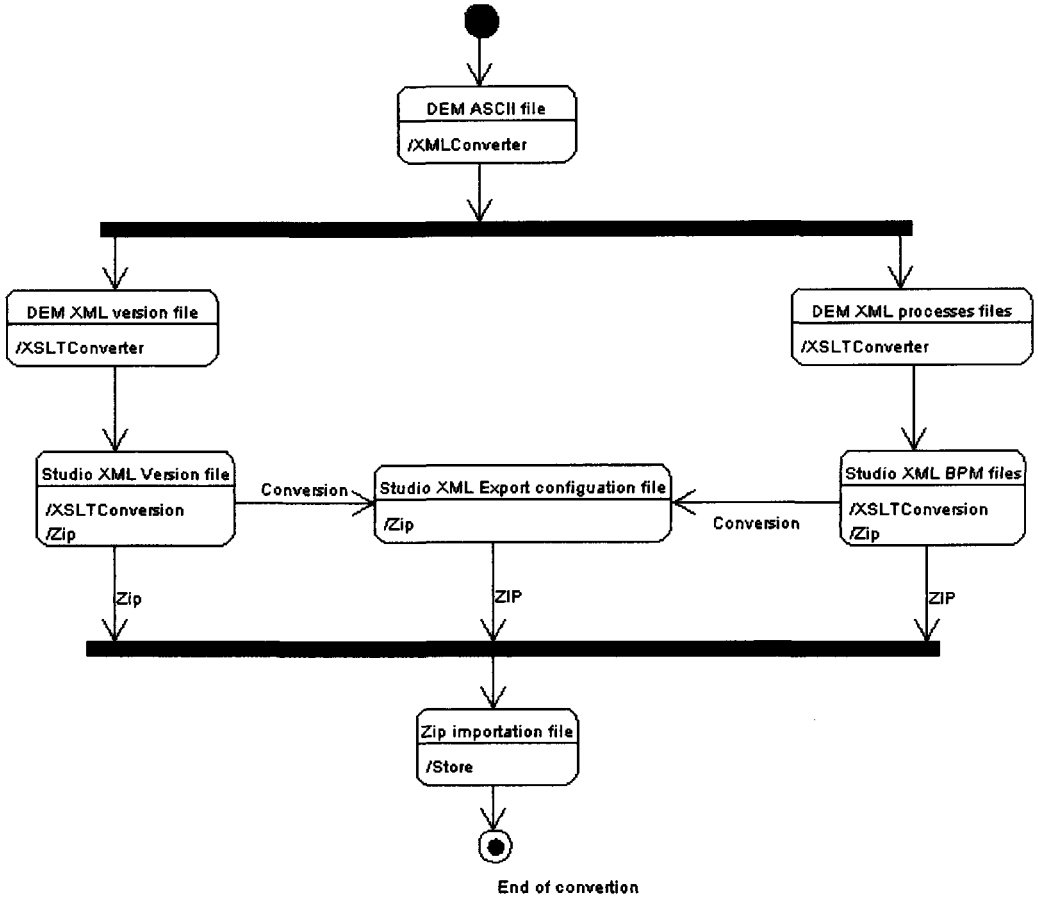


IV.5.3 Sequence diagram

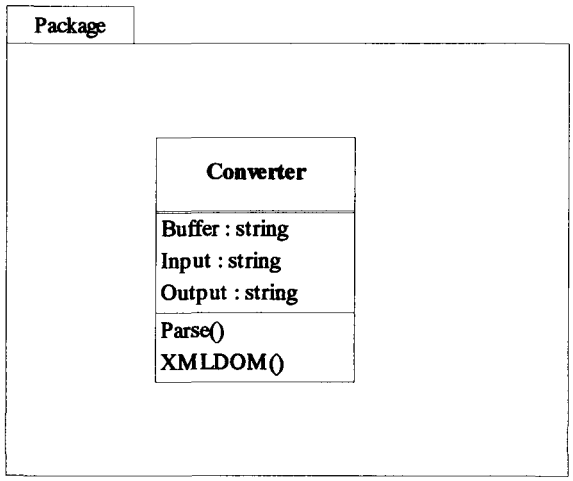
We develop the Sequence diagram for this new functionality, *"Import DEM model".* Note that CTU is an abbreviation of *"Content Transfer Utility"* here, and we do not consider the user choosing a non-ASCII file in the browser, since the scenario of the user case is specific for import DEM models. The behavior can be understood easily following the diagram, so no further information is needed in this case.


