MOMOCS

Model driven Modernisation of Complex Systems

DELIVERABLE #D31B

METHODOLOGY SPECIFICATION

Dissemination Level:	Public
Work package:	3
Lead Participant:	SOFT
Contractual Delivery Date:	M24
Document status:	Final
Preparation Date:	03/10/2008

INDEX

INDEX OF TABLES

INDEX OF FIGURES

1 Scope

1.1 Purpose of the Document

This document describes the XIRUP modernisation methodology. This methodology identifies a set of activities, deliverables, milestones and iterations that are aimed to the modernisation of complex software systems. The elements of the methodology are interrelated in a set of diagrams using SPEM modelling standard [SPEM], which facilitates the formalization of the recommended software process, its deliverables, involved roles, guidance, and tools.

The document is driven by "MOMOCS - Description of Work", "D11 - Methodology Requirements", "D12 – Methodology Requirements: Industrial and Telecom Use-Cases", "D13 – State of the Art". It takes advantage of "D21 – Supporting Tools Requirements" in order to link the methodology with supporting tools.

The document involves notions defined in "D32 – Methodology Standards" and is applicable for "D41 – Supporting Tools Specification".

The document comes as a consolidated version after the WP6 – XIRUP methodology and supporting tools validation and evaluation phase.

The validation of the methodology has demonstrated how the set of method fragments defined in the first version of Methodology Specification [D31], where each method fragment represents a well isolated engineering process for model-driven modernization, can be integrated in the MOMOCS case studies by Telefonica and SIEMENS without additional changes.

The current version of the document includes two main major chapters that detail the above mentioned results: XIRUP Walkthrough and Tool Interfaces.

1.2 Document Structure

Chapter 2 reviews the requirements that were specified in [D11] and [D12], to state the scope of XIRUP methodology. These requirements and the study of the state of the art (see [D13]) have contributed to the definition of an iterative, feature-driven, model-driven and component-based modernisation methodology, whose basic principles are explained in chapter 3. The XIRUP modernisation process is specified by using SPEM in chapter 4. It is interesting to note that XIRUP states methodological principles and a generic process for complex systems modernisation, but this can be particularized for specific cases, as chapter 5 illustrates with two case studies. Finally, chapter 6 discusses the accomplishment of requirements by XIRUP.

2 Requirements Analysis

This section provides a summary of the main drivers for definition of the XIRUP process. This analysis is based on the requirements specified in [D11] and [D12].

The table below briefly lists the most important drivers for the XIRUP process. The column "drivers" gives important keywords extracted from [DOW], [D11] and [D12] that lead the definition of the methodology. Then the column "Understanding / Analysis" describes understanding of the requirement. References to the requirements are given in "Related Requirements" column.

Drivers for Methodology	Understanding / Analysis	Related Requirements from [D11, D12]
Generic methodology	The XIRUP methodology will treat modernisation problems in a generic way. In order to ensure its generality, the methodology definition is driven by two distinct use cases from industrial partners [D12]. Generality is needed to cover the usage in many engineering disciplines as required.	GR6, GR7, MR3, MR7, IND.2
Taking advantage of existing system	Modernisation implies existence of a system to be modernised. The process will start with gathering of knowledge about existent system and its analysis.	TELCO.9, TELCO.15

	During the modernisation, a possibility to reuse existent components will be considered.	
Agility	Optional and Mandatory steps, Process Components, extendibility.	[DOW], TELCO.6, TELCO.7, TELCO.25
Incremental and Iterative	The modernisation should be carried on in an incremental manner – feature by feature, functionality by functionality, and component by component.	GR2, GR4, TELCO.7
Model Driven Modernisation	The modernisation process will take advantage of model driven development techniques and approaches.	[DOW], GR3
User-centred	It is required that the modernisation process should focus on use-cases and features.	GR4, MR1, TELCO.1
Assuring of migration period	During the modernisation at some stage the new system may coexist with the old system. The modernisation process should alleviate the difficulties the situation may cause.	TELCO.8, TELCO.10

Table 1 Requirements Analysis

3 XIRUP Modernisation Methodology Principles

In the next sections the principles of the XIRUP methodology are presented.

3.1 XIRUP iterative and feature driven process model

In this section the generic XIRUP modernisation process is defined.

XIRUP process model defines several paths of iteration, as the SPEM diagram in Figure 1 shows. It consists of four main activities, which are refined in workflows. Note that apart of the first (Preliminary evaluation) the rest of activities may iterate over each other, and that these iterations are mostly driven by features.

A modernisation starts with a *preliminary evaluation* whose purpose is to make an estimation of the cost of modernisation and decide whether the modernisation is going to take place. This activity involves acquiring knowledge of existing system and the purpose of modernisation in order to prepare an estimation of cost for modernisation, preferably using some tool. This is easier when the modernized system is going to be implemented on an existing framework, as it is usually the case. The estimation of the cost can be done by identification of the features to be modernized. This would allow also preparing a tentative plan for the modernisation. Preliminary Evaluation represents an extremely important phase within a XIRUP process instance since it has the goal to provide the company with a concrete evaluation on pros and cons about modernizing a system: if this phase is not well-performed, the modernisation will start with weak fundaments, generating time, effort and cost consuming activities without a real need.

The next activity consists of *understanding* the system and the requirements of modernisation with greater level of detail than in the preliminary evaluation because the company has already decided to go on with modernisation. In the case of complex systems, this can be done in iterations. At each iteration, some feature or features of the system are taken into account. Understanding refers both to the existing system as well as the modernized system. The engineer needs to understand how the existing system is and which components are related with the feature(s) under study, in order to identify components for the modernized system, and their relationships. A tool that facilitates the description of existing components and assisting in the selection of target components improves the performance of this activity.

With the components for modernisation, the *building* of the modernized system can be performed by establishing the transformations from components in existing system to components of the modernized system. These transformations, once defined, have to be checked for consistency with the rest of transformations. Then, they can be applied to get the implementation of the modernized components, which have to be tested for their validation.

Once a feature has been modernized, the corresponding components can be deployed and used, replacing existing components. It is also possible to wait for other features to be ready to make the deployment. *Migration* is not a simple activity as the transition from the old to the modernized system has to be smooth and get end-users acceptance, with minimum disturbance of their normal activities. Another issue here is that migration requires, quite often, the coexistence of components of old and modernized systems. Migration considers not only deployment of new

components, but also the migration of data from existing to modernized system, depending on components. This data migration could be supported by tools.

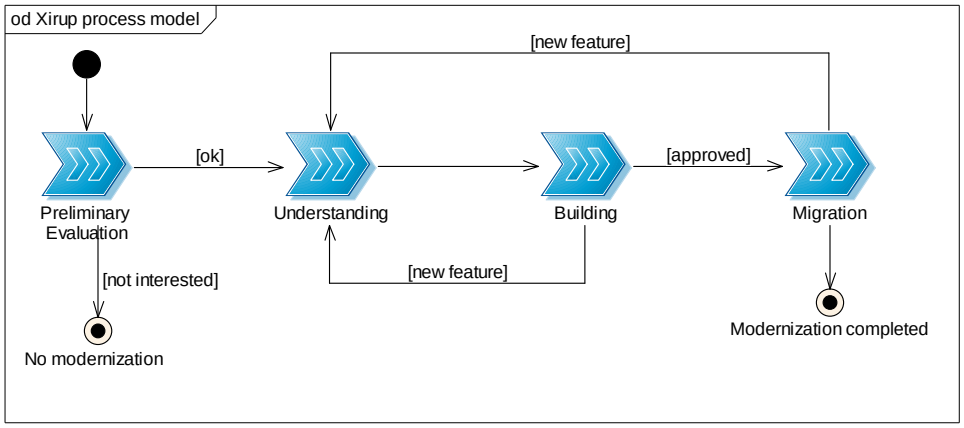

Figure 1 XIRUP process model

Transitions in the activity diagram occur when an activity has finished. Here, when preliminary evaluation has finished, the customer should decide whether or not to perform the modernisation and which features to consider. The result of understanding is the identification of components to build for the modernized system. These can be components from a framework, which have to be configured (e.g. by transformation from existing components information), components that wrap existing components to facilitate their integration with the infrastructure of the modernized system, or components that need to be build from scratch (but with a clear specification of their functionality and interfaces). After building, if the resulting components do not pass validation tests, their implementation has to be reviewed. Component validation refers to the satisfaction of functional and non-functional requirements. Tests should consider both aspects, for instance, that the component is able to perform certain operations (functional requirement validation) and that it is able to support a certain throughput (non-functional requirement validation). Although testing is normally envisaged at implementation level, it is

possible to consider the use of verification and validation tools at model level, for instance, to verify some properties of the models or validate the compliance of the modernized system model with respect to the legacy system. If components are validated with success, then it is possible to consider another feature or start the migration with new components. Once the migration of these components has been performed, if there are pending features to modernize, it is necessary to come back to understanding activity.

3.2 XIRUP Motivation

Existing software process models need adaptation for the problem of modernisation. For this reason, XIRUP methodology is different from main-stream software development methodologies, although it may take some parts of them. An analysis of methodology requirements (see [D11]) shows the need for a new methodology.

With respect to Unified Process (UP), XIRUP process model is, generally, a shorter and more agile process. System requirements are better known (or easier to get) than in a typical software development process, where a system is built from scratch. In a modernisation, there is already a running system and this is something concrete to start with. The functionality of the system is well defined and, usually, easy to get (for instance, by observing its execution). New requirements are already known, as they are the reason for the modernisation: a technology change, integration of data or processes, some new functionality, etc. With respect to system architecture, the existing system has already one implemented. The modernized system usually will base on a well-proven architecture, therefore modernisation has to take into account transformation from components in one architecture to another. This is one of the issues that motivate the interest in model driven modernisation, as this allows to

work at a higher level of abstraction, more in line with the architectural view of the system.

The main concern of the modernisation engineer is how to transform and migrate the existing system into a modernized system. Although at the beginning the engineer should get knowledge of the existing system, and this is also an important activity, this is done in a different sense than in the UP.

Another important issue to consider is that the modernisation process, as well as the UP is iterative, but with the particularity that parts of the existing system and the modernized system may coexist for a time. Therefore the transition phase is not the same as in the UP.

As a result, the main phases defined in the UP (inception, elaboration, construction, and transition) get a different meaning, and their activities are organized in a different manner. We will see later in the description of the XIRUP process model that this is organized in phases that do not run sequentially as UP phases, and that there is not a one to one correspondence with UP phases (although XIRUP also defines four). Similar considerations can be made with respect to Basic UP and other UP derived methodologies. In other words, XIRUP doesn't follow the rigorous sequential approach that heavily characterized these methodologies.

With respect to agile methodologies, XIRUP, although it intends to be agile, is different because it manages a broader view of the system. XIRUP has concerns with system architecture, both for existing system and modernized. Also, building of the system is driven by transformations, it is component-based, and the main artefacts of the engineer are models. XIRUP shares with XP a high relevance for adaptability. But design comes first than implementation, which is, as far as possible, done automatically by applying model to model and model 2 text transformations to the

existing system. Thus, despite the agile style represents an extremely important XIRUP feature, this methodology is also guided by a sequence of predetermined and general phases that can be executed in an iterative and retroactive manner, depending on different concomitant factors such as the type of the system under modernisation (see the industrial cases) and company's software development style (or policies).

XIRUP takes also from feature driven development (FDD) the basic structure for iterations. Planning of the development process is driven by features. XIRUP adapts the basic principles of FDD to modernisation requirements. Features in XIRUP focus on some specific parts of the system to be modernized. Some are typical candidates:

- A components or application service with concrete functionality that is going to be re-placed by a set of components on the target platform.
- Change of support technology.
- New (functional or non-functional) requirement to be considered for the system.

In summary, XIRUP modernisation methodology is based on the following principles:

- **Highly iterative process, driven by features.** This allows for a smooth and controlled migration for an existing system to a modernized system, by minimizing risks and improving end-user acceptance. Iterations are also required since it is considered that the systems may be modernised part by part and thus the same modernisation process may be required to repeat for each of modernised parts. Another consideration is resource/time limitations

which may lead to partial modernisation of the system leaving some open issues for the future.

- **Model driven engineering.** A way to cope with system complexity is by abstraction. XIRUP proposes to work with models and move from code to models, models to models, and models to code by using transformations.
- **Component based modernisation.** Most current frameworks are component-based and XIRUP assumes that most target platforms for systems modernisation will be component-based. This facilitates encapsulation of legacy code for example by means of apposite wrappers and promotes feature-driven approach.
- **Extensive model based validation.** The methodology should be supported by tools that facilitate validation of models, transformations and testing of final implementation. Testing and validation are performed on the products of each activity to ensure quality of the modernized system.

These principles are described in the following sections.

3.3 Model Driven Engineering for Modernisation

XIRUP relies on model-driven engineering (MDE) techniques and practices, as those illustrated in Table 2.

MDD Activities	**MDD Techniques Applicable**
Architecture Recovery – rising level of abstraction	• Reverse engineering – Code-to-Model transformation also known as PSM recovery.

	• "Black-Box" modelling. • Manual modelling assisted by CASE tools.
Architecture Refactoring	• Model-to-Model transformations • Abstraction – eliminating platform dependencies for PIM creation. • Patterns, componentisation, wrapping, architecture restructuring, etc. • Transformation Definition • Manual modelling assisted by CASE tools.
Model Evaluation	• Model Analysis: • Metrics – Quality Analysis • Dependency – Clustering, Partitioning, Banding • Verification: • Deadlocks, Loops, Conformity, Performance, Security, Semantics • Simulation – Model Execution • Test Generation and Execution • Performance, Non-regression, Unit
Code Generation	• Model annotation – PSMs creation

	• Model-to-Code transformation • Model-to-Document transformation

Table 2 Applicable MDE Techniques

During understanding phase, XIRUP takes as input the available information of the legacy system (source code if available, documentation, interfaces specification, execution sequences) and builds a model. Usually, the model can have several viewpoints, to facilitate its management and understanding. During the building phase, XIRUP tools will allow to work with the model in order to build a model of the target modernized system. This process is supported by transformation tools, from model to model. The resulting model will be the basis for the application of model-to-code tools, in order to generate the implementation of the final system. A simplified view of this process is illustrated in the diagram of Figure 2.

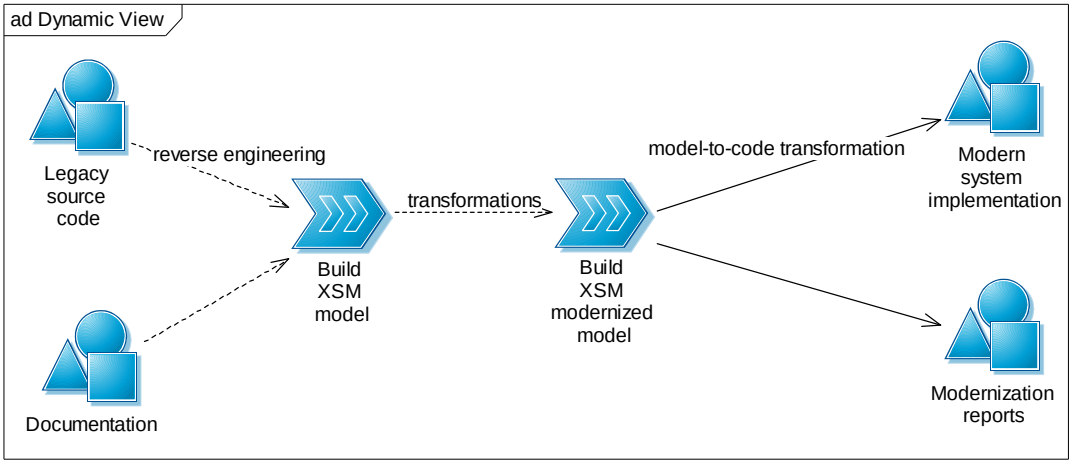

Figure 2 Working with models in XIRUP (simplified view)

There is also the possibility to define at model level transformations that are finally implemented as code-to-code transformations, especially when certain well-known patterns apply. For instance, when a legacy component is integrated in a modernized system by using a wrapper, the object adapter pattern can be identified at model level. As this is a well-known pattern, some specific transformation at implementation level can be used to generate automatically wrapper code from the definition of the interfaces of the legacy component and the required interfaces in the modernized system.

3.4 Component based approach

Software components are the basics of most current software frameworks. Therefore, XIRUP is considering modernisation of complex systems on component-based frameworks as target. There are several definitions of what a software component is, but most agree to consider it as a system element offering a predefined service and able to communicate with other components. In this sense, it is important to consider

the separation of component interfaces and implementation. A classical book on components [Szyperski 2002] identifies five criteria for what a software component shall be to fulfil the definition:

- Multiple-use.
- Non-context-specific.
- Can be combined with other components.
- Encapsulated i.e., non-investigable through its interfaces.
- A unit of independent deployment and versioning.

A simpler definition can be: *A component is an object written to a specification*. It does not matter what the specification is: COM, Java Beans, etc., as long as the object adheres to the specification. It is only by adhering to the specification that the object becomes a component and gains features like reusability and so forth. Software components often take the form of objects or collections of objects (from object-oriented programming), in some binary or textual form, adhering to some interface description language (IDL) so that the component may exist autonomously from other components in a computer.

But more generally, component based frameworks are associated to the concept of *container*. Containers aid in the construction of components, making it faster to develop than before, with less code, as they provide common functionality for certain application domains. So, a component may declare its default configuration, but include an expression that is able to obtain information from the system using it to populate it at runtime, without needing to know anything about the target application at design time. The target application may override the configuration entirely based on its own needs, or even based on other runtime environment factors.

In this sense, containers make easier also the configuration of components and their management.

In XIRUP, components facilitate the identification of features to be modernized and the separation of concerns. Also, some legacy parts can be componentised in the target framework, so they can be reused as they are, as a black box. This is illustrated in one of the case studies in section 5.1.

3.5 Notation – Software Process Engineering Metamodel

Software Process Engineering [SPEM] has been chosen for the XIRUP process definition, as it provides a convenient way to represent main elements of the development process. SPEM provides a graphical notation for definition of:

- **Roles** involved in the process;
- **Work products** elaborated during the process: models, code, documentation;
- **Workflow** – definition of activities and transitions between them
- **Guidances** – supporting information sources, such as applicable documentation or standards
- and **Tools** used to assist to the activities: CASE tools, compilers and etc.

For more information about the metamodel please refer to [SPEM] and D3.2.

In addition, we use the notion of "process fragments" or "process components" as per SPEM 1.1. This mechanism allows defining an isolated subset of methodology treating a particular activity. The goal is to provide users with a "palette" of model based process for the system modernisation. Respecting the agility of the XIRUP, this

"palette" may be extended with more process – contributions from other research projects in MDE area, like ModelWare or ModelPlex. The users are then invited to customize the modernisation methodology using predefined, well-elaborated methods.

4 XIRUP Process Specification

This section specifies the XIRUP process using SPEM notation.

4.1 Scope of XIRUP Process

The "Description of Work" [DOW] and user requirements [D11, D12] state that a complete model-driven modernisation process should be specified. The work done for the generalisation of current practices for modernisation shows that engineering processes are widely setup in companies. From our experience we may state that the adoption of new engineering processes is extremely difficult in big companies as Telefónica and SIEMENS are. The adoption is simpler when new concepts are introduced as an upgrade of current engineering process and are well isolated.

In addition, the engineering processes vary a lot from company to company taking into consideration different factors: their engineering domain (software, systems), economical sector (telecom providers, airport systems manufacturer, etc.), risks to be faced by the developed systems (mission critical or low risk systems) and etc. Taking into account these considerations companies chose a model that fits their needs, such as different types of waterfalls [WFALL], V-model [V], Spiral model [SPI], Unified Process [RUP] or Agile [XP]. Providing a modernisation engineering process covering all domains would result in a very generalized process having little applicability.

In order to cope with the abovementioned difficulties, we propose the following solution:

- we define a set of method fragments (in line with Situational Method Engineering techniques [KumWel92]). Each method fragment represents a well isolated engineering process for model-driven modernisation;
- we will demonstrate how these fragments can be used as a basis of a complete agile engineering process called XIRUP;
- we will illustrate how the method fragments can be integrated in the MOMOCS case studies by Telefónica and SIEMENS.

We will intentionally concentrate on the generic method fragments for model based modernisation, since, we believe, they represent one of the most significant added-value of the project.

4.2 Generic XIRUP Process

The section 3.1 defined the notions about the XIRUP process. The following phases were specified:

- **Preliminary Evaluation** – an early evaluation of the current system and modernisation problem;
- **Understanding** – in-depth analysis of the current system and modernisation problem by means of building of a modernised systems using components;
- **Building** – implementation of the system modernisation specified by the component model;
- **Migration** – transforming the components for a specific platform.

Taking the description of the applicable MDE techniques from the section 3.3 the following XIRUP method fragments have been identified, required to cover the generic activities:

- **Architecture Recovery** by reverse engineering;
- **Architecture Modernisation** by model-to-model transformations;
- **Transformation Definition** to assist in transformations
- **Model Evaluation**:
    - Model Quality Analysis by metrics evaluation;
    - Model Verification;
    - Model Simulation by model execution;
    - Model Testing;
- **Code Generation** by model-to-code transformations.

The following table illustrates the fragment usage in the XIRUP process.

XIRUP Phases	**XIRUP Method Fragments Usage**
Preliminary Evaluation	Architecture Recovery for building a snapshot of the system to be modernised. Model Evaluation for preliminary analysis of the current state of the system and the modernisation problem. It may also drive the decision to modernise the system. Architecture Modernisation and Code Generation for building prototypes of the modernised system.

Understanding	Architecture Recovery for building a model of the system for further analysis. Architecture Modernisation and Transformation Definition for creating a component model of the modernised system. Model Evaluation for in-depth analysis of the current and modernised systems. Code Generation for building prototypes.
Building	Code Generation for building required components. Model Evaluation for testing the generated components
Migration	Code Generation for building required components for specific platforms. Model Evaluation for testing components in the platform environment.

Table 3 Method Fragments in XIRUP

4.3 XIRUP Method Fragments Details

The following section describes process fragments covering most of the **MDD techniques** used in complex system modernisation. These components should form a "palette" to be used for customisation of the XIRUP methodology and can be applied separately or within the existing software engineering process.

Please note that the roles and tools used for description are abstract and should be specialized for each concrete scenario.

Some of the functionalities mentioned below will be implemented by XIRUP tools defined in [D21].

4.3.1 Recovering Architecture of Existing System

In order to be capable to modernise a legacy system, it is essential to understand it. The common way to simplify understanding is raising the level of abstraction of the system representation. In MDE the model allows to analyse the system as a whole, build different viewpoints and separate concerns. The legacy system may have already the model available, however, in many cases the model has to be built from legacy system artefacts like source code, documentation or logs (e.g. for black box modelling). The following process fragment was elaborated to explain the workflow for the architecture recovery.

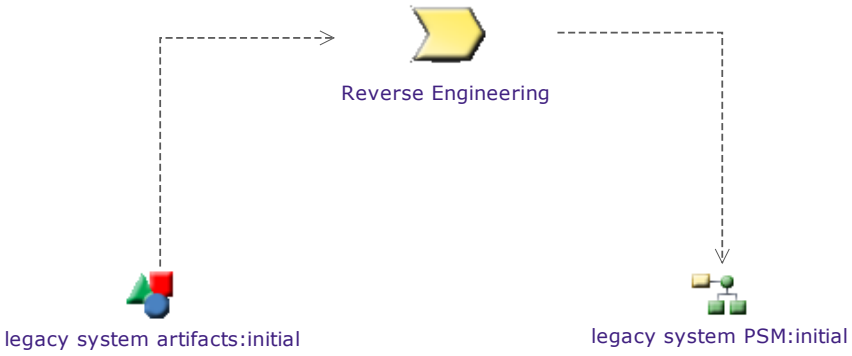

Figure 3 Architecture Recovery Workflow

Figure 3 depicts a generic workflow for architecture recovery. Using reverse engineering techniques (code and logs analysis, transformations of ASTs and documents) the legacy system artefacts are transformed into a model very close to

the platform – PSM. This PSM is then subject for the application of other MDE technique described in the later sections.

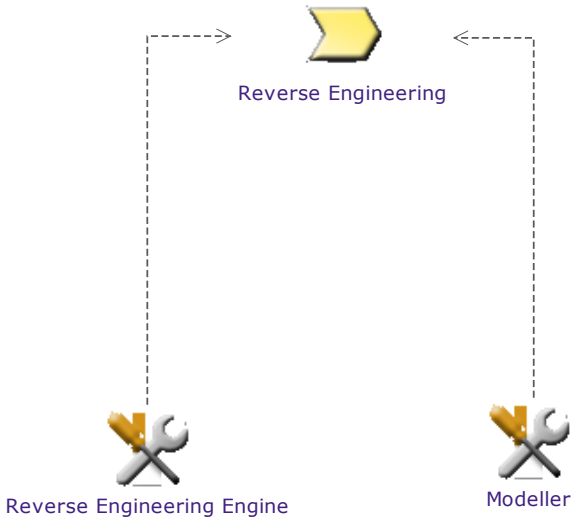

Figure 4 Architecture Recovery - Tools

As it is shown in Figure 4, the architecture recovery may be automated reverse engineering tools like UML Case tools (Objecteering, IBM RSA, Borland Together and etc.) , while some more manual modelling will most probably required and a modelling tool will required to build the PSM model.

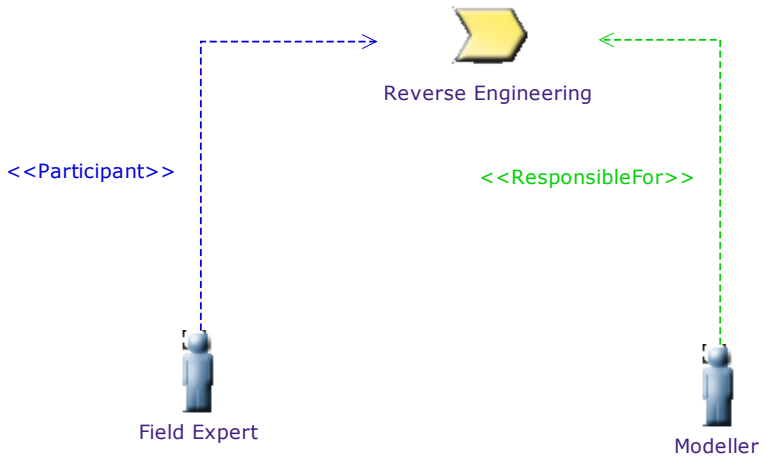

Figure 5 Architecture Recovery - Roles

In this process two roles have been identified (Figure 5), the Modeller is an MDE engineer that actually creates the model, while in many cases a Field Expert participation may be required to provide a deep knowledge of the subjected system.

4.3.2 Architecture Modernisation

When the architecture modernisation task is required, the Model-to-Model Transformation techniques are usually applied in MDE. These transformations may be used to apply specific architectural patters - best engineering practices treating concrete problems like MVC for GUI development or componentisation by wrapping and etc. "Abstraction" transformation may be required for building a PIM from PSM. Some specific transformation may be required to needed for rearranging the architecture, e.g. layering or repackaging. Adding new elements like components or

classes implementing particular features is also a typical architecture modernisation task.

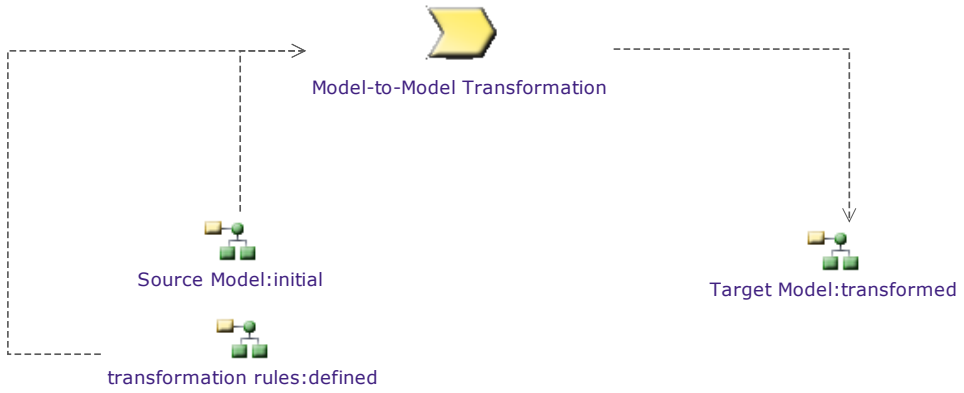

Figure 6 Architecture Modernisation Workflow

Figure 6 depicts a generic workflow for transforming the initial system architecture to the modernized system. It should be taken into account that usually the modernized system relies on a well-proven architecture (for instance, an existing framework where the modernized system is to be implemented).

The source model is transformed into the target model according to the predefined transformation rules. For the transformation definition, please refer to section 4.3.4.

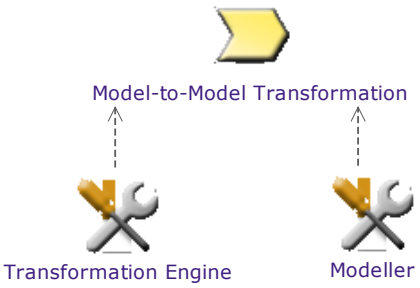

Figure 7 Architecture Modernisation – Tools

For Model-to-Model transformation (Figure 7) specific engines (QVT, ATL) may be applied to automate transformations. In addition, the model editing tools are usually required in order to refine models after transformations or perform the transformation manually.

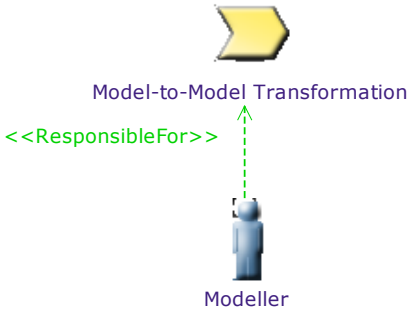

Figure 8 Architecture Modernisation - Roles

As it is shown in Figure 8, MDE engineer is involved to fulfil this task.

4.3.3 Generating Code

Automated code generation is actually the feature that allows keeping models the centre piece in MDE. The technology allows code generation for such object oriented platform as C++, Java, CORBA, J2EE, but also with specific profiles for FORTAN, C or other platforms.

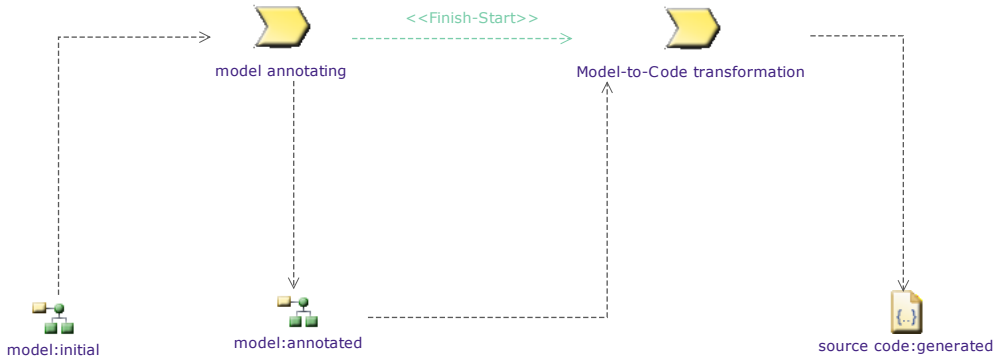

Figure 9 Code Generation Workflow

Figure 9 depicts a generic workflow for code generation. Model Annotation is actually a kind of Model-to-Model transformation in which a PSM is created. In other words initial model is annotated with a PSM metamodel notation, for example, with UML2 Profile for Java. Tags and stereotypes are added to the model to help in generation specific targets, e.g. Javadoc comments, specific modifiers for methods or attributes, even excerpts of specific code in special type of comments. Then the Model-to-Code transformation is applied, which generates the source code.

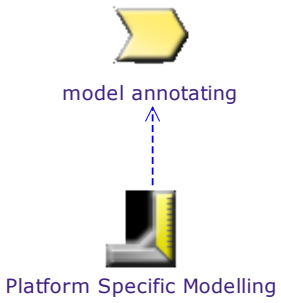

Figure 10 Model Annotation Guidance

The model annotation process can be automated, but also may require users to intervene. In this case, as it is depicted in Figure 10, specific guidance may be required for helping users annotating the code. The guidance may be standards describing concrete programming languages, coding conventions and etc.

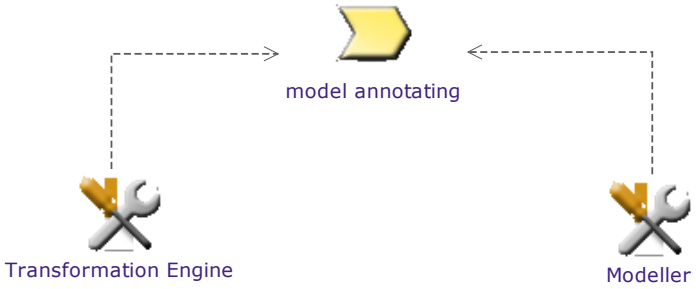

Figure 11 Model Annotation - Tools

Figure 11 describes that for automated annotation Transformation Engines can be applied to create PSMs from PIM. For many CASE tools this transformations are transparent for users – the tools seamlessly switch the metamodel when a user

decides to switch from modelling to the development mode. In many cases, the Modeller tool is still required to manually refine the resulting PSM model.

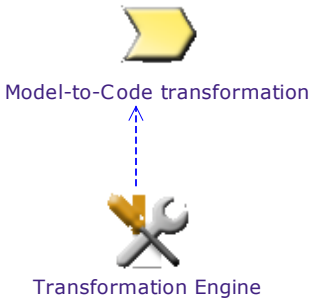

Figure 12 Model-to-Code Transformation - Tools

The transformation from a model-of-code to source code (PSM-to-code) is generally fully automated. As it is depicted in Figure 12, it requires an appropriate Transformation Engine.

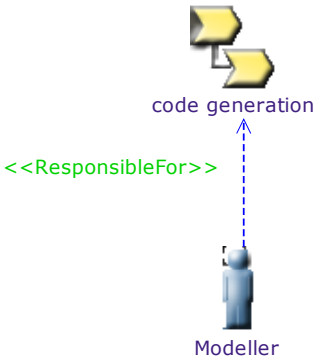

Figure 13 Code Generation - Roles

Figure 13 shows that Modeller role is in charge of code generation task.

4.3.4 Defining Transformations

In the principles of MDE the focus from source code elaboration is moved to elaboration of models and their consequent transformations. The effort to develop a transformation is actually comparable with an effort required for building the application itself. However, the motivation for developing transformation instead of the code is the possibility to reuse transformations several times.

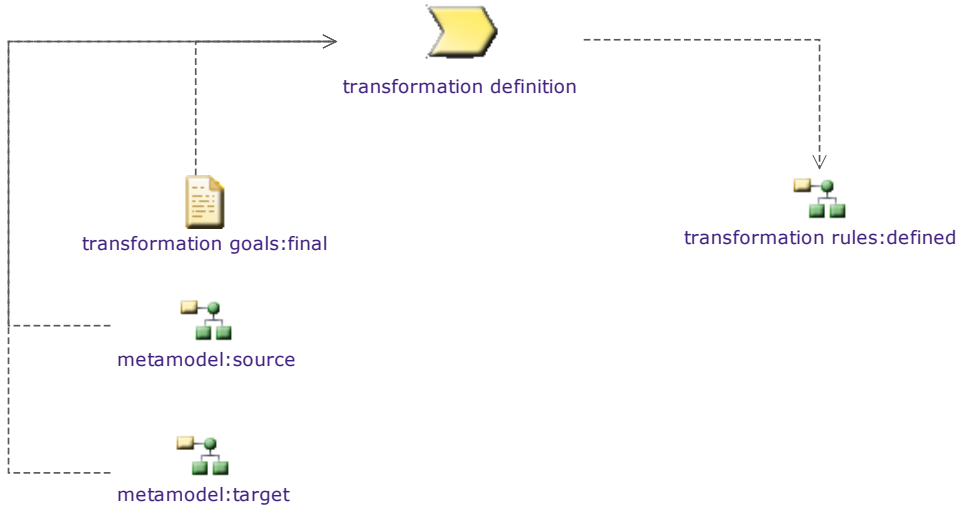

Figure 14 Transformation Definition Workflow

Figure 14 depicts a typical process for transformation definition. The transformation goals should first be elaborated to define the objectives of transformation. The process requires the knowledge of source and target metamodels. The task results in a model of transformation rules in transformation language notation.

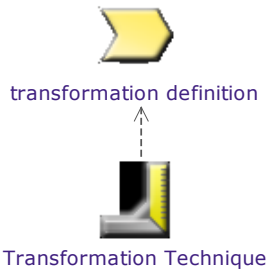

Figure 15 Transformation Definition - Guidance

Figure 15 depicts guidance for the transformation definition – standards and conventions applicable for transformations. OMG proposes standards QVT and MOF2Text as notations for transformation definition. However, other techniques like ATL or triple-graph grammar can be applied as well.

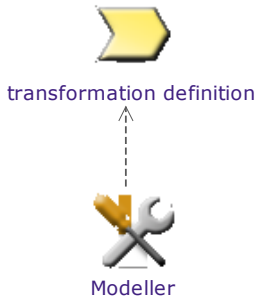

Figure 16 Transformation Definition - Tools

The transformation rules are modelled with a tool (Figure 16) that allows editing in the language chosen for transformations.

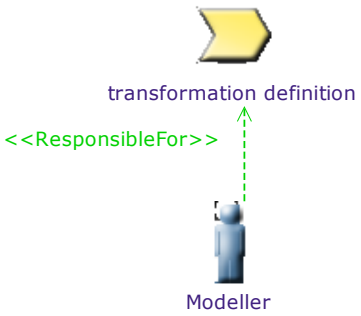

Figure 17 Transformation Definition - Roles

Figure 17 depicts the role involved in the process. Generally, since the task requires deep knowledge in the subject, metamodel experts are involved in transformation definition task.

4.3.5 Model Evaluation

This section presents several common method fragments for model evaluation. Please note that the list is not exhaustive, though presents the current state of practices.

The model evaluation is quite broad domain. Several groups of techniques can be listed depending on the goal of the analysis. The techniques are used for detection of error patterns on the modelling phase and thus allow to avoid further escalation on the implementation phase. The features provided are:

- Syntactical checking on run-time;
- Semantics checking on run-time and during the post-analysis;

- Conformity, performance and security verification;
- Model quality analysis based on metrics evaluation;
- Model based test generation;
- Simulation by Model Execution.

All this techniques can be used to setup objectives and finishing criteria for the modernisation process.

The following sections some details of the above-mentioned processes are given.

4.3.5.1 Model Quality Analysis

The model quality analysis permits to provide a report on subsystem dependencies, as well as reusability and maintainability parameters. The metrics are the basis for definition of a quality model. According to the ISO 9126 standard the following major characteristics reflect the software quality:

- Functionality;
- Reliability;
- Usability;
- Efficiency;
- Maintainability;
- Portability.

Each of these characteristics is a set of attributes that bear on different software capabilities. In the metrics based analysis, these attributes are represented by factors,

like “complexity”, which in turn calculated on sets of different metrics like “number of parameters passed by value” or “ratio of repeated inheritances in the application”. Insufficiency in model quality can drive the modernisation activities.

In this section we present a process component for model quality analysis based on metrics evaluation.

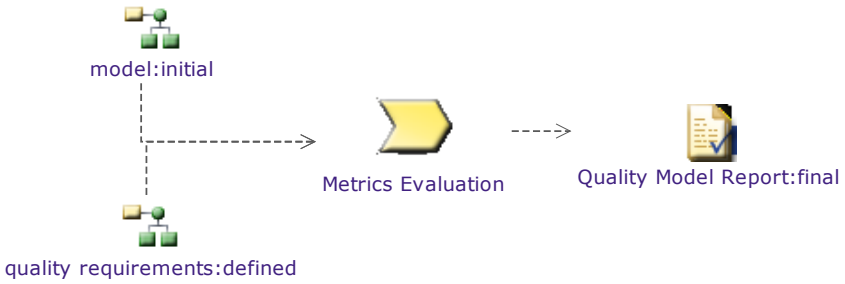

Figure 18 Quality Analysis Workflow

Figure 18 depicts a generic workflow for model quality analysis. The quality requirements, i.e. boundaries for metrics, should be provided as well as the model itself. The process results in quality model report containing the calculated metrics and their analysis according to provided boundaries.

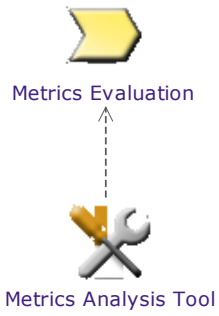

Figure 19 Quality Analysis - Tools

The metrics calculation and analysis process is well automated with the tools (Figure 19).

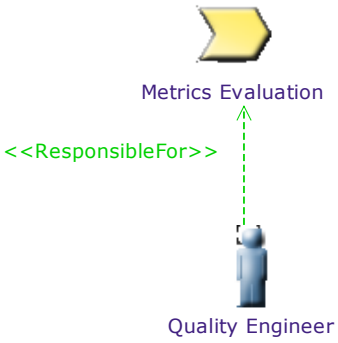

Figure 20 Quality Analysis - Roles

Quality Engineer (Figure 20) should be involved in the process, in order to correctly setup the metrics boundaries and to interpret the analysis results.

4.3.5.2 Model Dependency Analysis

The formal methods for graph dependencies analysis are also applicable for models. The analysis may be performed at run-time for early loops detection as well as at the post analysis phase to detect partitions, clusters, bands. The modernisations activities can also be driven by this analysis.

Figure 21 Dependency Analysis Workflow

Figure 21 depicts a generic workflow for dependency analysis. The initial model is subjected to dependency analysis techniques and a report is elaborated.

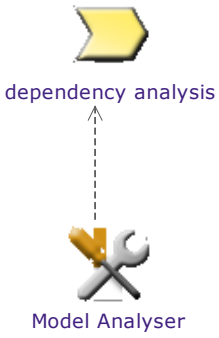

Figure 22 Dependency Analysis – Tools

This type of analysis is formal and well supported by tools. In this way a specific analyser tool (Figure 22) is involved to the process.

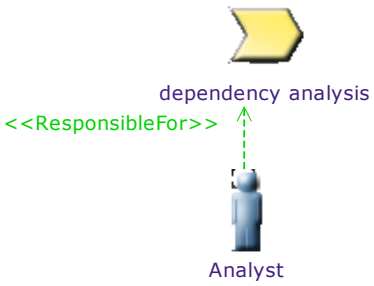

Figure 23 Dependency Analysis - Roles

Involvement of analyst (Figure 23) is necessary to interpret the results.