User:
UI (Cordys Studio):
Software:
Internal logic:
ASCII file:
Coboc DB:
Select CTU
Display CTU
Select Options
Browse and select ASCII file
Call Import ()
Extension ()
Extension OK
Call Converter ()
Call Read (ASCII)
Return ASCII
Call Convert (ASCII, XML)
Return XML
Call Mapping (XML, BPM)
Return BPM
Call Zip(BPM)
Return Zip file
Return Zip file
Call Store (Zip file)
Storage OK
Importation OK
Imoprtation OK
Extension
{if it is not an ASCII file send an error message}

IV.5.4 States diagram



IV.5.5 Classes Diagram

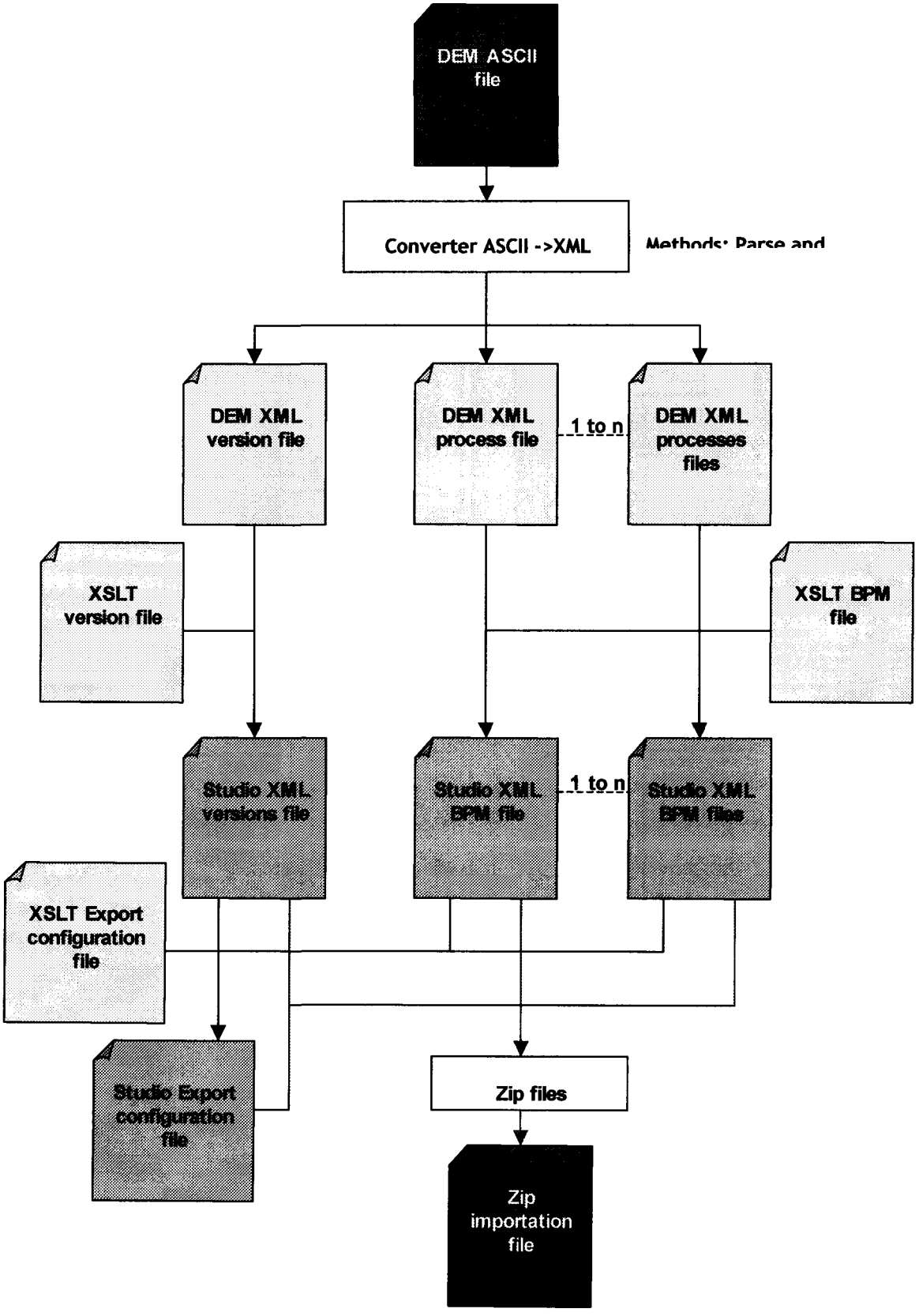


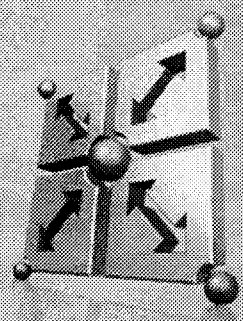
IV.5.6 Architecture

Entering inside the structure of our converter program, in further detail than with the UML diagram, we developed the architecture diagram displayed in the next page. To explain it is first necessary to establish that the software is structured in two main phases. The first one is to do a direct translation from ASCII code to XML code. However, Studio has its own standard format to represent BPM models using the XML language. Thus, a second phase is necessary to adapt the first XML code, which is obtained after the translation, to a new XML code according with the Studio standard.

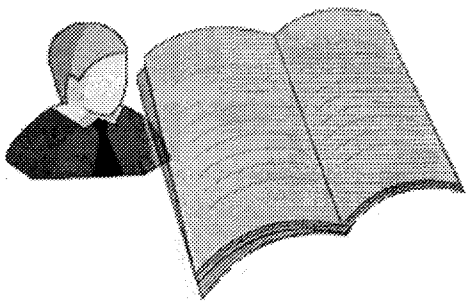
To code the first part of the program we used Java combined with XMLDOM to create the XML tree, while XSLT was the language used for the second part of the converter. We decide to structure the program in this way to make the software as flexible as possible in this second step. This is because Studio is a new platform, which makes more probable a change in the standard format of XML to represent BPM models. Thus, XSLT is a highly flexible language that will allow in a future to does changes in XML Studio format without entering into modify a complex code, but just implementing the changes in a general template.

Then, following the diagram below to explain the steps in more detail, we start by transforming the original DEM ASCII file into all the required XML files in Studio, that is, a versions file, an export configuration file, and one file per each BPM model. However, these are still XML files without the Studio standard format neither the right name of XML tags, this is what we call XML-DEM format. After that, each one of the generated XML-DEM files should be called from the same Java program to apply the correspondent XSLT template and make the file compatible with Studio. Finally, the files should be zipped to be able to import it in Studio.