4.3.5.3 Model Verification

The verification techniques can be preformed in run-time as well as during the post analysis. They allow syntactical and semantics checking; deadlocks, bottleneck, loops detection; patterns and anti-patterns recognition; models conformity; verification for security and performance.

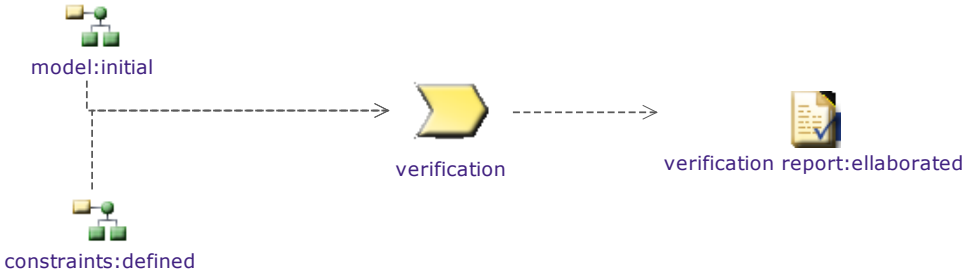

Figure 24 Model Verification Workflow

Figure 24 depicts a generic workflow for model verification. The model is subjected to verification according to the provided constraints. In result, a verification report is elaborated providing evaluation of model compliance to the constraints.

Figure 25 Model Verification - Tools

The techniques are generally well automated and require a checker tool (Figure 25).

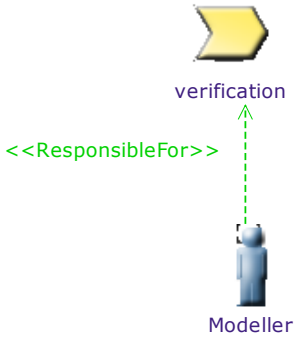

Figure 26 Model Verification - Roles

A modeller (Figure 26) role is involved in the process in order to specify the input work products and interpret the results.

4.3.5.4 Model Simulation

Model Simulation techniques allow estimation of the system behaviour by executing the model. The technology permits to simulate the contingency situations and can be used for test definition of the modernised system.

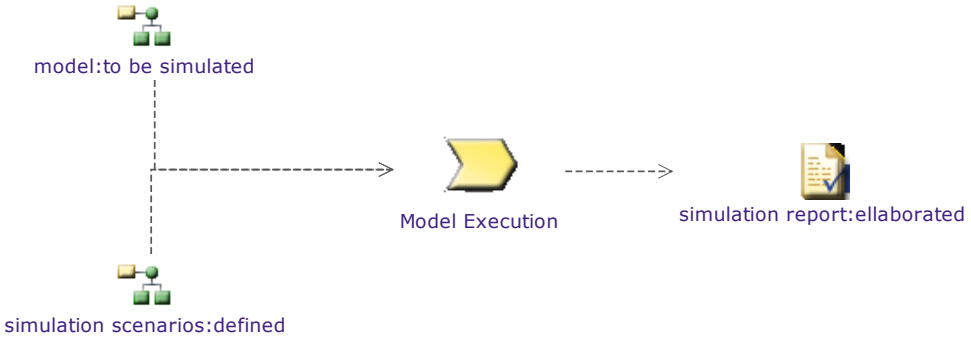

Figure 27 Model Simulation Workflow

Figure 27 depicts a generic workflow for model simulation by execution. For execution a model is annotated according to the simulation scenarios – definitions of simulated contingency situations and simulation execution parameters. After the scenario executions an analysis report is generated.

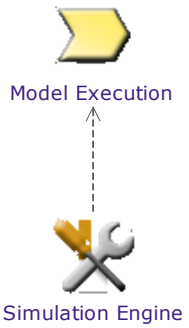

Figure 28 Model Simulation - Tools

For model execution a simulation engine (Figure 28) is required. It allows running the scenarios and visualising simulation results.

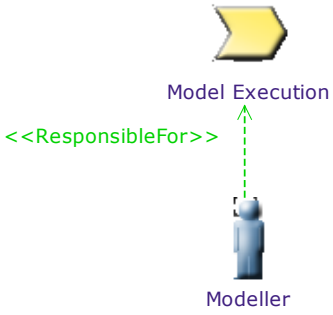

Figure 29 Model Simulation - Roles

Modeller role (Figure 29) is involved in this process in order to define the simulation scenarios, execute them and interpret the simulation results.

4.3.5.5 Model Testing

System testing can also be defined on the model level. U2TP and Eclipse TPTP allow expressing the tests using the system models. Different categories of tests can be specified: unit, non-regression, performance and security.

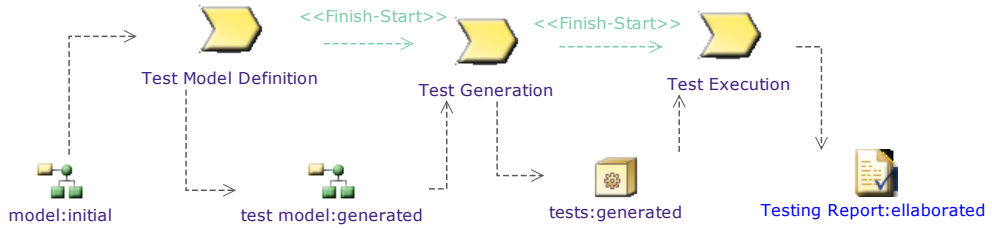

Figure 30 Model Testing Workflow

Figure 30 depicts a generic workflow for defining and executing of tests. First, a test model is defined specifying the test scenarios. After that, tests are generated for concrete testing platform (JUnit, Nunit and others). After all, the tests are executed and a test report is elaborated.

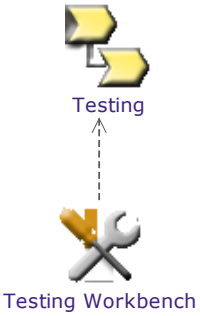

Figure 31 Model Testing - Tools

Generally, a testing workbench (Figure 31) is required for elaboration of testing model, generation of tests and their execution.

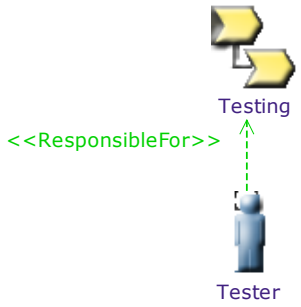

Figure 32 Model Testing - Roles

An experienced user, called tester (Figure 32) should be involved to specify the test model and elaborate the testing results report.

4.3.5.6 Requirements Traceability

This section presents a model based approach for requirements traceability.

Let us consider a typical MDE process depicted in Figure 33. MDE suggests to transform requirements into UML Use Cases. Then, from Use Cases, they are transformed into Conceptual Classes, which in turn are transformed in technical Design Classes. Finally, these classes are used for generating the code.

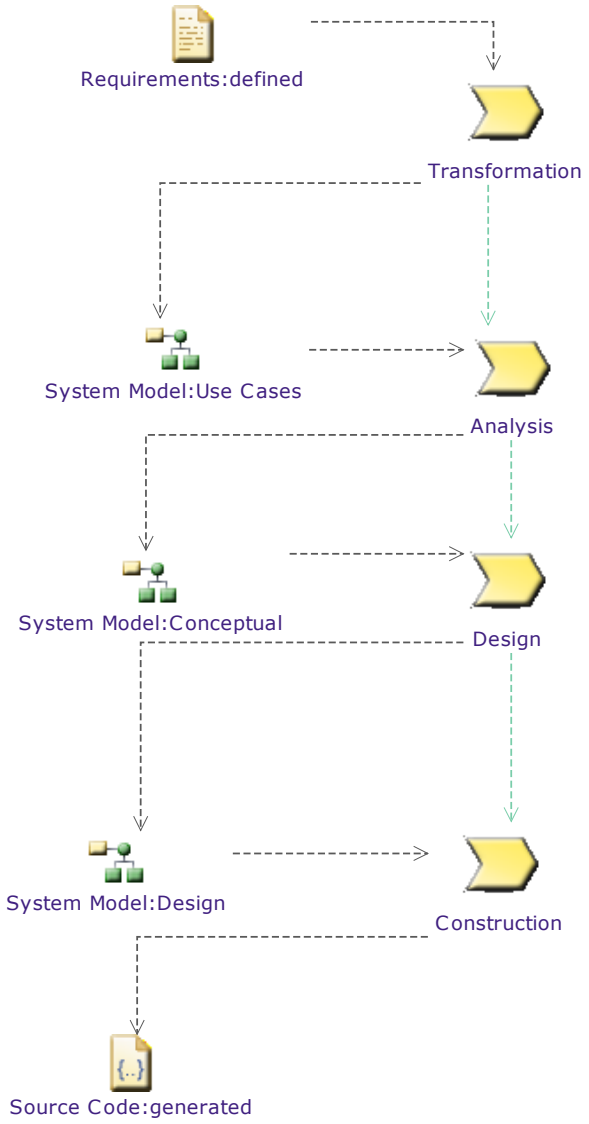

Figure 33 MDE Development Process Example.

The current approach for managing requirements traceability consists in linking requirements elements from Requirements Management Systems (Doors, Mantis and etc.) with UML elements. This is usually managed by a specific UML Profiles introducing "traceability links". While creating UML Use Cases from the requirements, the traceability links are established. The automatic transformations also take care of creating corresponded traceability links with newly created UML elements. In this way, at the final step, the generated source code is also linked with the requirements. Thus the complete chain of traces is established: Textual Requirements – Use Cases – Conceptual Classes – Design Classes – Source Code Classes. In accordance with the previous statement, the changes on any level of the process can be traced up to the requirements in order to check the consistency of changes and the dependencies with other requirements.

4.4 XIRUP Method Fragments Usage in a Custom Process

Figure 34 depicts a workflow example for a custom modernisation process using XIRUP methods fragments.

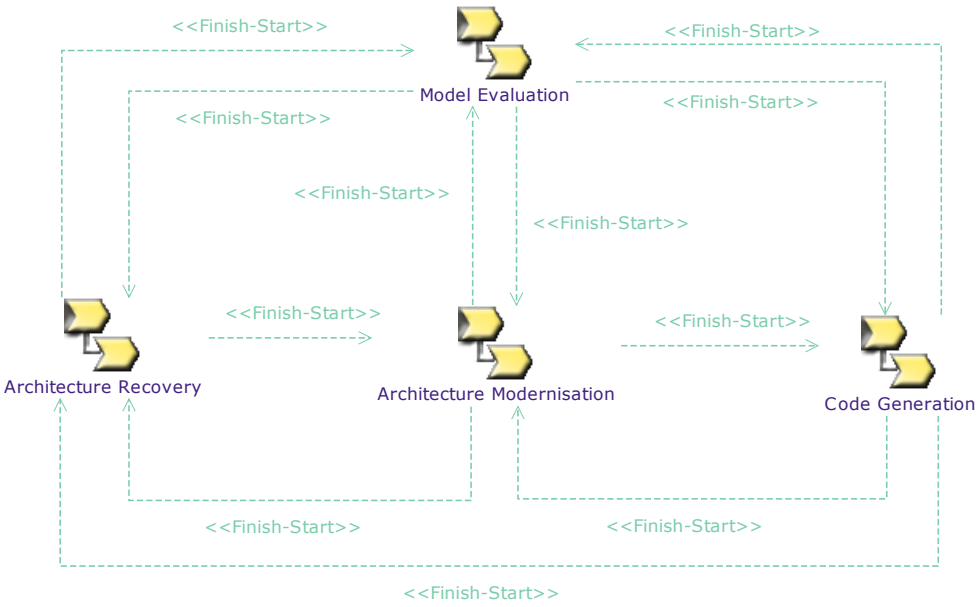

Figure 34 XIRUP Method Fragments Usage

The workflow starts with an **Architecture Recovery** from which the recovered model can be qualified with the **Model Evaluation** techniques and forwarded to the architecture modernisation task.

In the **Architecture Modernisation** task a new modernised model is elaborated which can be then subjected to a **Model Evaluation** techniques and the **Code Generation**. The process can return to the **Architecture Recovery** e.g. for recovering another part of the system necessary for modernising the system with another feature.

During the **Code Generation**, source files of the new modernized system are generated. The code may be tested using model testing techniques from **Model Evaluation**. The analysis may lead to regeneration of the code or to getting back to

the model level modernisation in the **Architecture Modernisation** task. Finally, getting back to the **Architecture Recovery** is allowed in order to get a new part of the system to be modernised.

The process can be stopped at any moment user decides the modernisation is finished.

4.5 Linking Abstract Tools with XIRUP Tools

In this section it is intended to present the mapping between the tools defined in [D12] and the abstract tools defined above for the XIRUP method fragments.

XIRUP Tools	**Abstract Tools**
K-Xtraction Tool	Reverse Engineering Engine, Modeller
XSM-Analysis Tool	Model Dependency Analyser, Model Checker
Constraint Editor Tool	Modeller
XSM Visualisation Tool	Modeller
XSM Transformation Tool	Transformation Engine

Table 4 Mapping between XIRUP tools and Abstract tools.

It was identified that one-to-one mapping is difficult to established, since the approaches for tool derivation was completely different. For the XIRUP tools, the requirements drive the tool specification, while for the abstract tools, the current MDE practices were reflected. However a tentative correspondence can be established as it is shown in Table 4.

As it is described above the K-Extraction Tool has properties of Reverse Engineering Engine and Modeller to be used in **Architecture Recovery** method fragment. The XSM-Analysis Tool should correspond to Model Dependency Analysis and Model Checker tools. The Constraint Editor functionalities should be present in Modeller as well as XSM Visualisation functionalities. The XSM Transformation Tool should be based on a Transformation Engine.

The XSM Type Library, Front-end and Requirements Elicitation tools have no corresponded abstract tools, since they are very case study specific.

5 Usage Examples for XIRUP

The XIRUP methodology can be particularized to different settings. This chapter illustrates this with two case studies by Telefónica I+D (Telco case) and Siemens (IND case).

5.1 Telco Case

5.1.1 XIRUP Interpretation in the Telco Case

5.1.1.1 Preliminary evaluation

In order to prepare an offer for modernisation, the solutions provider (SP) requirements engineer needs to maintain interviews with SME responsible. With these interviews, the SP requirements engineer can collect information that describes the existing system and about the purpose of the modernisation. These constitute a set of requirements, which should be stored in a requirements elicitation tool. These allow also making a first identification of existing components, which allow to prepare a first specification of the existing system, to be stored in K-Xtraction tool. Knowing the modernisation requirements and the components of the existing tool, a cost estimation tool can assist to prepare an offer for the SME. If this is not accepted, then the modernisation process finishes by leaving the system as it is. If the offer is accepted, the SP starts the *Understanding* activity.

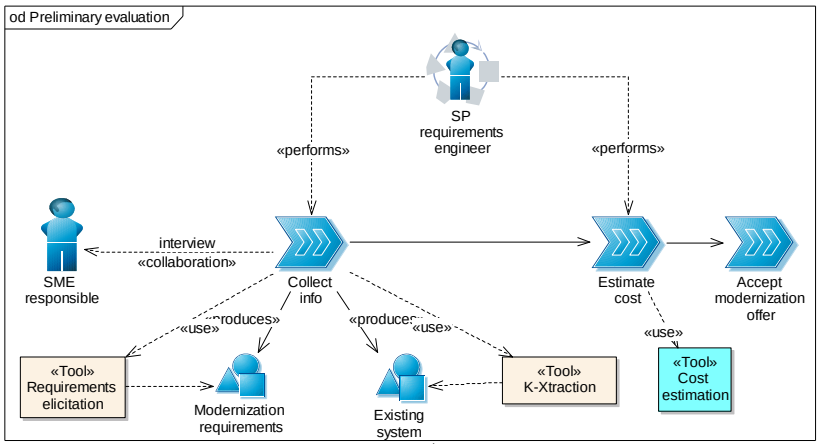

Figure 35 Preliminary evaluation scenario

The following XIRUP methods fragments are used for implementing this phase:

- **Architecture Recovery** fragment the is applied for getting an exhaustive view of the existing system in the Collect Info activity;
- **Model Analysis** fragment is to be used for modernisation cost analysis in Estimate Cost activity.

5.1.1.2 Understanding

Understanding the modernisation involves both analysing the existing system and the modernized system. The result is the identification of the components for the modernized system. The engineer has to build first a model of the existing system, by identifying components.

The process is iterative and is driven by features. A feature can be a modernisation requirement (e.g., need to move to a new infrastructure) or a component. Therefore, the engineer will consider one and identify involved components in the legacy system. With the description of these components, the XSM analysis tool, which

should have access to the description of components of the SP framework, is able to provide a list of components for the modernized system. These are generic components, and the engineer should decide which are those that best fit with the intended solution. The engineer should define relationships of those selected components and in this way a model of the modernized system is specified. This is stored in the XSM analysis tool, and is the input for the Building activity.

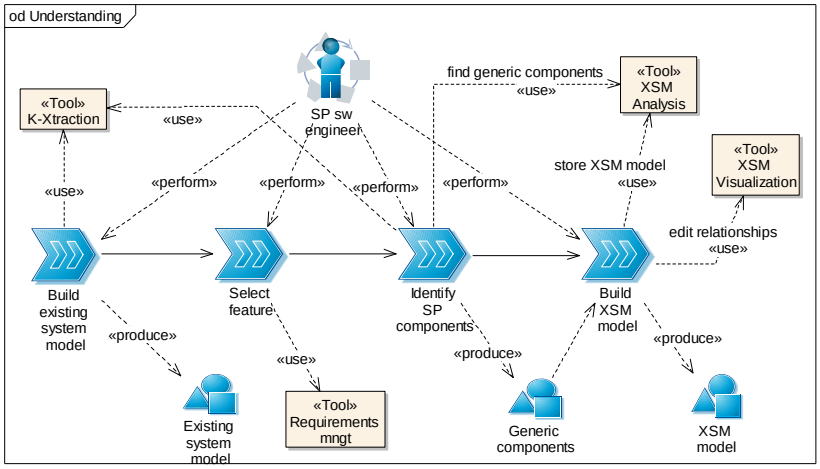

Figure 36 Understanding the modernisation

The following XIRUP methods fragments are used for implementing this phase:

- **Architecture Recovery** fragment is applied for getting an exhaustive view of the existing system during the Building Existing System Model activity.
- **Architecture Modernisation** fragment is to be used to during fulfilment of Select Feature, Identify SP Component, Build XSM Model activities.

5.1.1.3 Building

Building means customizing the generic XSM model. As components derived from understanding are generic, they have to be adapted, by configuration or by adding some code. This adaptation can be done by transformations. The input of the transformations is the information of components of existing system. The output is a set of deployable components.

Transformations have to be defined with the assistance of a XSM transformation tool, which allows to specify how elements from the existing system components are mapped into elements of XSM model components. The transformations have to be checked for consistency with other transformations and with constraints that may be defined for the use of XSM components.

Once transformations are validated, they can be applied. The resulting components have to be tested and then integrated. New tests have to be performed at a more general level (Test service). When these are validated, components are ready for migration.

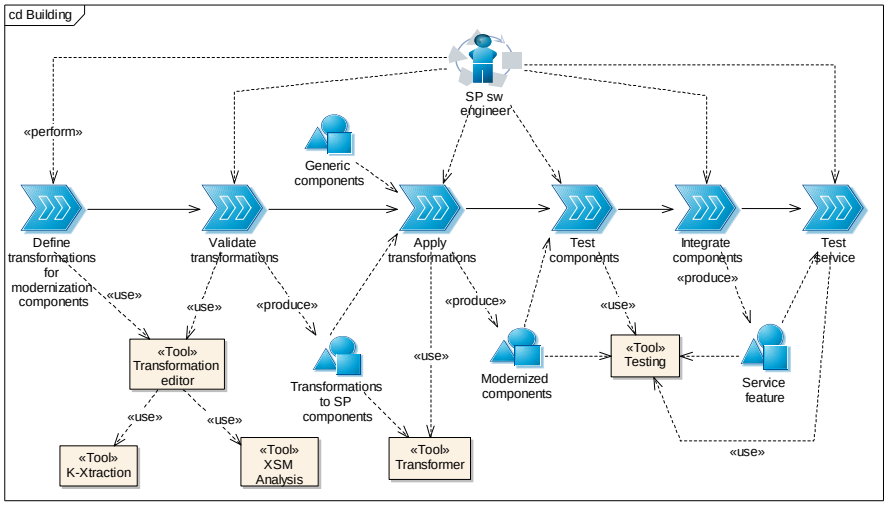

Figure 37 Building the modernized system

The following XIRUP methods fragments are used for implementing this phase:

- **Transformation Definition** fragment is applied for fulfilment of Define Transformation, Validate Transformation activities.
- **Architecture Modernisation** fragment is applied during Apply Transformations, Integrate Components activities.
- **Model Analysis** fragment is to be used for Test Components, Test Service activities.

5.1.1.4 Migration

Migration involves the deployment on specific platforms. Therefore, there is still a transformation to be defined from specific components to build a platform specific model. Usually, these transformations should be available in a transformation database as they are common for multiple developments (probably, for most common platforms, they are provided by commercial development tools).

By applying the transformations, the components can be deployed and the resulting system has to be again tested.

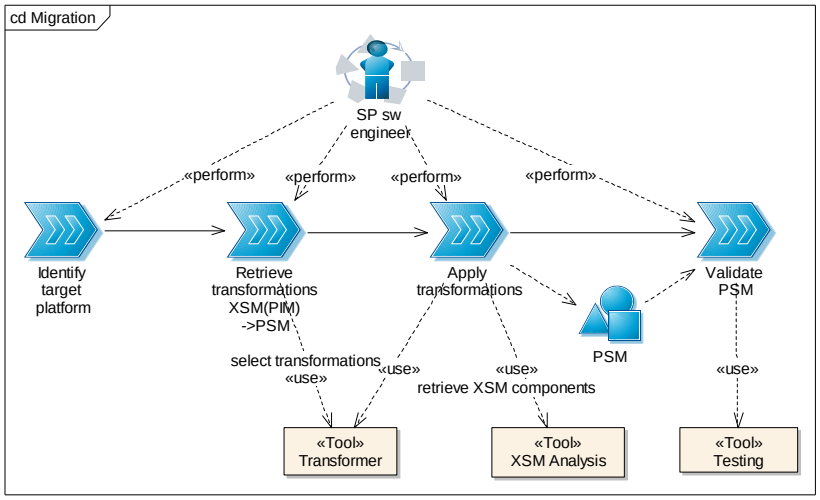

Figure 38 Migration

The following XIRUP methods fragments are used for implementing this phase:

- **Architecture Modernisation** fragment is used for transforming system models for a particular platform.
- **Model Analysis** fragment is used for validation of transformation and finally the system.
- **Code Generation fragment** is to be used for generating the source code for the target platform.

5.1.2 Telco Case Study Details

In order to illustrate usage of the above-described process the current section presents the details of the Telco case study.

Initially, the Telco case study considers a SME, with a computing infrastructure of isolated PCs for managing client list, employees, clients accounting, and billing. They have also a traditional telecom infrastructure of fixed and mobile phones. As support services they have some applications for accounting, clients, and billing. More details on data, process and xWare are given in the scenarios in [D12] The purpose of modernisation is to adapt this infrastructure and services to an integrated framework that is provided and supported by a Solutions Provider (SP). The SP has an infrastructure that allows adapting generic components to customers needs. There are generic components for secure access, CRM (Customers Relationship Management), EWFM (Enterprise WorkForce Management), location, and communications. The SP framework is facilitated by a service architecture, which is made up of software components and based on architectural patterns, at different levels: organization, control, resource, and basic components. This is described also in [D12].

In order to show the iterative and feature oriented approach of XIRUP methodology, and to consider modernisation in different settings, we are considering three concrete use cases that can be derived from the Telco case study. These are illustrated in the use case diagram of Figure 39 and will be described in the following sections.

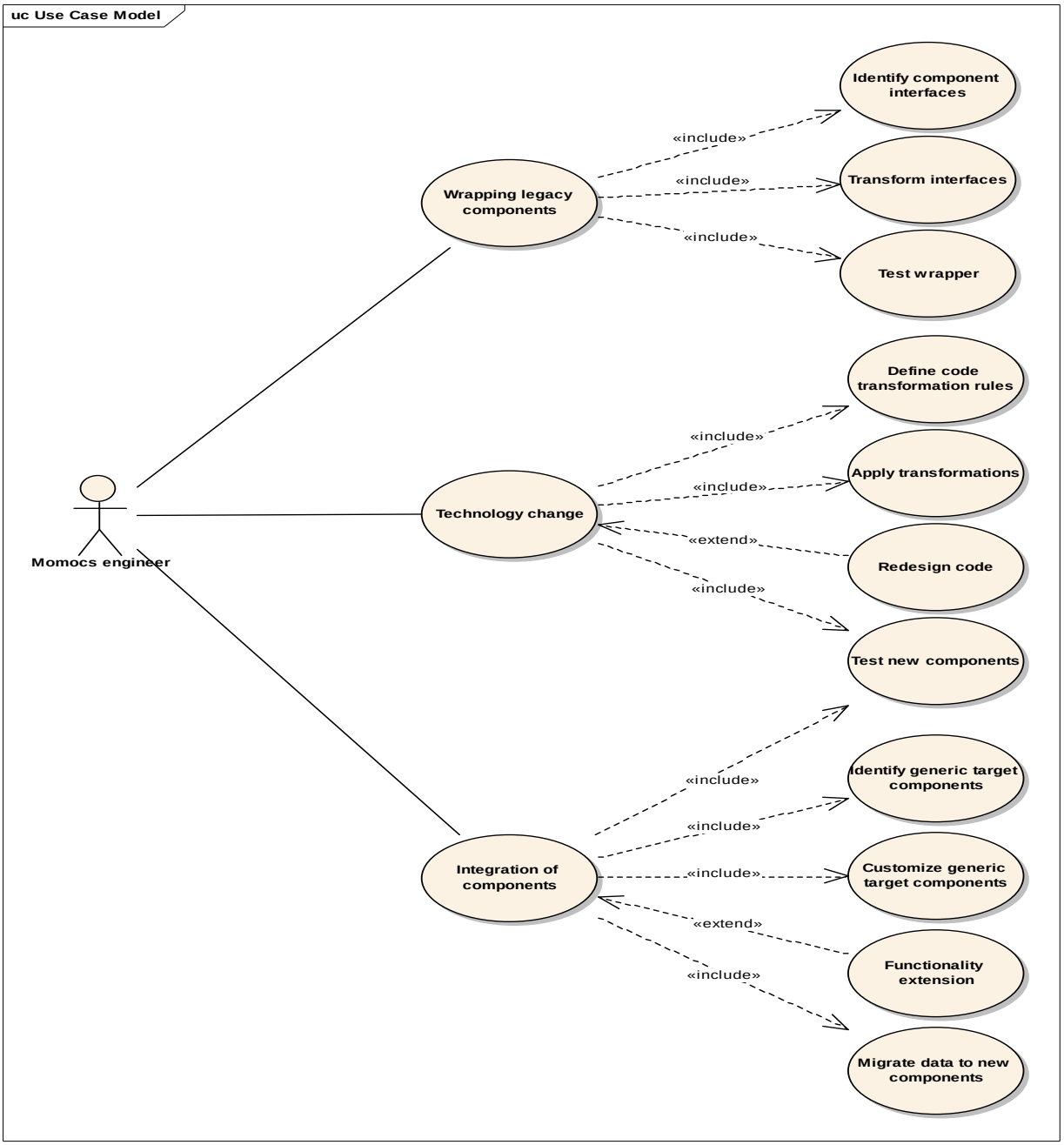

Figure 39 Telco use cases

The scenario for the first use case takes into account a concrete application running on SME premises, for billing. This application is based on open software, so source code is available. There are no billing components in the SP platform, and taken into consideration the characteristics of the application, the decision of the Momocs (modernisation) engineer is to wrap this application to integrate it in the SP framework as a resource. Therefore, the scenario considers how to extract a model of billing application services and how to build a wrapper according to SP framework as a resource. The input is the code of billing software and the SP resource specification. The output is code to wrap the billing software. Data do not need to be modified in this case.

The scenario for the second use case takes one of SP components, whose logic is currently build on ILOG JRules (a rule engine), and modernize it to an Open Source product called Drools (or JBoss Rules, see http://labs.jboss.com/portal/jbossrules).

The scenario for the third use case is the more complex as it considers several independent applications of the SME and tries to make an integrated access system. For this, it takes SP framework access components and configures these in order to fulfil the requirements from different applications. It provides a role based access system to different services of the modernized system.

5.1.2.1 The SP framework

An application in the SP framework is modelled as an organization made up of controller components (called *agents*) and resources. Therefore, there are two layers in the organization: the control layer, which is made up of controller components, and the resource layer, made up of the components that supply information or

provide some support functionality to the controllers to achieve their goals. The service's organization is shown in Figure 40.

The Control Layer contains two categories of controller components: *managers* and *specialists*. Their interfaces and internal structure are similar, however, they play different roles. Manager components are responsible for the management aspects of the service such as installation, configuration, activation, monitoring and exception handling. Specialist components are in charge of achieving the functionality of the service. Managers and specialists collaborate to accomplish their tasks during the whole service life cycle.

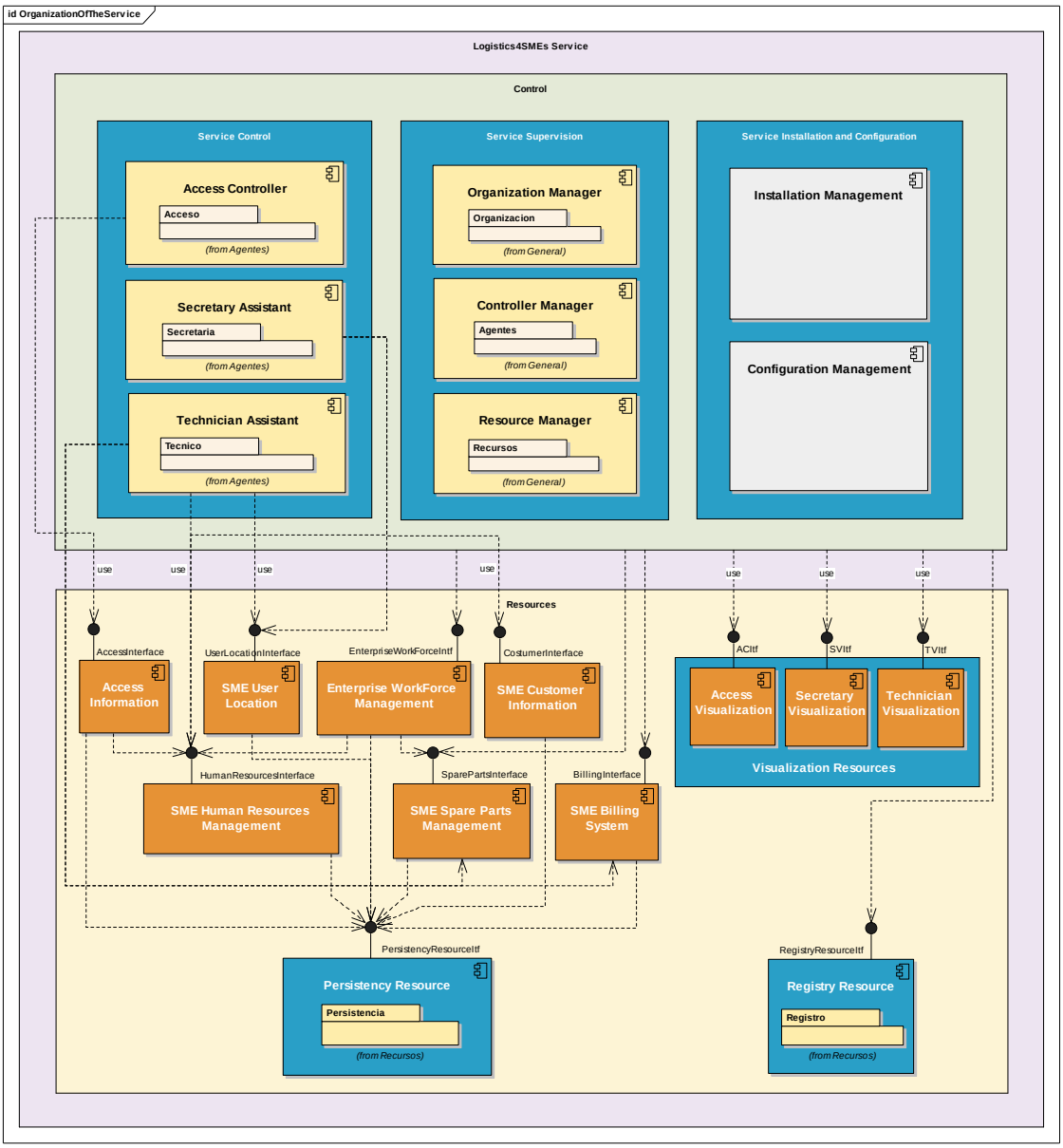

Figure 40 SP framework architecture

To highlight the role of each controller, the control layer is divided into three areas (as shown in Figure 40):

- *Service Control.* This area contains the components implementing the service functionality: Access Controller, Secretary Assistant and Technician Assistant.
- *Service Supervision.* This area contains the Organization Manager, the Controller Manager and the Resource Manager. The aim of the Service Supervision component is to manage the agents and the resources of the system.
- *Service Installation and Configuration.* This area contains the Installation Manager and the Configuration Manager which will provide service installation and configuration functionality.

The Resource Layer considers three basic types of resources, although others could be considered when needed. These are:

- *Persistency Resources:* provide object persistency through relational database management. They offer operational interfaces to store and recover application data.
- *Registry Resource:* this component is used to register and access the system's available web services.
- *Visualization Resources:* there are three Visualization Resources: Access Visualization, Secretary Visualization and Technician Visualization (PC and PDA sub-systems). Those resources UI are composed of presentation screens and user data acquisition.

More details on relevant components of this framework are described below when needed for the particular scenarios.

5.1.2.2 Modernisation by wrapping existing software

The scenario for the first use case considers a billing application that is running independently of the rest of the applications in SME premises. This application is Open Source, available at http://www.jbilling.com/. The purpose is to integrate this with the rest of the system by wrapping this application as a Resource of the SP framework.

The SP framework has a Resource Manager, which is the component responsible for controlling all the resources in the system. Each resource has a management interface, which is used by the Resource Manager, and a usage interface, with specific operations for the clients of the resource (see Figure 41). Therefore, the wrapper for the billing application will have to implement both interfaces.

In order to understand the requirements for this wrapper, in the following there is a description of the working of the Resource Manager, which is the client for the management of the resource, the billing application in this case, and a description of how usage interfaces should be in order to be integrated with the rest of the system.

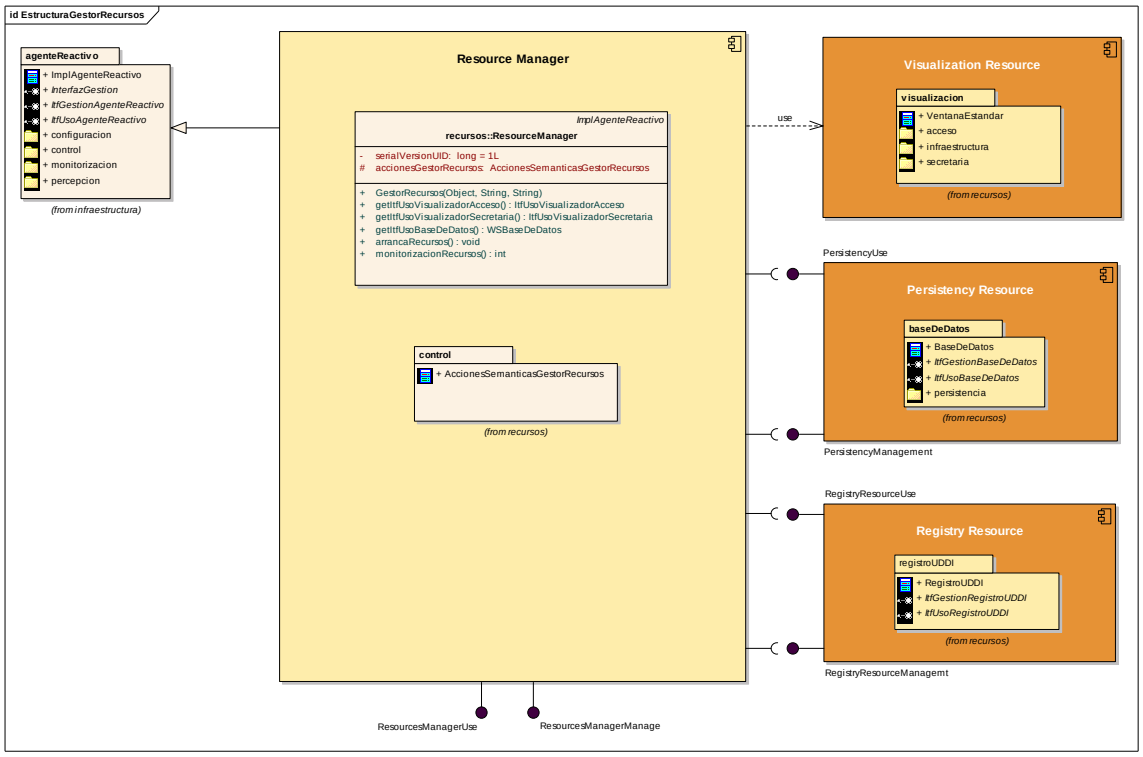

Figure 41 Resource Manager components and interfaces

The Resource Manager is implemented as a reactive agent and is responsible for the following activities:

- *System resources creation.* This component creates all the system resources. If an error occurs during the creation, this agent will inform the Organization Manager, which will decide what action to take. This is the first action this component takes after its creation.
- *System resources start-up.* After creating all the resources, this agent will order them to start running and wait for incoming requests.

- *System resources monitoring.* The Resource Manager has to check periodically every resource in the system is working properly. If it detects an error, it warns the Organization Manager, which is responsible for taking an appropriate decision.
- *System resources shutdown.* When required, this component sends a termination message to all the system resources. If an error occurs during this action, this agent informs the Organization Manager.

The Management interface is used by the *Organization Manager* to activate and monitor the Resource manager.

Use interfaces are offered to the resources to allow asynchronous sending of required information.

Behavior model

The behavior of the Resource Manager is defined by a Finite State Automaton (see Figure 42), which has five states: the initial state, the final state and three that represent the intermediate situations in which the automaton could find itself. In any change of state (known as transition), an action is triggered (generally as a method call). The first state indicates that the system is started and it is working fine. The second state represents the monitoring action the agent carries out to find problems during execution. The third state is achieved when an error occurs while monitoring.

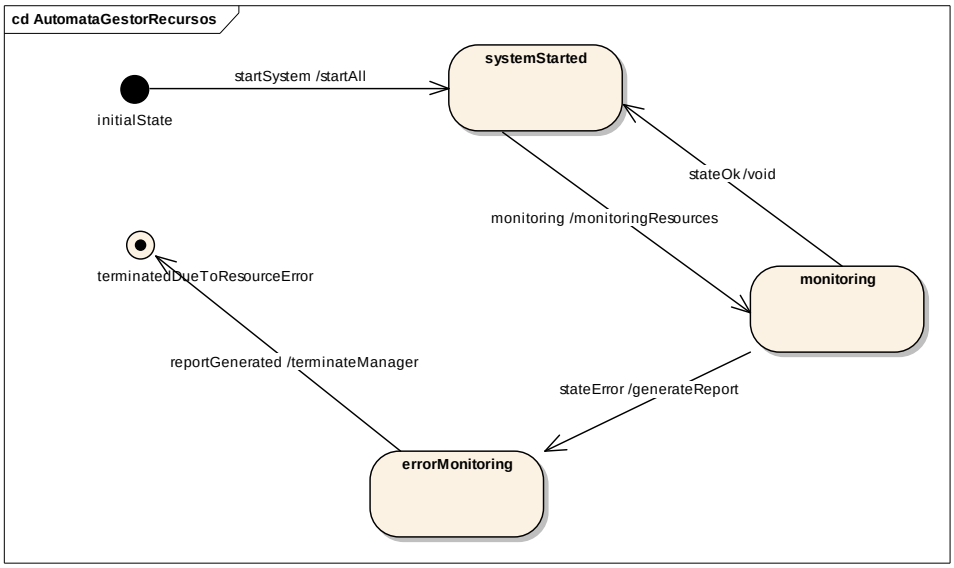

Figure 42 Resource Manager automaton

Usage interfaces for the wrapper of the billing application have to be extracted from the functionality provided by JBillling code. With this purpose, original source code is provided. This process can be conducted by some Momocs tool and supervised by the modernisation engineer.

In principle, the usage interface for the billing application can be derived from the interface `organizacion.infraestructura.patronRecursoSimple.ItfUsoRecursoSimple`, which is defined as follows:

```
package organizacion.infraestructura.patronRecursoSimple;
import java.rmi.Remote;
public interface ItfUsoRecursoSimple extends Remote { }
```

This means that the integration is quite simple as only there is a need to identify operations of usage classes in the JBilling application and add them to the usage

interface of the wrapper (see Figure 43). In this case, JBilling is structured as three sets of components: database, server and client. Those client components offer interfaces for managing billing application objects, and the resource wrapper has to manipulate and access those components.

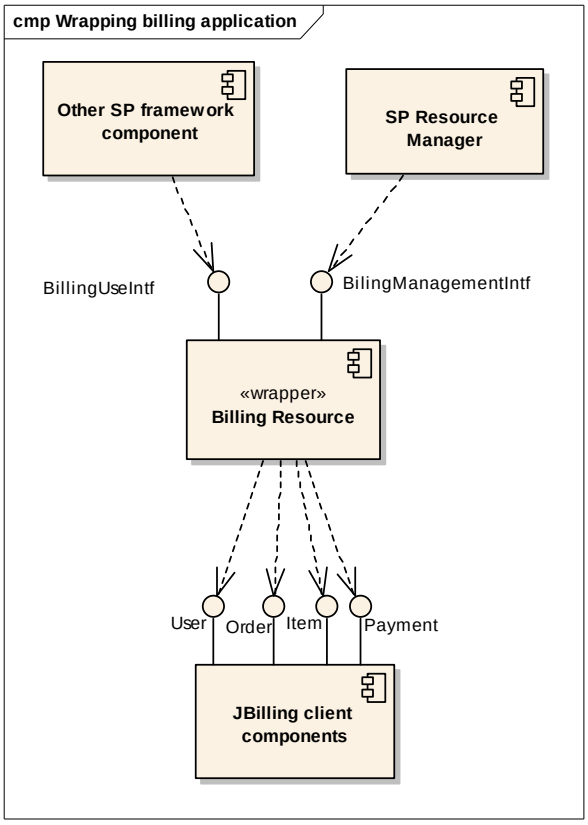

Figure 43 A resource as a wrapper of the JBilling component

In summary, this scenario shows how information from JBilling source code is extracted to identify relevant interfaces that have to be called by the wrapper. In this case, the best pattern to implement this wrapper is an object adapter, which gives more flexibility than a class adapter. Interfaces for the wrapper are derived from the specifications of the resource model of SP framework.