DEM ASCII file
Converter ASCII ->XML
Methods: Parse and
DEM XML version file
DEM XML process file
1 to n
DEM XML processes files
XSLT version file
XSLT BPM file
Studio XML versions file
Studio XML BPM file
1 to n
Studio XML BPM files
XSLT Export configuration file
Studio Export configuration file
Zip files
Zip importation file



V Technical realization



Introduction

In this chapter is developed a general explanation about the two main issues that we deal with in the technical realization. First, the parser used in Java to classify the input file records and translate it into an XML-DEM file is explained here without entering in technical Java details. Little parts of the Java code are pasted to show how the program links with each part of the parser.

Also an explanation about XSLT language and an example file is included, to clarify details about the second step of the technical realization.

V.1 Parser Method

Here we will include a big part of the Java code used for the first part of the conversion to comment how we parse the DEM ASCII files into XML-DEM files. After parse all the components of the ASCII files, we stored them in a DOM tree, which we will use to apply a hierarchy within the different XML tags. Now we show and explain the parts of the code referred to the parse method:

- First we skip all the initial part of the file, which has no useful information, until we find the first line starting by "TX":

```
while(begin==false){
            line=lnr.readLine();
            if((line.charAt(0) == '#') && (line.charAt(1) == 'T')){
                  begin=true;
            }
      }
```

- Then we start to analyze the ASCII representation of the DEM model. Firstly, we check that lines are not empty, because we only will work with lines that are not empty. After we check if the first character of the line is different of "#". If is different we are in a content value line.

```
while(true){
      if(line.length() != 0){
            if((line.charAt(0) != '#') && (skip==false)){

            Behavior for content value lines.

            }
```

-The variable *"skip"* is to show if the content value is from a field that is not going to be skipped. We will skip the fields #KW and #EOP, so after we will fill the value Boolean variable *"skip"* to true when we will find these fields, to show that the contents should be skipped in the conversion. Following, we read the value and we store it in the DOM tree, but the code is not included here since is just technical Java language.