5.1.2.3 Modernisation by changing one specific technology associated to a component

Some of the components of SP framework are implemented with a proprietary tool, ILog JRules (http://www.ilog.com/products/jrules/). The purpose of this scenario is to consider the migration of components that are implemented with a concrete technology (ILog JRules in this case) to an implementation with other technology (Drools, or JBoss Rules, see http://labs.jboss.com/portal/jbossrules). This scenario shows a case of modernisation of components of the SP framework.

More concretely, the scenario considers modernisation of cognitive agents in the SP framework. This modernisation may affect three types of components:

- Modernisation of the Inference Engine. In this scenario we illustrate the steps needed to update the technology associated to a specific component. The goal is to change the rule engine, which is responsible of the inference mechanism in the cognitive control, by substituting the current component by an open source rule engine (Drools). External and internal interfaces of the Cognitive agent pattern should be maintained.
- Modernisation of rules that define the behaviour of the cognitive agent. As a consequence of the change of the Inference Engine component and the associated technology, there is a need to change the rules. This is mostly a syntactic transformation, although semantics consistency should be assured.
- Modernisation of (Java) objects that can be affected by changes on the inference engine or the rules (in principle this is minimized if interfaces are respected).

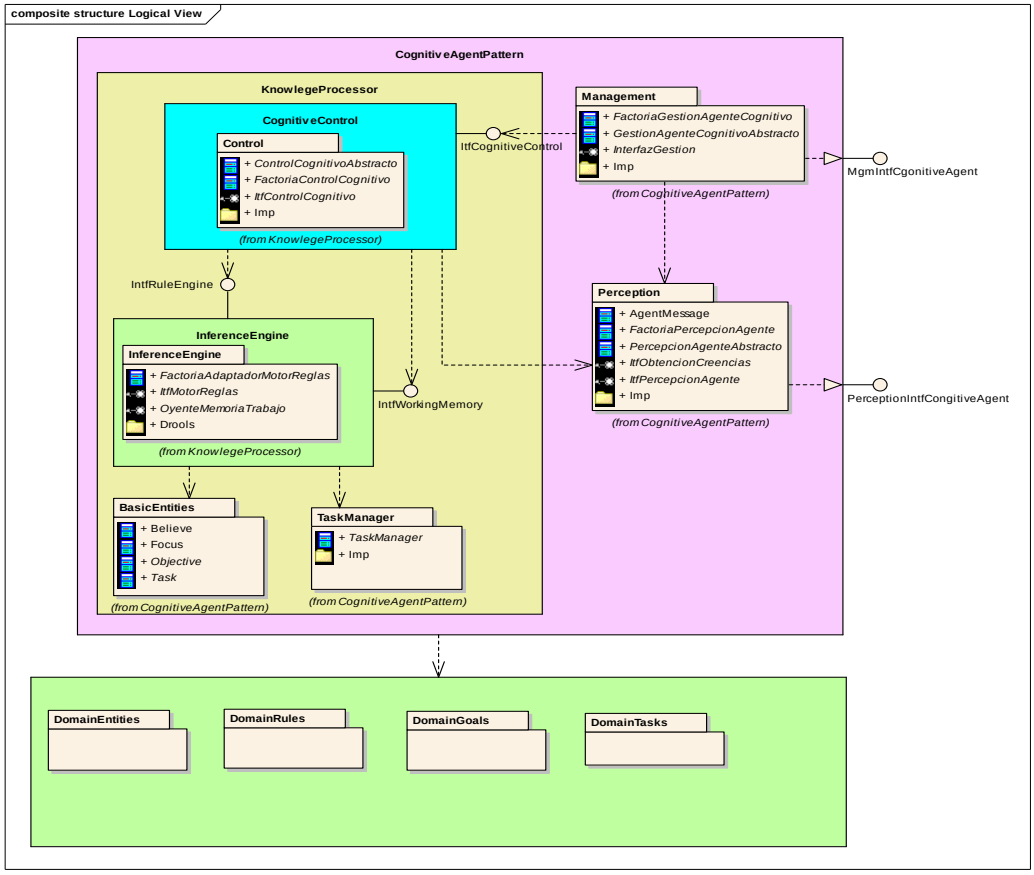

Figure 44 Cognitive agent architecture

This problem is basically of definition of transformations from rules in ILog JRules to rules in Drools, and analyzing how objects that are referenced in the rules can be invoked in rules for the target system, with Drools. When considering the change of platform, we can discover repetitive patterns on how to rewrite code from the initial system to the target. In this sense, it should be possible to identify transformation rules for making the modernisation process with some automated support, using XIRUP methodology and Momocs tools.

Additionally, in the process of doing this migration from one platform to another, some improvement in the code can be performed (this is indicated with use case *Redesign code*, which <<extends>> *Platform migration* use case in the diagram in Figure 39).

This scenario is currently being performed by hand (i.e., without MOMOCS methods and tools) and measures are being taken from this process. These measures will be useful to compare with those resulting from doing the same work with MOMOCS.

Interesting to note is that considerable parts of this scenario are repetitive tasks that should be automated with MOMOCS tools. An example of a rule coded with JRules and the resulting code with Drools is the following:

JRules code:

```
rule SearchComponentsGoal _init {
        when {
                ?obj:   SearchComponentsGoal (state == PENDING);
                ?focus: Focus (foco == ?obj);
        } then {
                ?obj.setSolving();
        }
}

rule SearchComponentsGoal _search {
        when {
                ?obj:   SearchComponentsGoal (state == SOLVING);
                ?focus: Focus (foco == ?obj);
                ?creencia: Belief(          emisor    instanceof String;
                                  emisor.equals("visualizador");
                                   contenido instanceof BuscarCmp;
                                   ((SearchComp)contenido).getTipoBusqueda() == SearchComp.CRITERIO_BUSQUEDA;
                                 );
        } then {
                bind ?task = new TaskBuscarRepositorioCriterioBusquedaBib(?context, ?obj);
                bind ?busqueda = (SearchComp) ?creencia.contenido;
```

```
                ?task.ejecutar(?busqueda.getTipoEntidad(), ?busqueda.getGenericidad(), ?busqueda.isEsSubtipo());
                retract(?creencia);
        }
}
```

Drools code:

```
rule “SearchComponentsGoal init”
        when
                obj:   SearchComponentsGoal (state == PENDING);
                focus: Focus (foco == obj);
         then
                obj.setSolving();
end

rule “SearchComponentsGoal search”
        when
                obj:   SearchComponentsGoal (state == SOLVING);
                focus: Focus (foco == obj);
                creencia:Belief(emisor==”visualizador",contenido:contenido
                 -> ((SearchComp)contenido).getTipoBusqueda() == SearchComp.CRITERIO_BUSQUEDA);
        then
                bind task = new TaskBuscarRepositorioCriterioBusquedaBib(context, obj);
                bind busqueda = (SearchComp) creencia.contenido;
                task.ejecutar(busqueda.getTipoEntidad(),busqueda.getGenericidad(), busqueda.isEsSubtipo());
                retract(creencia);
        end
```

Summary of differences:

Rule syntax:

JRules : rule <RuleName> when {<LHS> } then {<RHS>}

Drools: rule <RuleName> when <LHS> then <RHS> end

Variables:

JRules: ?<VarName>

Drooles <VarName>

Columns:

JRules: fact(<JavaConstraints>{;<JavaConstrants>}*)

Drools: fact(<attributeConstraints>{;<attributeConstraints> }*)

5.1.2.4 Modernisation by transformation and integration of software components

In this scenario (as example of the third use case) the purpose is to take make an integrated access control for all services in the modernized system. As initially the SME has several independent applications, the different information and processes at each one have to be used to configure and feed with data the access controller of the modernized system.

In the SP framework, the Access Controller component manages the user access to the system. When the system starts running, the Access Controller is invoked to carry out the user authentication.

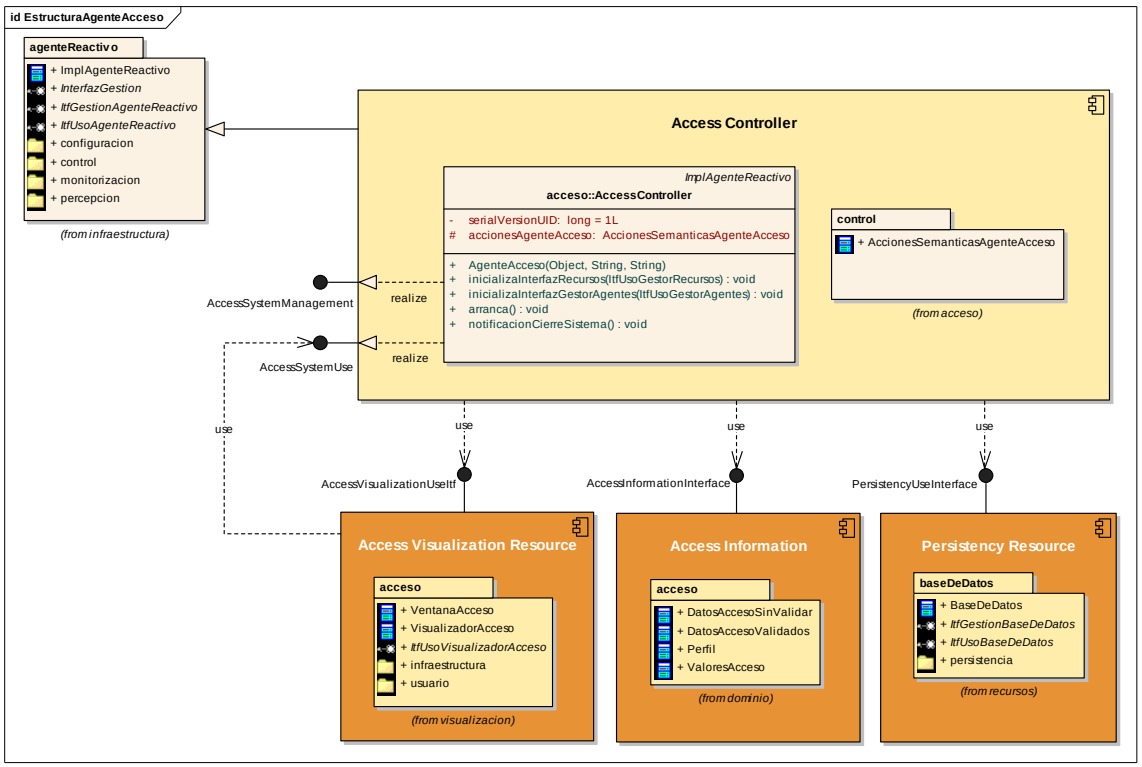

Figure 45 Access Controller

The Access Controller is responsible for *User authentication:* this control component uses the Access Visualization Resource to show users a login window, where they can write their username, password and profile. This agent also requires the use of the Persistency Resource to verify this data and authenticate users.

When a user is authenticated, the Access Controller is ordered to stand by and wait for another service request from the Controller Manager.

This component provides two interfaces:

Use interface: controllers are only able to access this interface by requesting it to the Controller Manager. It is used, for example, to process the authentication data of the

system users. This interface is also used by the Access Visualization Resource to create a communication link between the Access Controller and the users of the system.

Management interface: this interface is used by the Controller Manager to start, stop or monitor this agent.

The Access Controller uses the following interfaces:

Access Visualization Resource's use interface: this interface is used by the Access Controller to interact with the user that is attempting to access the system: shows a login window, shows error messages, and get user data.

Persistency Resource's use interface: this interface is used to get and store user data in order to validate the data entered by the user in the login window.

Behavior model

Start-up: the Access Controller, as well as the rest of the system controllers, is created immediately after the Controller Manager is created. After these controllers are created, they are sent a "start" message. This is shown in the sequence diagram in Figure 45.

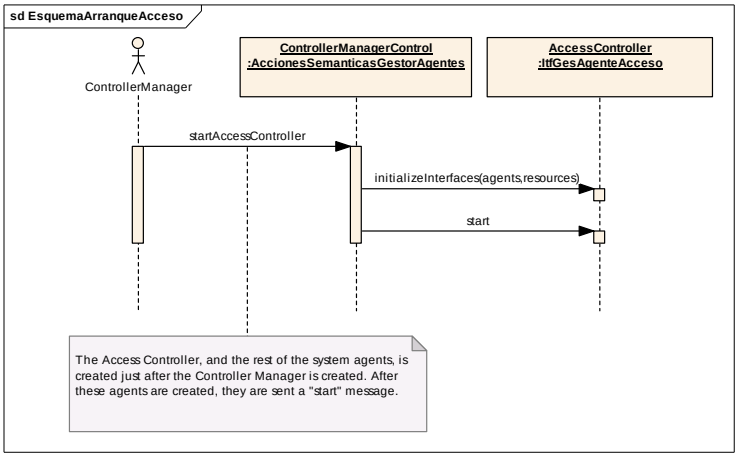

Figure 46 Access Controller's start-up

Start: after creating and starting all the controllers under its supervision, the Controller Manager yields system control to the Access Controller, which requests the Access Visualization Resource to show a login window where the user must write his/her authentication data (Figure 47).

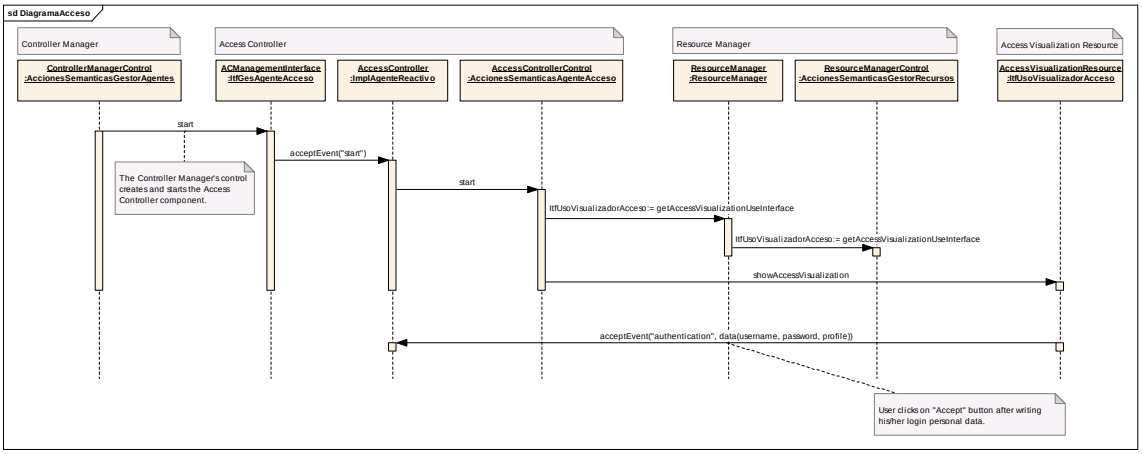

Figure 47 Access Controller's beginning

User authentication: when the Access Controller takes system control, it orders the Access Visualization Resource to show a login window (a form with fields for

username, password and profile). After clicking on the "Accept" button, the user authentication data is sent to the Access Controller, which will make a request to the Persistency Resource to check the user's data.

In this case, we are considering the result of this check true, so the user will be able to access the application. If this happens, the login window is closed and the agent corresponding to the user's selected profile takes control of the system.

If the result of the data check is false, the Access Controller will request the Access Visualization Resource to show the user an error message, informing about the problems found.

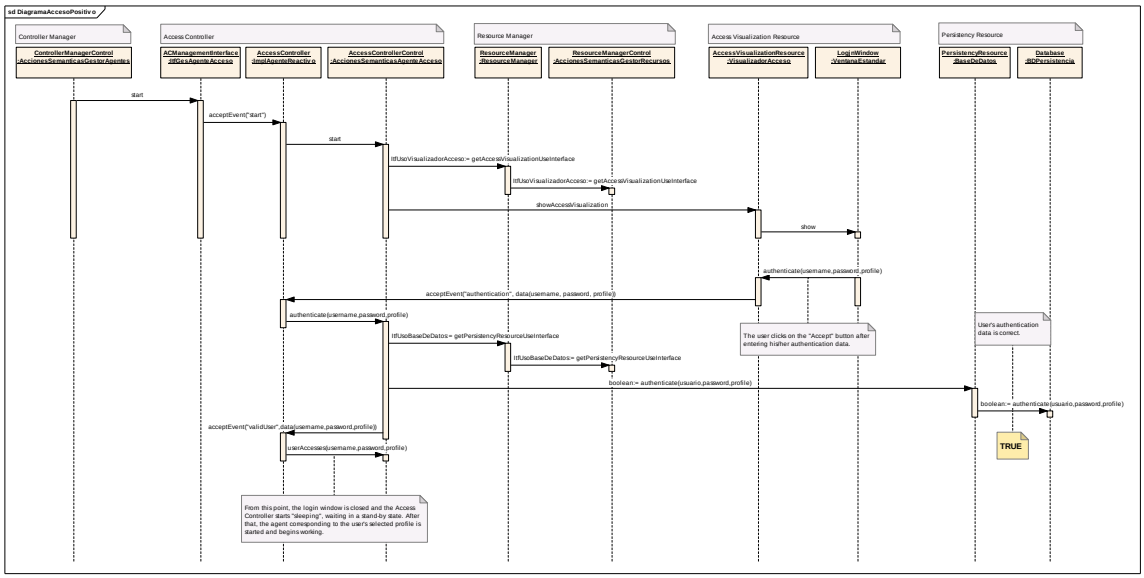

Figure 48 User authentication

Finite State Automaton: Specific control of the Access Controller is defined by a state machine (Figure 49). When this agent is created, it waits in the "initialState" for a "start" input. If this input is received, the automaton advances to the "waitForLogin" state, where it will wait for an authentication request.

If an error occurs during the authentication process, the automata will transition to a special state, where the error will be addressed.

There is also a "sleeping" state, where the agent waits when it is not working. If it receives a "wakeup" event, the agent will wait again for authentication requests.

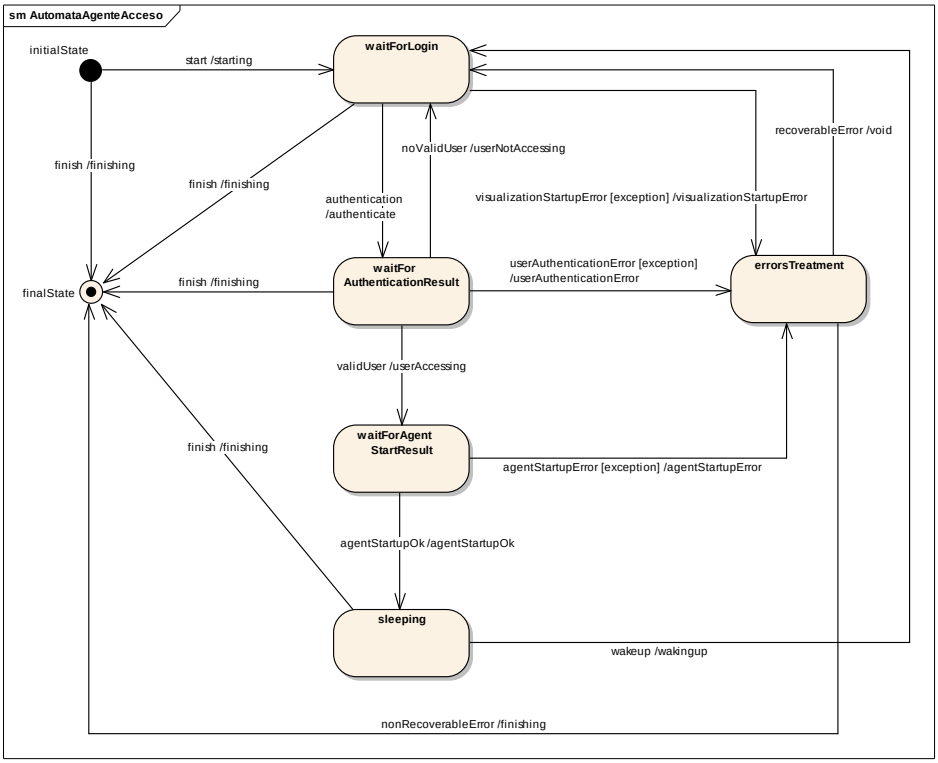

Figure 49 Access Controller state machine

The modernisation process in this case has to take care of helping to customize the access components of the SP framework. In order to do this, it needs to identify the different roles of the SME, the services each role will have to use, and with this information customize access components and data.

5.2 IND Case

To see the potential use of the above defined methodology and methodology fragments we will describe first the typical process and the related challenges of typical modernization project in the industrial solution business in this section and then map the methods fragment to the appropriate phases.

5.2.1 XIRUP Interpretation in IND Case

In the solutions business the process is generally divided in two main phases (Figure 50).

The first phase is the bid phase where the customer places a tender for selecting a supplier for the second phase. Normally there is even a phase before, in which the customer is deciding if a modernization or a rebuild of the plant should take place. This phase is not in our scope, but is a potential opportunity for future supplier to offer consultancy services and apply the above mentioned methods/ methods fragments.