- Next necessary thing is to determine behavior for lines that start by the "#" symbol.

```
if(line.charAt(0)=='#'){ ...
```

-Thus, the first we do with these lines is determined if the next character is a "T". If that's the case, we have found a table line or a field line.

```
      if((line.charAt(1)== 'T')){
            if(line.charAt(2)=='X' && line.charAt(3)=='G'){
                  skip=true; //We skip the #TXG field
            }
```

```
            else{
                  skip=false;
            }
...
```

- If the third character after "#T" is a "N", we have detected the end of the file, and we apply the correspondent behavior. In this case, we only need to create the last bpm file and the export configuration file, but we also don't include here the Java details.

```
 if(line.charAt(2) == 'N') { //End of file detected
...
```

- Otherwise, if the third character after "#T" is a "B", what we have detected is a table entry. We determine what kind of table is by calling the function newtable().

```
if(line.charAt(2) == 'B') {
      space = line.indexOf(' ');
      table = line.substring(3,space);
      newtable(table);

      //WRITE TABLE IN THE DOM TREE
      node_table = doc.createElement(table);
      Rest of behavior for table entries
}
...
```

- Else, if the third character after "#T" is not "N", neither "B", what we have found is a normal field entry:

```
else { //Otherwise is a field
      Behavior for field entries.
}
...
```

- Now, we come back to an **else** clause that refers the case in which the first character is "#", but the second is not "T". These are the fields that should be skipped in the conversion, so the only we do here is put to true the value skip, to note also that the posterior content values should be skipped as well.

```
else{ //If the second character of the line is not T
      skip = true; //We have detected #KW or #EOP
}
...
```

- After all this parse method, we take a new line of the file. We continue then analyzing all the file like this, until we find the end of the file line "#TN".

```
try {
      line=lnr.readLine();
} catch (IOException io) {
      System.err.println("I/O failed in reading a new line\n");
}
```

V.2 Modular structure

To make the overview of the Java program more complete, in this point, we explain the functions that we use in the Java program, as well as the main relationships between them.

These relations are showed in the diagram displayed below, where you can see that the *main()* program, after apply a parser to the initial input (ASCII file), calls the function *createFile()* to obtain the XML-
DEM intermediate files, and from *createFile()*, the function *formatFile()* is called to obtain the final ouput (XML Studio files).

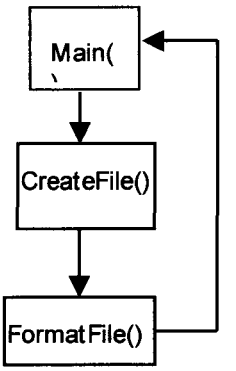


Figure 2: Structure of the Java functions

Here is a list of the used functions with a small explanation about each one:

- **newtable():** This function is called from the main program when a new table record is found to determine what kind of table it is.
- **createFile():** Function used to create the first XML-DEM files uing the DOM tree that has been filled according to the parser.
- **formatFile():** This function is applied after *createFile()*, to apply the XSLT templates to the intermediate XML-DEM files. After call this function, we obtain XML files that are similar to the final result, but a few details must be changed.
- **moveFile():** This function create the necessary folders to organize the files to import according to the Studio standards.
- **delete():** Here are deleted all the imported fields after zip them in only one file.
- **transformDate():** Function used to transform the Date fields from the DEM format (*date(22,06,2003)*), to the Studio format (milliseconds).
- **transformFont():** We use this function to transform the font of the free text.
- **Connectors():** Function used to transform all the fields related to connectors that had no direct correspondence in Studio.

V.3 Template: XSLT Version file

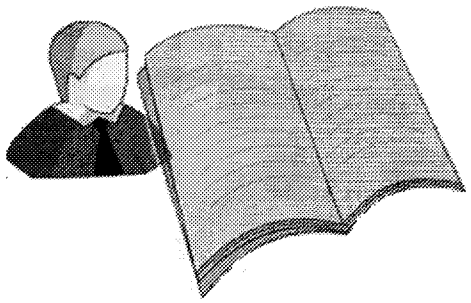
XSLT means **eX**tensible **S**tylesheet **L**anguage **T**ransformation. It is a language for transforming XML documents into other XML documents. It can add new elements into the output file (XML file), or remove elements. It can rearrange and test elements, make decisions about which elements to display. The execution of XSLT transforms an XML source structure into an XML result structure (tree). To execute this transformation, XSLT uses Xpath. Xpath is syntax for defining parts of an XML source document.