The second phase is the execution phase, the modernization project is delivered by the supplier.

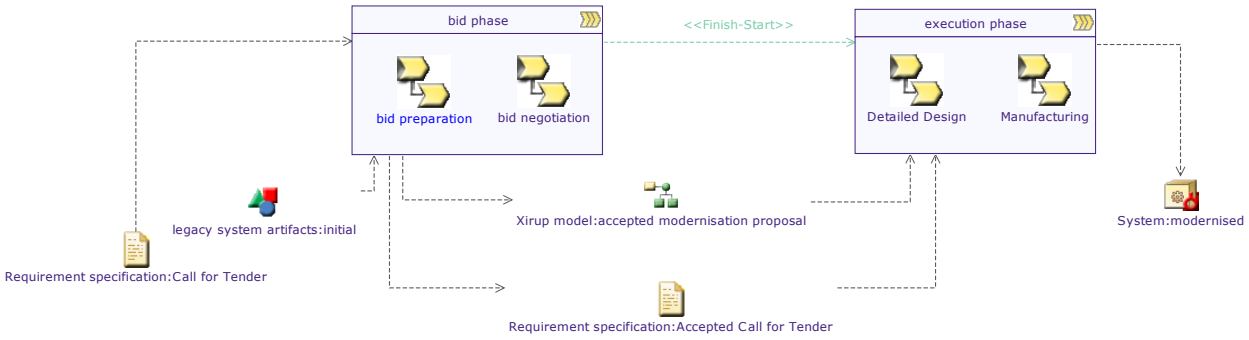

Figure 50 Siemens's Modernisation Process

5.2.1.1 Bid phase

During the "bid" phase the requirements for modernisation are analysed, as well as all information related to the modernisation tender. The decision about continuation of the activities is taken.

Typical challenges of the bid phase are the understanding of the existing solution, and also the review of all available documentation and extract all function/ non-functional requirements. Concerning the requirements the most crucial point is that the supplier must fulfil all requirements with the offered solution as well not more than demanded. In order to create a competitive and even better/cheaper bid the solution should use as much as possible existing assets by reusing components. The bid must be realized within a short time, finding the right degree of detail taking into account risks/migration strategies, relevant costs and future evaluations of the solution. Finally it comes to the decision to make or not to make a bid.

Bid preparation

Possible actions in this phase of the projects are to get an overview over the existing architecture; the current realized process and gain all relevant information of the system. Since the lack of time and existing assets it is not necessary to create a complete model of the to be modernised system (TBMS). In the first version of the TBMS model the focus should be to refine the effected i.e. to be modernized parts of the system, without going into the final details. While creating the TBMS model critical parts or areas of the modernization within the plant, hence the model can be identified. In order to reduce the endeavour to create the model with its components from scratch a component type library can support the modelling through reuse of components. Having this library in the background it is easier to group components and configure their properties comparable with a "plant configurator". When single components are created or some of them are aggregated the traceability of the assigned requirements and functionality should be ensured to prove its fulfilment. Regarding the properties of the components it is also feasible to distinguish between their visibility since specialists of the system are only interested in values/ information concerning their discipline by equipping the components with views. By applying these views an electrical engineer for example will see only components containing electrical attributes affecting his work and on the other hand the project manager is able to see every component since he needs all information. Finally, in order to guarantee the requested functionality of the system, different rules are assigned to the TBMS model in order to support the XIRUP engineer during the modernization and to help him meeting all the customer requirements and all edited constraints.

Bid negotiation

In the second process step of the bid preparation the bid negotiation takes place. In this phase the customer collects the bid documents from all suppliers and updates the customer specification. Considering the created TBMS model of the bid preparation step and the new information the XIRUP engineer is enabled to go through possible "what-if"-scenarios. By using these scenarios the effort of exchanging components is shown. Also the already assigned or to be introduced constraints support the XIRUP engineer within these steps by showing the modernization consequences when he exchanges a component. Being aware of the necessary modernization measures like restructuring, introduction new component types and constraints etc. the XIRUP engineer is capable to estimate all costs and calculate the price of the modernization requested by the customer with a high accuracy. Based on the estimation in the end there will be a bid or a no bid for the modernization project.

5.2.1.2 Execution Phase

If a contract between customer and supplier is signed the project execution phase is started. During this phase a detailed design of the modernised system is elaborated and validated by the customer. After this point the manufacturing phase is started including steps like components procurement, production and assembling. The last steps of this phase are deployment, validation and customer acceptance. These phases are out of focus in MOMOCS but mentioned in order to be complete.

Challenges of the execution phase are first of all to ensure the consistency of the work results, tracing and verification of the customer requirements, quality

assurance/management, super visioning and monitoring of the project and managing the change request and version control.

Detailed Design

Starting point of this phase is the already created TBMS model of the bid phase and documents like functional concepts, selection/classification/specification of the equipment or architecture. Most of these items are already defined during the bid phase, but as stated before normally they are a lot of changes during the negotiation or they are not detailed enough. So the design of the plant has to be updated according to the results of the negotiation phase. So the TBMS model representing the current as-built solution is step by step enriched to the modernised system (MS) model including among others reuse of components and consideration/introduction of constraints. While doing these actions always the customer requirements have to be kept in mind and therefore continuously tracked. Typical use cases and simulating the behaviour of the system support the XIRUP engineer by this iterative procedure. In the end code based on the model should be generated.

Activity Mapping

The presented phases and process steps can be link directly to the methods fragments, according to the section 4.3, see Table 5.

In the bid phase as well as in the execution phase all methods fragment are utilized since the model has to be created/refined/transformed/measured/documented. The level of detail is varying in the different phases, also the amount of use of the fragments.

Phase	XIRUP phase	Method fragments
Bid preparation	Preliminary Evaluation Understanding Building	Recovering Architecture Architecture Modernisation Transformation Definition Model Evaluation Code Generation
Bid negotiation	Understanding Building	Architecture Recovery Architecture Modernisation Transformation Definition Model Evaluation Code Generation
Detailed design	Understanding Building Migration	Architecture Recovery Architecture Modernisation Transformation Definition Model Evaluation Code Generation

Table 5 Mapping Activities - "Model Modernisation" usage semantics

5.2.2 IND Case Study Details

To illustrate better the activities that take place in the above mentioned process we have compiled a simple case study from the industrial solution domain.

5.2.2.1 To-Be-Modernised System

This section is about the creation of the TBMS model and covers the XIRUP phases preliminary evaluation, understanding and building.

Architecture

The highest mother component of our model is the airport logistic in an airport. The part of the airport logistic we consider in this case study is the automated baggage handling system.

Getting one abstraction level deeper, the automated baggage handling system can be refined in following components:

- Unload
- Reclaim
- Transfer
- Error handling areas
- Early Baggage Store
- Load
- Sorting Loops
- Hold Baggage Screening
- Check-In
- Empty Tray Store

The first step consists in identifying components at a high level, as well as their connectors and connections. The automated baggage handling system is represented graphically in Figure 51.

The next step consists in identifying relevant attributes for our modernising scenario. How to define attributes is described in section

Data Structure

Then, the relevant components for our modernising scenario are identified, and only these ones are refined. These steps are iterated until the needed refinement level in the model has been reached.

In this case, it is the Empty Tray Store that we will focus on. The Empty Tray Store itself can be refined in two components (see Figure 52):

- ETS for Normal size trays (where ETS stands for Empty Tray Store)
- ETS for Over-sized trays.

In the following, we will focus on ETS for Normal size trays. In the initial system, the empty tray store consists in a belt conveyor on which empty trays (trays that are currently not needed) are stored sequentially according to the LIFO principle (see Figure 55).

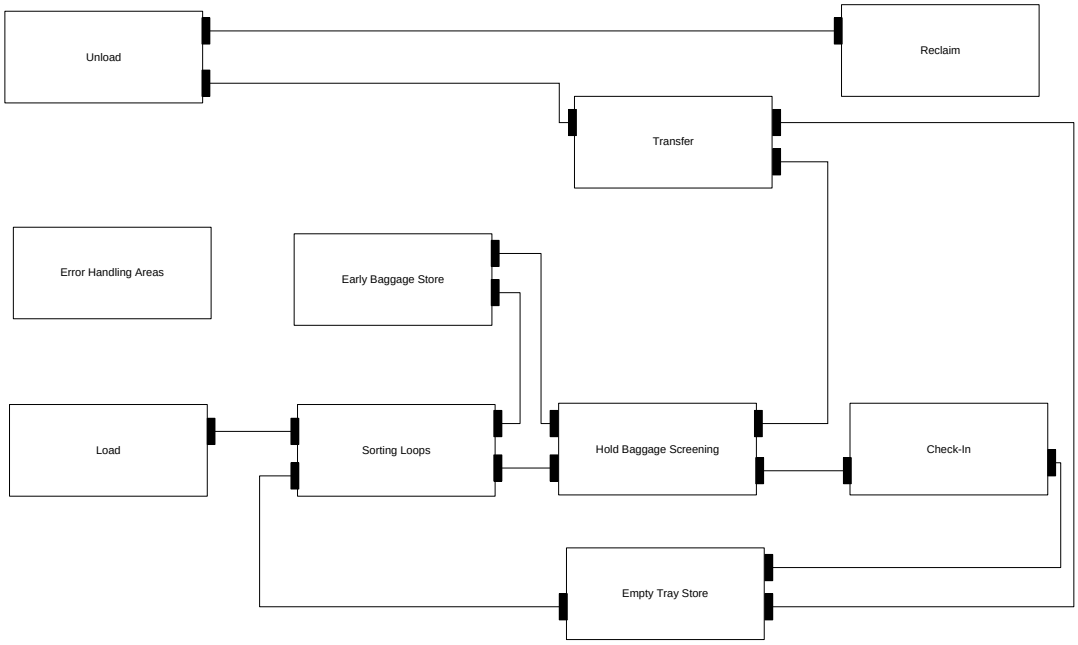

Figure 51 Automated Baggage Handling System

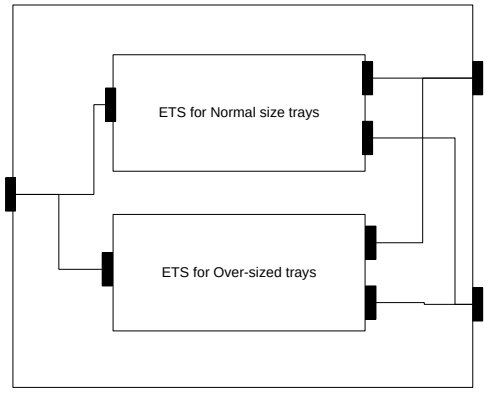

Figure 52 Empty Tray Store

In the following section, BC stands for belt conveyor, ETS for empty tray store, PC for photo cell, and PLC for programmable logic controller.

The ETS for Normal size trays can be again refined as seen in Figure 54.

At the end, the component that must be refined is ETS PLC, because it contains, amongst other, the software modules that are relevant for the empty tray store (see Figure 53):

- TSPLC-SWmod1
- TSPLC-SWmod2
- TSPLC-SWmod3

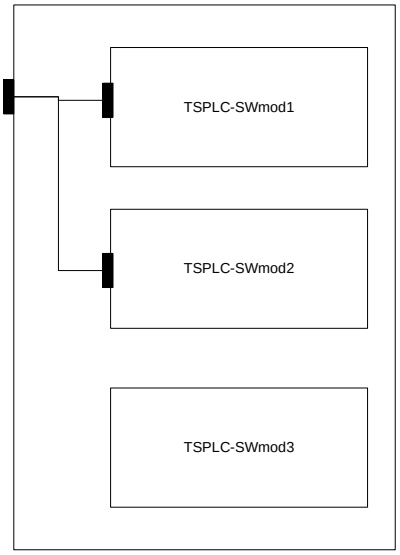

Figure 53 ETS PLC

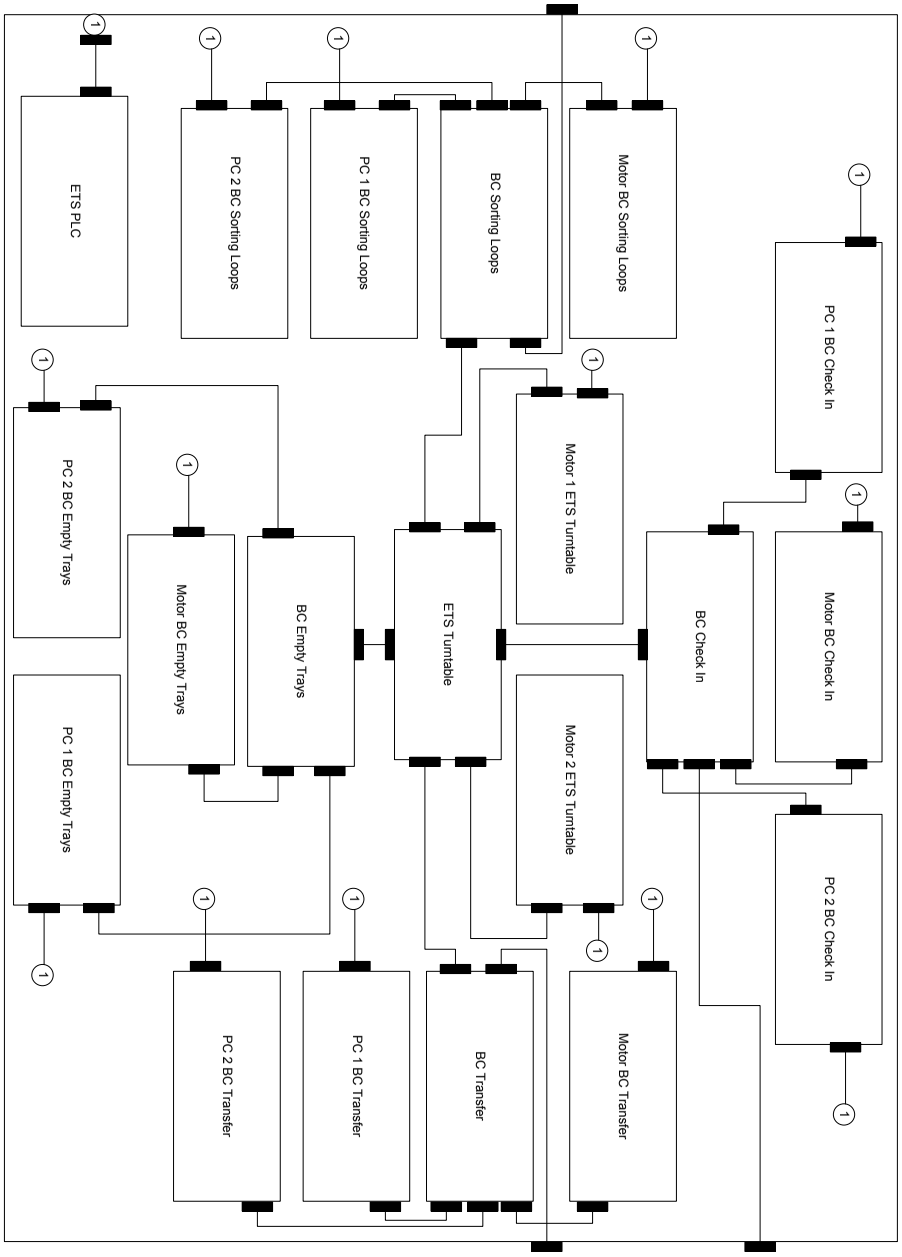

Figure 54 ETS for normal size trays

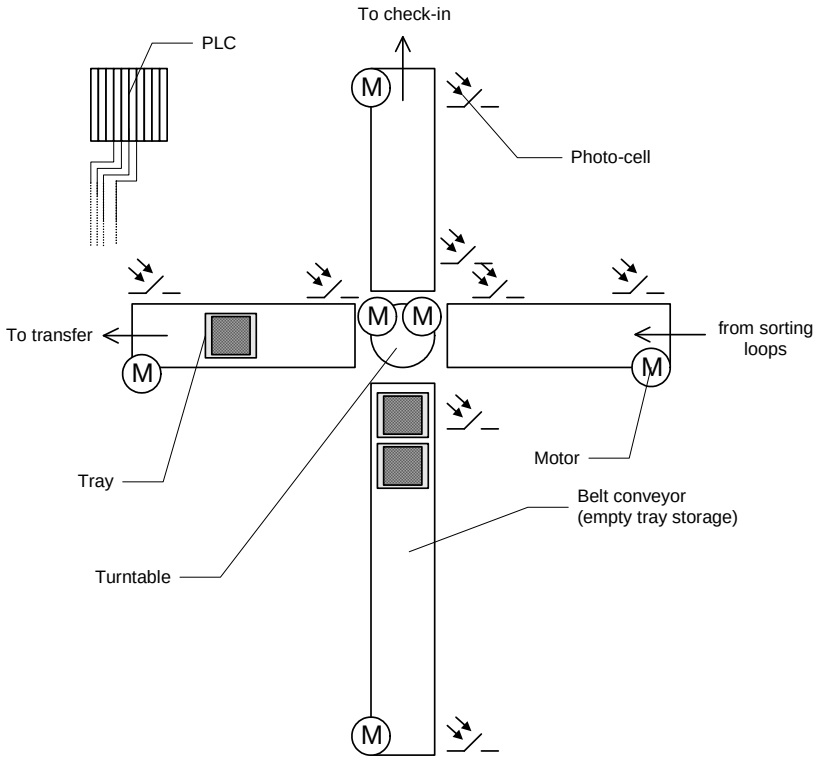

Figure 55 To-Be-Modernised empty tray store

Data Structure

The data structure of our model has to provide information about the mother component of a given component, so that a given abstraction layer (i.e. refinement layer) of the model can be identified. To this end, a clear identifier of components is needed to. A representation of this is shown in Table 6, where "Mother" is the mother component, "Component ID" is the identifier and "Type" is the type of the component (according to the type instance concept). In our data structure we have additionally a column for comments (e.g. for references on relevant project documentation). This allows sorting data according to these criteria, meaning these are the relevant keys in terms of database queries.

Mother	**Component ID**	**Type**
	Airport Logistic Center	Airport Logistics
Airport Logistic Center	Automated Baggage Handling System	Baggage Handling System
Automated Baggage Handling System (Automated BHS)	Check-In	AL Check-In
Automated BHS	Sorting Loops	AL Sorting System
Automated BHS	Transfer	AL Transfer
Automated BHS	Early Baggage Store	AL Baggage Store
Automated BHS	Empty Tray Store	AL Tray Store
Automated BHS	Hold Baggage Screening	Baggage Screening System

Automated BHS	Error Handling Areas	AL Error Handling Areas
Automated BHS	Unload	AL Unload
Automated BHS	Load	AL Load
Automated BHS	Reclaim	AL Reclaim
Empty Tray Store	ETS for Normal size trays	AL Tray Store System
Empty Tray Store	ETS for Over-sized trays	AL Tray Store System
ETS for Normal size trays	BC Check In	Belt Conveyor
ETS for Normal size trays	BC Sorting Loops	Belt Conveyor
ETS for Normal size trays	BC Transfer	Belt Conveyor
ETS for Normal size trays	BC Empty Trays	Belt Conveyor
ETS for Normal size trays	ETS Turntable	Turntable
ETS for Normal size trays	Motor BC Check In	Electrical Motor
ETS for Normal size trays	Motor BC Sorting Loops	Electrical Motor
ETS for Normal size trays	Motor BC Transfer	Electrical Motor
ETS for Normal size trays	Motor BC Empty Trays	Electrical Motor
ETS for Normal size trays	Motor 1 ETS Turntable	Electrical Motor
ETS for Normal size trays	Motor 2 ETS Turntable	Electrical Motor
ETS for Normal size trays	PC 1 BC Check In	Photo Cell
ETS for Normal size trays	PC 2 BC Check In	Photo Cell

ETS for Normal size trays	PC 1 BC Sorting Loops	Photo Cell
ETS for Normal size trays	PC 2 BC Sorting Loops	Photo Cell
ETS for Normal size trays	PC 1 BC Transfer	Photo Cell
ETS for Normal size trays	PC 2 BC Transfer	Photo Cell
ETS for Normal size trays	PC 1 BC Empty Trays	Photo Cell
ETS for Normal size trays	PC 2 BC Empty Trays	Photo Cell
ETS for Normal size trays	ETS PLC	PLC
ETS PLC	TSPLC-SWmod1	Software Module
ETS PLC	TSPLC-SWmod2	Software Module
ETS PLC	TSPLC-SWmod3	Software Module

Table 6 Components

Afterwards, relevant attributes are identified, and in the same way, are structured in a table. The columns of this table are: "Attribute Id" (identifier), "Owner" (the identifier of the component to which the attribute belongs), "Type" (Type of the contents of the attribute), "Content" (the concrete contents related to the attribute), "Comments" (e.g. for references on relevant project documentation), and then comes one column for every concerned engineering domain (for example, an "X" in the corresponding cell tells this attribute is relevant for that engineering domain). The columns about the engineering domain allow sorting the attributes according to the desired domains, thus allowing generating specific views of the system (not represented here). Some relevant attributes are represented in Table 7.