The following XSLT program represents the Template Version file of our project. This file transforms a XML DEM file to a BPM Version file of Studio.

```
//The first tag declares the version and encoding of our XSLT document.
<?xml version="1.0" encoding="utf-8"?>

//The second tag determine the source document.
<?xmlspysamplexml C:\eclipse\workspace\DEMpackage\Versions File.xml?>
```

```
//The root element declares the document as an XSL style sheet.
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    //Ouput tag give the information on our result (output) file.
    <xsl:output
    method="xml"
    version="string"
    omit-xml-declaration="no"
    indent="yes"
    media-type="string"/>

    //Template determines the path in the source file structure, where we going to apply the
    following program.
    <xsl:template match="//tgbrg100">

        //All elements and attributes represent the structure of our result file.
       <xsl:element name="GetCollection">
          <xsl:element name="tuple">
             <xsl:attribute name="key">C:\eclipse\workspace\DEMpackage\Version
             </xsl:attribute>
             <xsl:attribute name="objectID"/>
             <xsl:attribute name="description"><xsl:value-of
select="tgbrg100.desc"/></xsl:attribute>
             <xsl:attribute name="lastModified"/>
             <xsl:attribute name="name">

                //Value-of copies the value of an element or attribute that is selected in our
                source file.
                <xsl:value-of select="tgbrg100.vers"/></xsl:attribute>
             <xsl:attribute name="lastModified">
                <xsl:value-of select="tgbrg100.rldt"/></xsl:attribute>
             <xsl:attribute name="level"/>
             <xsl:element name="vcmVersion">
                <xsl:element name="old">
                   <xsl:element name="documentProperties">
                      <xsl:element name="name">
                         <xsl:value-of select="tgbrg100.vers"/></xsl:element>
                      <xsl:element name="description">
                         <xsl:value-of select="tgbrg100.desc"/></xsl:element>
                      <xsl:element name="caption"></xsl:element>
                      <xsl:element name="mimeType"></xsl:element>
                      <xsl:element name="notes">
                         <xsl:value-of select="tgbrg100.txtn"/></xsl:element>
                      <xsl:element name="version">
                         <xsl:value-of select="tgbrg100.vers"/></xsl:element>
                      <xsl:element name="revision"></xsl:element>
                      <xsl:element name="createdBy">
                         <xsl:value-of select="tgbrg100.user"/></xsl:element>
                      <xsl:element name="createdDate">
                         <xsl:value-of select="tgbrg100.dvdt"/></xsl:element>
                      <xsl:element name="lastModifiedBy"></xsl:element>
                      <xsl:element name="lastModifiedDate"></xsl:element>
                   </xsl:element>
                   <xsl:element name="Content">
                      <xsl:element name="parentVersion">
                         <xsl:value-of select="tgbrg100.dfvr"/></xsl:element>
```

```
                    </xsl:element>
                </xsl:element>
            </xsl:element>
        </xsl:element>
    </xsl:element>
</xsl:template >

</xsl:stylesheet>
```



VI Conclusion

VI.1 About the project

This project provides a new functionality to Cordys Studio allowing the transfer of old DEM models from Baan Sw into Studio BPM models from Cordys BCP platform, to be executable on Cordys BCP.

The first part of the project was mainly focused on the conversion of DEM models from Baan Sw to Studio BPM models from Cordys BCP platform.

In a general way, we realized that it was necessary to understand a certain number of very different technologies like DEM (Baan platform), Studio (Cordys BCP platform), Java environment, DEM ASCII format, XML, XSLT, BPML, BPMN, etc... For each one of these technologies, the objective was to obtain a good control and knowledge of the related techniques, in order to design and get the most efficient software.

However, during the project we modified a little bit our original program architecture. This is because we encountered many problems with XSLT language. The main one was that it is not possible to use dynamic variables in XSLT , and it was necessary to convert the connector components, and other parts of the mapping. So finally we decided to solve these problems in Java and unfortunately a small part of the mapping is not integrated in our XSLT template, but in the Java Code.

Nevertheless, the idea of templates to realize the mapping was a good choice. Thereafter, if there is an evolution of Studio, it will be easy to modify the converter software with the XSLT templates without enter into further technical knowledge of the java conversion program, but just applying the changes in the XSLT templates.

This first part of the work will be completed by an integration of our converter software in Studio like a new functionality.

The second part of the project will be to choose a reference model, completing this model after the conversion to be executable in Cordys BCP platform and develop an education guide.

VI.2 Challenge

It was an opportunity to enter the fast developing future world of e-business. We developed experience with modeling techniques and tools. In-depth knowledge on the concepts, tools modeling and workflow was achieved.