Attribute ID	Owner	Type	Content	Comments
Unload_Con1_att1	Unload	Integer		Baggage flow in Trays per min
Unload_Con2_att1	Unload	Integer		Baggage flow in Trays per min
Reclaim_Con1_att1	Reclaim	Integer		Baggage flow in Trays per min
Transfer_Con1_att1	Transfer	Integer		Baggage flow in Trays per min
Transfer_Con2_att1	Transfer	Integer		Baggage flow in Trays per min
Transfer_Con3_att1	Transfer	Integer		Baggage flow in Trays per min
EBS_Con1_att1	Early Baggage Store	Integer		Baggage flow in Trays per min
EBS_Con2_att1	Early Baggage Store	Integer		Baggage flow in Trays per min
Load_Con1_att1	Load	Integer		Baggage flow in Trays per min
Loops_Con1_a	Sorting	Integer		Baggage flow in Trays per

tt1	Loops			min
Loops_Con2_att1	Sorting Loops	Integer		Baggage flow in Trays per min
Loops_Con3_att1	Sorting Loops	Integer		Baggage flow in Trays per min
Loops_Con4_att1	Sorting Loops	Integer		Baggage flow in Trays per min
HBS_Con1_att1	Hold Baggage Screening	Integer		Baggage flow in Trays per min
HBS_Con2_att1	Hold Baggage Screening	Integer		Baggage flow in Trays per min
HBS_Con3_att1	Hold Baggage Screening	Integer		Baggage flow in Trays per min
HBS_Con4_att1	Hold Baggage Screening	Integer		Baggage flow in Trays per min
ETS_Con1_att1	Empty Tray Store	Integer		Baggage flow in Trays per min
ETS_Con2_att1	Empty Tray	Integer		Baggage flow in Trays per

	Store			min
ETS_Con3_att1	Empty Tray Store	Integer		Baggage flow in Trays per min
Check-In_Con1_att1	Check-In	Integer		Baggage flow in Trays per min
Check-In_Con2_att1	Check-In	Integer		Baggage flow in Trays per min
Current speed	BC Empty Trays	Decimal		Linear speed Speed in m/s with 2 positions after decimal point; Can be negative, depending on direction
Max speed	BC Empty Trays	Decimal	10	Linear speed in m/s; in both directions with 2 positions after decimal point
Max Load	BC Empty Trays	Integer	700	Mass in kg
Current tray exchange	BC Empty Trays	Integer		current exchanged tray; can be +1, 0, or -1 depending on direction
Max tray capacity	BC Empty Trays	Integer	30	

Trays stored	BC Empty Trays	Integer		currently stored trays
Trays left	BC Empty Trays	Integer		number of trays that could still be hosted
BC-Lenghth	BC Empty Trays	Integer		in m
BC-Width	BC Empty Trays	Integer		in m
MotBCET-SteerSig	Motor BC Empty Trays	Decimal		Electrical steering signal in V with 2 positions after decimal point
MotBCET-Speed	Motor BC Empty Trays	Decimal		Circular speed in RPM
PC1BCET-Sig	PC 1 BC Empty Trays	Boolean		Logic Signal (TTL boolean); True when tray is passing
PC2BCET-Sig	PC 2 BC Empty Trays	Boolean		Logic Signal (TTL boolean); True when tray is passing

ETSPLC-Itf	ETS PLC	Tuple of 12 Elements		This is the interface-plug of the PLC, where all the control signals of the components are connected. The content is a tuple of diverse digital or analogic electrical signals. The first element is the steering signal for MotBCET-SteerSig. (ETSPLC-Itf-1, …, ETSPLC-Itf-12)
ETSNS_Con1_att1	ETS for Normal size trays	Integer		Baggage flow in Trays per min
ETSNS_Con2_att1	ETS for Normal size trays	Integer		Baggage flow in Trays per min
ETSNS_Con3_att1	ETS for Normal size trays	Integer		Baggage flow in Trays per min
BC_SL_Con1_att1	BC Sorting Loops	Integer		Baggage flow in Trays per min
BC_T_Con1_att1	BC Transfer	Integer		Baggage flow in Trays per min

BC_CI_Con1_att1	BC Check In	Integer		Baggage flow in Trays per min

Table 7 Attributes

Another table defines the connectors with following columns: "Connector ID" (identifier of the connector), "Owner" (the component to which the connector belongs), "Attributes" (the attributes that are related to that connector), and "Comments" (not represented here). There may be several attributes related to a connector, though there is only one in the following example. Some relevant connectors are represented in Table 8.

Connector ID	**Owner**	**Attributes**
Unload_Con1	Unload	Unload_Con1_att1
Unload_Con2	Unload	Unload_Con2_att1
Reclaim_Con1	Reclaim	Reclaim_Con1_att1
Transfer_Con1	Transfer	Transfer_Con1_att1
Transfer_Con2	Transfer	Transfer_Con2_att1
Transfer_Con3	Transfer	Transfer_Con3_att1
EBS_Con1	Early Baggage Store	EBS_Con1_att1
EBS_Con2	Early Baggage Store	EBS_Con2_att1
Load_Con1	Load	Load_Con1_att1
Loops_Con1	Sorting Loops	Loops_Con1_att1

Loops_Con2	Sorting Loops	Loops_Con2_att1
Loops_Con3	Sorting Loops	Loops_Con3_att1
Loops_Con4	Sorting Loops	Loops_Con4_att1
HBS_Con1	Hold Baggage Screening	HBS_Con1_att1
HBS_Con2	Hold Baggage Screening	HBS_Con2_att1
HBS_Con3	Hold Baggage Screening	HBS_Con3_att1
HBS_Con4	Hold Baggage Screening	HBS_Con4_att1
ETS_Con1	Empty Tray Store	ETS_Con1_att1
ETS_Con2	Empty Tray Store	ETS_Con2_att1
ETS_Con3	Empty Tray Store	ETS_Con3_att1
Check-In_Con1	Check-In	Check-In_Con1_att1
Check-In_Con2	Check-In	Check-In_Con2_att1
ETSNS_Con1	ETS for Normal size trays	ETSNS_Con1_att1
ETSNS_Con2	ETS for Normal size trays	ETSNS_Con2_att1
ETSNS_Con3	ETS for Normal size trays	ETSNS_Con3_att1
BC_SL_Con1	BC Sorting Loops	BC_SL_Con1_att1
BC_T_Con1	BC Transfer	BC_T_Con1_att1
BC_CI_Con1	BC Check In	BC_CI_Con1_att1

Table 8 Connectors

The last table describes the connections of the model. As a connection is a couple of connectors, this table has only two columns for the two identifiers of the connectors. Some connections are represented in Table 9.

Connector 1	**Connector 2**
Unload_Con1	Reclaim_Con1
Unload_Con2	Transfer_Con1
Transfer_Con2	ETS_Con3
Transfer_Con3	HBS_Con3
EBS_Con1	HBS_Con1
EBS_Con2	Loops_Con3
Load_Con1	Loops_Con1
Loops_Con2	ETS_Con1
Loops_Con4	HBS_Con2
HBS_Con4	Check-In_Con1
Check-In_Con2	ETS_Con2

Table 9 Connections

Behavior

In this scenario, the luggage pieces are conveyed within trays, in order to allow a higher transportation speed on some sections. Trays that are temporarily not needed are stored in the Empty Tray Store.

Baggage Workflow

Via the check-in the passenger's baggage is introduced into the BHS. Another possibility of getting baggage in the transport cycle is through unload, provided that the baggage is transfer-baggage. Afterwards the baggage is always searched for potential dangerous goods in the HBS (Hold Baggage Screening). In case that the flight is ready to get loaded, the affected pieces of baggage will be routed to load. Otherwise the baggage is stored in an Early Baggage Store, abbreviated EBS. If there is an overflow at the HBS, the baggage is also stored inside the EBS and will be scanned afterwards. Sorting is managed by using time slices. A pure flight assortment has turned out to be inefficiently. The application of the EBS type depends on the utilization. A solution without trays contains a certain complexity, because the conveyor access is managed over photo eyes. For this reason the positions of baggage has to be saved meticulously. Moreover, the remaining store quantum can only be estimated, because you can not consider typical dimensions of the transported baggage. Anyway, a trigger will reactivate the baggage and arrange the leading to the proper loading-gate.

If the destination airport is reached, the baggage is transported to the reclaim area. Transfer-baggage is transported to the corresponding gates via the transfer lanes. (see Figure 56).

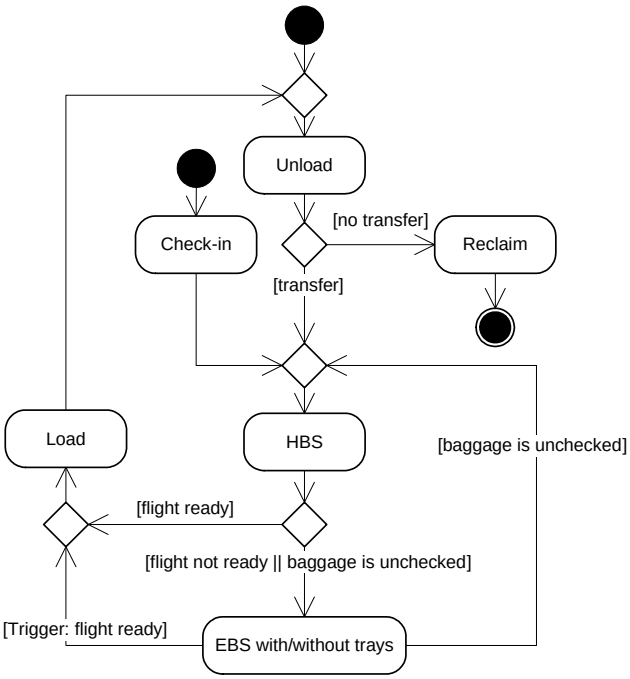

Figure 56 Baggage workflow

Tray workflow

The constant circulation of trays starts and ends at the empty tray store (ETS). Requested by check-in or transfer, the tray is routed to the corresponding station, picking up the piece of baggage. Thereby the 1:1 relationship of baggage-to-tray is set up. After the security-check by the HBS the availability of the flight is checked. If the flight is ready, the tray is transported to load and the piece of baggage tilted. Otherwise an EBS is used. In case the EBS is handled without trays the baggage will be removed from the tray and the empty tray returns to the ETS. In the tray based EBS this return is done after reactivating and delivering the baggage at the load-gate. (see Figure 57).

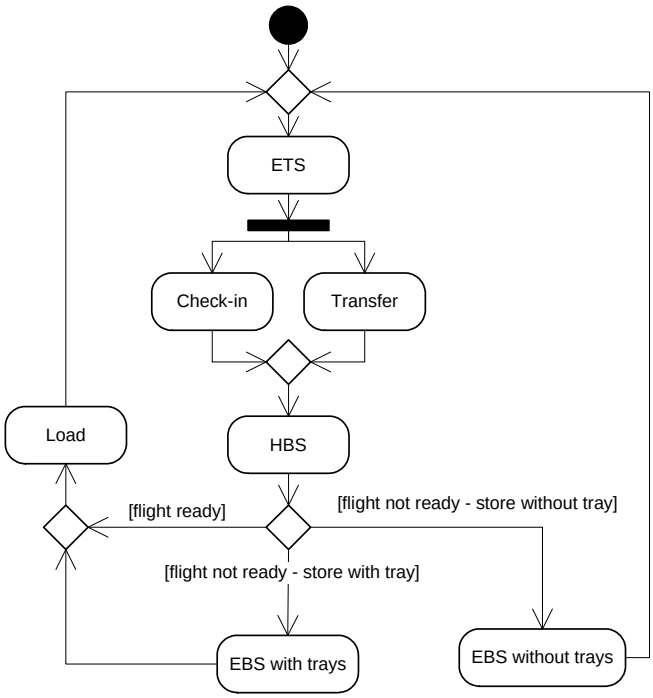

Figure 57 Tray workflow

Dependencies

The next step is to identify and define the rules governing the system. The rules should be edited separately, and as far as possible involve concrete contents of attributes, in order to be able to verify them automatically. Here are some examples.

The first obvious thing is to verify that the connectors of "child" components are compatible with the corresponding connectors of their mother components. For example:

ETSNS_Con1.ETSNS_Con1_att1 = ETS_Con1.ETS_Con1_att1

ETSNS_Con2.ETSNS_Con2_att1 = ETS_Con2.ETS_Con2_att1

ETSNS_Con3.ETSNS_Con3_att1 = ETS_Con3.ETS_Con3_att1

BC_SL_Con1.BC_SL_Con1_att1 = ETSNS_Con1.ETSNS_Con1_att1

BC_CI_Con1.BC_CI_Con1_att1 = ETSNS_Con2.ETSNS_Con2_att1

BC_T_Con1.BC_T_Con1_att1 = ETSNS_Con3.ETSNS_Con3_att1

In the same way, it must be verified if connectors of a component are compatible with the connectors of the other components to which they are connected.

One rule is that the belt conveyor of the empty tray store has to fit into the room, so there are constraints on its dimensions.

BC-Length < Length of the room

BC-Width < width of room + Room for operators to go around

The current speed of a belt conveyor is always inferior to its maximum speed.

Current speed <= Max speed

So there has e.g. to be verified, if there is a minimum speed that has to be reached in a technological process, and if this minimum speed is inferior to the maximum

speed. For example, the belt conveyor has to reach a minimum speed in order to host the arriving empty trays on time, if a certain flow is imposed (regulative rule).

This rule on its turn has consequences for the choice of the motor of the belt conveyor, since the speed of the motor is directly related to the speed of the conveyor (informative rule).

MotBCET_Speed = k1 * MotBCET_SteerSig

Where k1 is a decimal constant.

The speed of the belt conveyor is dependent on the speed of the motor.

BC Empty Trays.Current speed = k2 * MotBCET-Speed

Where k2 is a decimal constant.

The steering interface of the PLC determines the steering signal of the motor:

MotBCET-SteerSig = ETSPLC-Itf-1

(ETSPLC-Itf-1 determines MotBCET-SteerSig, not the opposite)

At the end, in the Software module TSPLC-SWmod1, there is a steering variable SteerVAR1 that determines the value of the steering signal of the PLC.

ETSPLC-Itf-1 = TSPLC-SWmod1.SteerVAR1

5.2.2.2 The Modernised System

This section is about the creation of the MS model and covers the XIRUP phases, understanding building and migration.

The modernisation idea is to install a stacker in the empty tray store in order to put trays in a pile. When trays are piled onto each other, a considerable amount of space is saved, thus augmenting the capacity of the empty tray store. Only when the maximum height of a pile is reached, is the pile conveyed one tray length further on the belt conveyor. Additionally, the belt conveyor is split in two belt conveyors: one conveyor hosting the stacker and one buffering conveyor before it (in order to let time for the stacker to pile the trays). Since we want to modernise the ETS for Normal size trays, this component is refined as well.

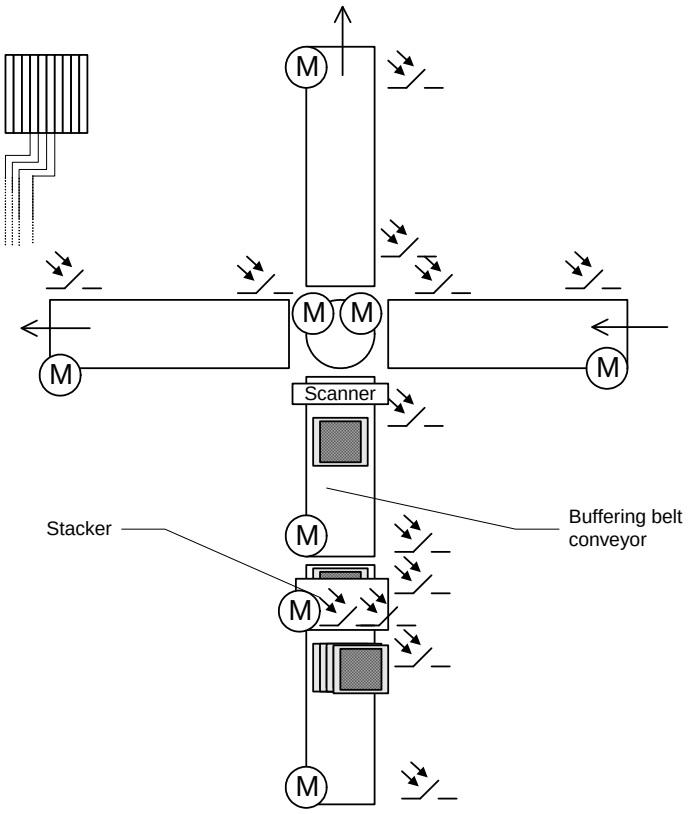

Figure 58 Modernised System

Architecture

In the modernised ETS for Normal size trays the components belonging to sorting, transfer, check in and the turntable are the same as in the current system. In order to be able to stockpile trays the Empty Tray components have to be exchanged.

Therefore the following components are included (see Figure 59):

- BC Buffer Empty Tray
- BC Stacker Empty Tray

- ETS Stacker
- Motor BC Buffer Empty Tray
- Motor BC Stacker Empty Tray
- Motor ETS Stacker
- PC 1 BC Buffer Empty Tray
- PC 2 BC Buffer Empty Tray
- PC 1 BC Stacker Empty Tray
- PC 2 BC Stacker Empty Tray
- PC 3 BC Stacker Empty Tray
- PC 1 ETS Stacker
- PC 2 ETS Stacker
- Scanner

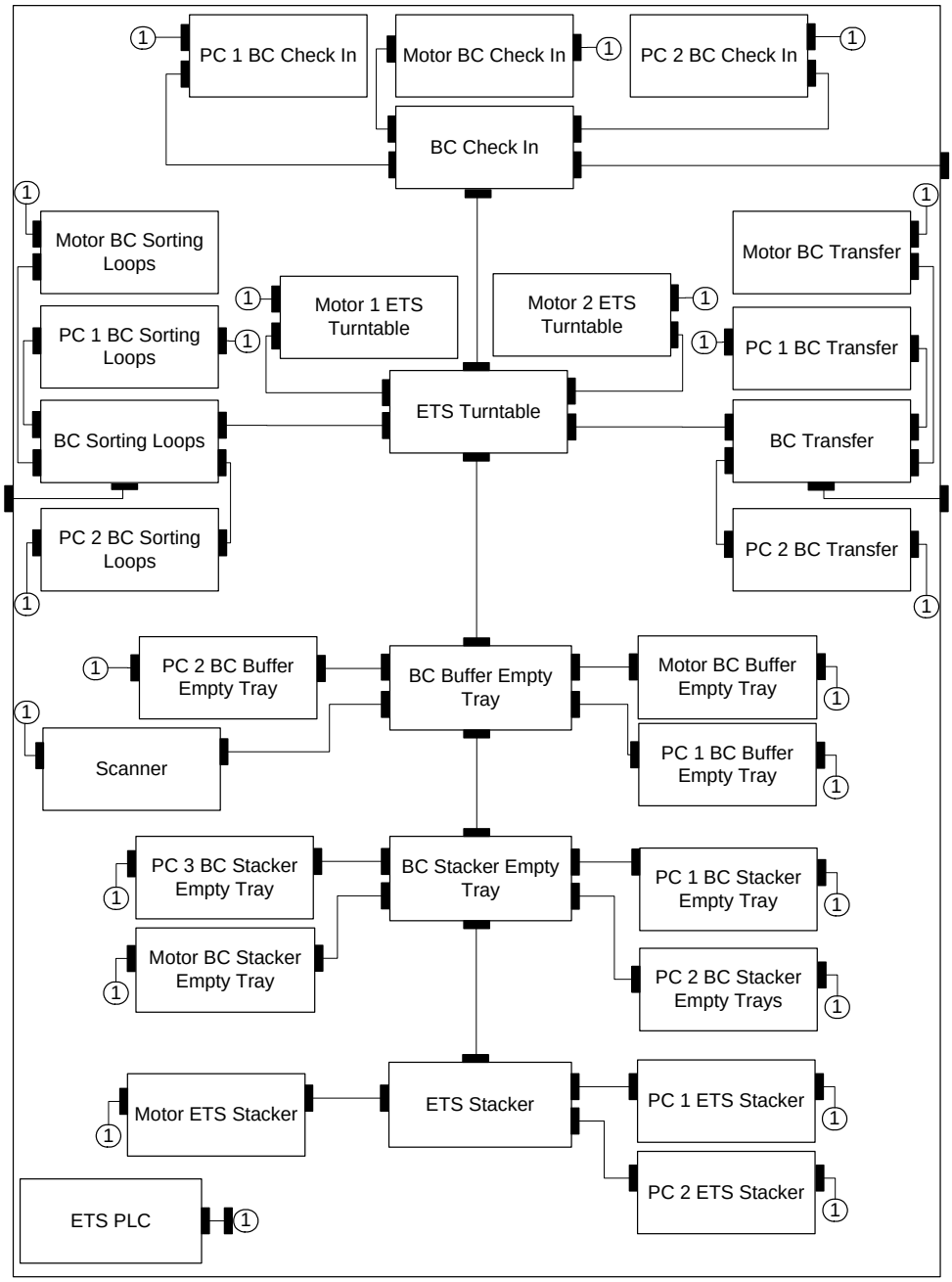

Figure 59 Modernised ETS for normal size trays

Behaviour

First all trays pass the scanner storing their IDs in order to know which trays are temporarily not used. Afterwards the arriving trays are buffered and step-by-step transported to the processing position by the stacker's belt conveyor, underneath the stacker. The stacker will lift the first tray, hold the tray in this upper position and remain in this position until the next tray arrives (see Figure 60).

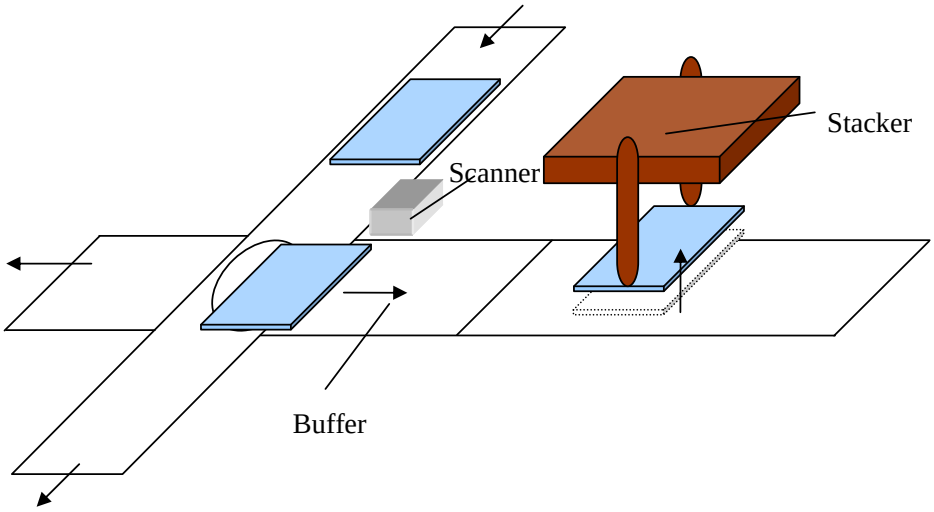

Figure 60 Modernised ETS for normal size trays - Stacking

After putting down the first arrived tray onto the second tray, the stacker moves its grippers to the second tray and lifts both trays in the upper position (see Figure 61). The stacker will continue with the other trays in the same way until the maximum number of stackable trays is reached.

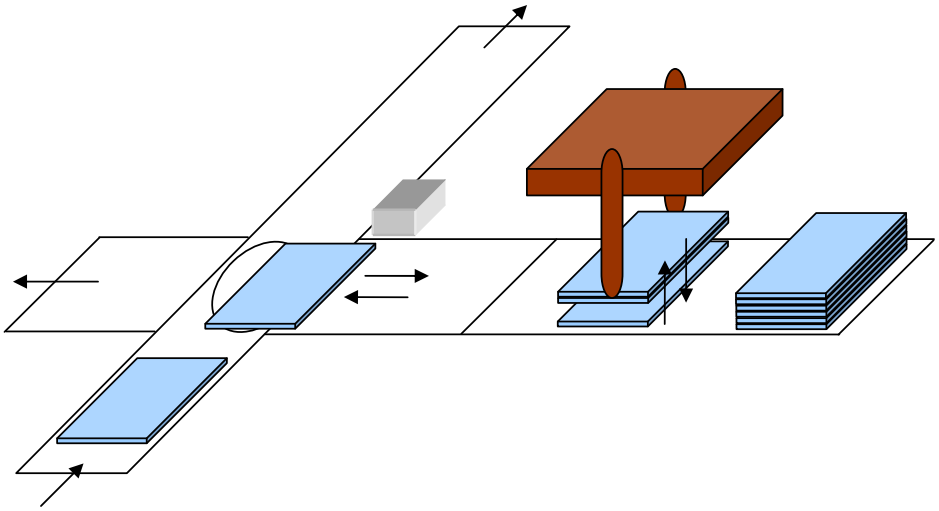

Figure 61 Modernised ETS for normal size trays – Stacking 2

If one pile is completed, the stacker will put down the stockpiled trays onto the conveyer belt and the conveyer belt will transport them to the waiting position. This sequence continues according to the LIFO principle until either the tray buffer is full or the trays will be needed again. Removing the trays from the tray pile will work in reverse way.

Dependencies

During the modernisation phase, one can try to replace components of the model of the initial system with new components in this structure, and then test if the model still verifies all the rules (e.g. new belt conveyor's size and speed). This way, several modernisation scenarios can be tested on the model before taking a final decision.

6 Compliance with Main Drivers

This section presents a brief analysis on how the main drivers are respected by the proposed methodology. In addition, it brings traceability with the main generic requirements for the methodology.

Drivers for Methodology	**Related Requirements from [D11, D12]**	**Coverage**
Generic methodology	GR6, GR7, MR3, MR7, IND.2	Generic XIRUP process is proposed (3.1), as well as several MDE method fragments to be used for methodology fulfilment (4.3). The methodology is interpreted for 2 different case studies presenting 2 different engineering disciplines.
Taking advantage of existing system	TELCO.9, TELCO.15	The XIRUP Process starts with the system evaluation in order to take advantage of the existing system. The model of existing system is built with the Architecture Recovery method fragment.
Agility	[DOW], TELCO.6, TELCO.7, TELCO.25	XIRUP proposes set of method fragments to be combined and used for customising the modernisation process. Different interpretations of the generic XIRUP

		process are possible depending on the engineering discipline, problem complexity and other parameters.
Incremental and Iterative	GR2, GR4, TELCO.7	The generic process is iterative, which is specified by backwards transitions in the workflow definition. The incremental nature is specified by the principles – the modernisation should be performed feature by feature and component by component.
Model Driven Modernisation	[DOW], GR3	A set of MDE method fragments for system modernisation is proposed, which cover the current state of the art in ADM.
User-centred	GR4, MR1, TELCO.1	XIRUP process is feature driven, while features are directly related to user requirements. Thus, the modernisation is driven by end-user demands.
Assuring of migration period	TELCO.8, TELCO.10	The migration is one of the phase of the generic XIRUP process.

Table 10 Main Drivers Compliance

7 XIRUP Methodology Walkthrough

7.1 Patterns and Tools

7.1.1 Generic Process and Fragments Reminder

This is an excerpt from the section 3.1.

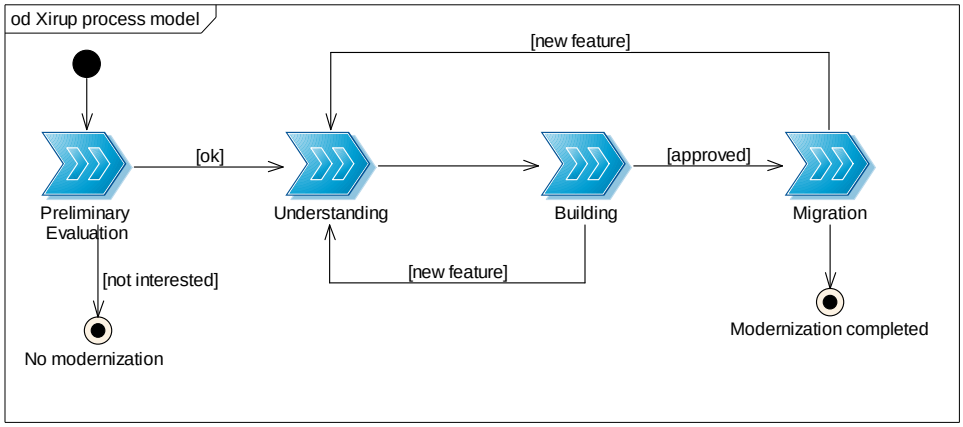

Figure 62 XIRUP process model

The following table illustrates the fragment usage in the XIRUP process.

XIRUP Phases	XIRUP Method Fragments Usage
Preliminary Evaluation	Architecture Recovery for building a snapshot of the system to be modernised. Model Evaluation for preliminary analysis of the current state of the system and the modernisation problem. It may also drive the

	decision to modernise the system. Architecture Modernisation and Code Generation for building prototypes of the modernised system.
Understanding	Architecture Recovery for building a model of the system for further analysis. Architecture Modernisation and Transformation Definition for creating a component model of the modernised system. Model Evaluation for in-depth analysis of the current and modernised systems. Code Generation for building prototypes.
Building	Code Generation for building required components. Model Evaluation for testing the generated components
Migration	Code Generation for building required components for specific platforms. Model Evaluation for testing components in the platform environment.

Table 11 Method Fragments in XIRUP

7.1.2 Corresponding Tools

The following table presents the patterns and the tools to be used for their running.

XIRUP Fragments	Corresponding Tools
Architecture Recovery	For the architecture recovery different tools are applicable: • For the Telco case study a specific tool has been developed. • Code reverse engineering into UML 2 + XSM Transformation Tool (part of the MOMOCS Tool Suite). UML2 reverse engineering tools can be used to create UML2 models from source files (C++, Java, C#) for example Objecteering CASE Tool by SOFTEAM (http://www.objecteering.com). Then the XSM TT tool can be used to transform the UML2 models into XSM models. • Custom transformation with MODISCO tool (http://www.eclipse.org/gmt/modisco/)
Architecture Modernisation	Different approaches can be followed: • Manual modelling with XSM Editor and Analysis (part of the MOMOCS Tool Suite) • Automatic transformation with XSM TT (part of the MOMOCS Tool Suite) by means of user-defined transformations or predefined transformation patterns. For more information about predefined transformation patterns please see [D53] • Component Grouping using XSM Editor and Analysis

	tool (part of the MOMOCS Tool Suite) • Other transformation engines: o oAW http://www.eclipse.org/gmt/oaw/ o Epsilon http://www.eclipse.org/gmt/epsilon/ o Kermeta http://www.kermeta.org/
Transformation Definition	Depending on the engine used the native transformation languages can be chosen. MOMOCS team proposes to use ATL upon which a dedicated solution (XSM TT) was developed. This solution aims at speeding up the definition of MOMOCS transformations.
Model Evaluation	
Quality Analysis	The metrication solutions should be used: • For the UML2 Objecteering Metrics can be used (available for the consortium) • Metrication using XSM Editor and Analysis Tools (part of the MOMOCS Tool Suite)
Verification	If model visualization is enough for verification, it should be done with: • XSM Editor and Analysis Tools • A model transformation in UML or XMI format allows using external modeller – like Enterprise Architect or

	Objecteering. Model verification should be also done by creating OCL constraint inside the model and by using the OCL Checker included to the XSM Editor and Analysis Tool.
Simulation	Currently only prototype tools are available for model based simulation. An appropriate candidate is ModelPlex SVT workbench, which will be available at the end of 2008.
Testing	Currently only commercial tools are available for model-based testing, e.g. TTworkbench [D53] can be used for definition and execution of the TTCN3 test cases. Eclipse TPTP (http://www.eclipse.org/tptp/) tries to provide an open source solution for a testing environment.
Code Generation	Template based tools and general use model transformation engines can be used for code generation, please see Eclipse JDT, MOFScript, ATL, oAW and Epsilon. In addition, Objecteering CASE Tool can be used for generation of C++ and Java code from UML models. In this case a typical process will be: (1) transformation of XSM models into UML with XSM TT; (2) code generation with Objecteering.

Table 12 Corresponding Tools

7.2 Typical Workflow

Based on the Travel Agency scenario, the following workflow and the tool chain are used. This all is illustrated with the screenshot demo[1]

Preliminary Evaluation:

XIRUP Fragment	Tools Used
Architecture Recovery: The model of the Travel Agency is manually created using the available legacy artefacts: site navigation map, server configuration files, source code	XSM Editor and Analysis tool is used to manually create the XSM model.
Model Analysis, Verification: In order to determine the architecture changes to be done to solve the modernization problem ("Ensure higher performance"= "Support greater number of connections"), OCL constraints are specified (number of connection supported should be greater than X, cost of the solution should not exceed Y)	XSM Editor and Analysis Tool: Constraints are associated to the root element of the XSM model. The verification is run. It is identified that the DB server should be replaced.

Table 13 Typical Workflow - Preliminary Evaluation

[1] http://www.viewzone.org/momocs/index.php?option=com_content&task=view&id=40&Itemid=28

Understanding:

XIRUP Fragment	Tools Used
Model Analysis: After previous step a list of the components to be replaced should be determined.	Some search queries with KBR can be done in order to identify potential candidates for replacement. DB Server Oracle is found.
Model Transformation: Candidate components are integrated one by one using model transformation techniques. This step is repeated for each identified replacement component, in order to find the most suitable.	TT and "component replacement" transformation pattern are used to replace existing DB server with new one.
Model Analysis: Verification: The model constraints are rechecked in order to find out whether the new integrated model satisfies modernisation goals and to identify the most suitable replacement components. This step is repeated for each identified replacement component.	OCL Checker of Editor and Analysis tool is used for running verification.

Table 14 Typical Workflow - Understanding

A decision for continuing modernization can be finally done.

The next phases implement the actual modernization.

Building:

N/A for this scenario

Migration:

N/A for this scenario

7.3 Telco Specific Workflow

This section presents an example of the XIRUP methodology applied for a concrete Telco case study including method fragments and tools used.

7.3.1 JBilling Wrapping

The below tables provide additional information to the scenario described in ([D12])

Preliminary Evaluation:

XIRUP Fragment	**Tools Used**
Architecture recovery: J2EE Application model should be recovered from the sources.	J2EE Extractor reverse Billing Application source code directly into XIRUP model. Objecteering Java Developer is used to first reverse Java code into a UML model. XIRUP Transformation Tool is used to transform the UML model into a XIRUP model by means of a special UML2Xirup transformation for J2EE Architectures. XIRUP Editor and Analysis Tool is used for manual work for recreation or modification of the XIRUP Model.
Model Evaluation: Requirements Elicitation and Cost	Objecteering Scope Manager and XIRUP Editor and Analysis Tool are recommended

Estimation steps may require using of model evaluation method fragment.	for using. Objecteering Scope Manager provides a flexible requirements modelling framework fully compatible with XIRUP Tool suite. XIRUP Editor and Analysis Tool's OCL checker can be used for the gathering of the statistical information on the recovered XIRUP model and thus for the cost estimation.

Table 15 JBilling Wrapping - Preliminary Evaluation

Understanding:

XIRUP Fragment	**Tools Used**
Architecture Recovery is applied for building of the JBilling model.	The same tools (see previous phase) are used for this fragment.
Model Evaluation method is required for identification of the components to be wrapped.	Objecteering Scope Manager could be used for requirements analysis. XIRUP Editor and Analysis Tool should be used for building XSM model.
Architecture Modernization is applied for redesigning JBilling model.	XIRUP Editor and Analysis Tool should be used.

Table 16 JBilling Wrapping – Understanding

Building:

XIRUP Fragment	Tools Used
Model Evaluation is used when it is needed to identify required transformations.	XIRUP Editor and Analysis tool is used to evaluate the available system models.
Transformation Definition method is applied for creating missing transformations.	XIRUP Transformation Tool is used to define new transformations. Knowledge Base repository is used to store and locate transformations.
Architecture Modernization by Model Transformation is necessary for upgrading the system model with wrappers.	XIRUP Transformation Tool is used for running particular transformations or transformation patterns to fulfil required activities.

Table 17 JBilling Wrapping – Building

Migration:

XIRUP Fragment	Tools Used
Transformation Definition	XIRUP Transformation Tool is used to define new transformations. Knowledge Base repository is used to store and locate transformations.
Model Transformation for creating a PSM	XIRUP Transformation Tool is used for running a particular XSM-to-PSM (UML) transformation.
Code Generation	Objecteering Java Developer and a special Objecteering macro are used to generate executable Java source code. Eclipse or other Java IDE are used for the adjustments of the code and potential debugging.

Table 18 JBilling Wrapping - Migration

7.4 Ind Specific Workflow

The workflow in the solution business is normally divided into two main phases. In the *bid phase* the provider is compiling a bid to participate in the tendering process of the customer for a modernization. All bids are then compared and at the end of this phase an offer is place by the customer.

In the execution phase the provider is going to develop and implement the modernization in an already exiting plant

7.4.1 Bid Phase

Preliminary Evaluation:

XIRUP Fragment	**Tools Used**
Architecture Recovery	XIRUP Editor and Analysis Tool for manual creation of the system architecture. XIRUP KBR is used for storing all associated artefacts. Objecteering Scope Manager is recommended for requirements modelling (sometimes other tools are requested by the customer e.g. DOORs). MS Office tools are used for cost calculations. Teamcenter is used for collaborative team work and as document storing and management

	system for all project documentation.
Model Evaluation	XIRUP Editor and Analysis Tool is used for the verification of the design constraints using OCL checker.

Table 19 Bid Phase - Preliminary Evaluation

Understanding:

XIRUP Fragment	**Tools Used**
Model Evaluation	XIRUP Editor and Analysis Tool for various queries of the system model.
Architecture Modernization	XIRUP Editor and Analysis Tool for manual modernization of the system architecture.

Table 20 Bid Phase – Understanding

Building:

XIRUP Fragment	**Tools Used**
Architecture Modernization	XIRUP Editor and Analysis Tool for manual modernization of the system architecture as preferred tools. Enterprise Architect, COMOS PT or Automation Designer (customer dependent) are to be used for detailed architecture.

	XIRUP Transformation Tool is to be used for applying predefined transformation patterns such as component replacement.
Model Evaluation	XIRUP Editor and Analysis Tool can be used for validating the OCL constraints associated to the system model.

Table 21 Bid Phase - Building

7.4.2 Execution Phase

Understanding:

XIRUP Fragment	**Tools Used**
Architecture Modernisation	XIRUP Editor and Analysis Tool can be used for updates within the model due to the negotiation phase. Enterprise architect and AutomationDesigner can be used accordingly. Teamcenter is again used as document storing and management system.
Transformation Definition	XIRUP Transformation Tool can be used correspondingly for adapting the constraints.

Table 22 Execution Phase – Understanding

Building:

XIRUP Fragment	Tools Used
Model Evaluation	XIRUP Editor and Analysis Tool can be used for further evolution of the model. The XIRUP Knowledge Base Repository for saving/comparing the system model progress. Enterprise Architect, Automation-Designer and system specific automation tools are used for the data exchange to the automation engineer.
Transformation Definition	The XIRUP Transformation Tool for creating new transformations.

Table 23 Execution Phase – Building

Migration:

XIRUP Fragment	Tools Used
Code Generation	A large number of solution, customer and system specific tools are used. Most of the time only code fragments are generated to the build manually the automation software (e.g. SPPA-T3000, PCS 7. or WinCC).

Table 24 Execution Phase - Migration

8 Interfaces between third party and MOMOCS tools

This section describes the operational interfaces between tools (third party and MOMOCS) that participate in the XIRUP method fragments. See sections 4 and 7 for a detailed explanation of the XIRUP method fragments, and their instantiation through the usage of available third party and MOMOCS tools.

Only interfaces corresponding to XIRUP method fragments participated by two or more different tools are described.

8.1.1 Architecture Recovery

The "architecture recovery" XIRUP method fragment can be implemented by the jointly usage of a third party UML2 Case Tool, like for example SOFTEAM Objecteering CASE Tool or Sparx Systems Enterprise Architect and the MOMOCS XSM Transformation Tool (TT). The UML2 Case Tool receives several code sources and generates one or more UML2 models represented as XMI files. MOMOCS TT receives: a UML2 model, a predefined (as part of MOMOCS TT) UML2 to XSM set of rules enclosed within transformations and generates a XSM XMI representation, which can be opened by the MOMOCS XSM Editor.

The interface between the third party UML2 Case Tool and the MOMOCS TT is the Eclipse workspace and the Eclipse import mechanism, described as follows (see the table below): the UML2 model is available from the file system (it was created through available code sources by the third party UML2 Case Tool). The user opens the MOMOCS TT (included as part of MOMOCS Suite, a Eclipse plugin set) and uses

the Eclipse Import mechanism (menu File/Import) to import the UML2 model into the Eclipse workspace. Then, MOMOCS TT opens that models by selecting it from the workspace, selects the UML2 to XSM transformation (accompanied within the TT or retrieved from the KBR) and apply it or, in case, add new rules to fine-tune it. If no suitable transformation is available, a new one is created. Afterwards, a XSM is generated and saved into the Eclipse workspace for further usage by the XSM Editor.

The interface operation signature included in the table below:

importModel (UML2 model): XSM

is a conceptual description of the process performed by the XSM TT.

Architecture Recovery (UML2 Case Tool)		
Third Party ToolUML2 Case Tool	**Input**	**Output**
	Code sources (Java, C++, etc)	UML2 model
Interface	*importModel (UML2 model): XSM*	
CASE tool or Sparx Systems Enterprise Architect)	**Input**	**Output**
	UML2 model UML2 to Xirup transformation	XSM transformedModel

Table 25 Tools interface description for the architecture recovery fragment using a third party UML2 case tool

This process can also be undertaken by using the MoDisco Toolbox provided by the Eclipse GMT MoDisco project for model-driven reverse-engineering.

What MoDisco proposes is a generic and extensible metamodel-driven approach to model discovery.

Model discovery is a general process composed of the two following steps:

- the retrieval of the information from an existing system according to specific metamodels
- the injection of this information into one or several models that conform to the same specific metamodels

The process above needs to be implemented by a "discoverer" which is nothing but a program, a tool, etc, designed for it.

The MoDisco ToolBox provides a set of discoverers among which we can find different components such as:

- "JAR2UML" tool for generating UML models form Java APIs
- "Visual Basic 6 Discovery Tool" for discovering the structural part of the Visual Basic source code
- "Java2 SE 5.0 Discovery Tool" for discovering a complete model out of the source code of a Java 5 application (then compliant to the J2SE5 metamodel).

Other MoDisco tools are currently available. For more information please see [MoDisco].

After getting models, the MOMOCS Transformation Tool can be used to create and run proper transformations able to generate XSM models. Please see table below:

Architecture Recovery (MoDisco)		
MoDisco Discoverer	**Input**	**Output**
	Code sources (Java, C++, etc)	SourceCode model (according to the specific metamodel, J2SE, etc.)
Interface	*importModel (SourceCode model): XSM*	
XSM TT	**Input**	**Output**
	Source Code model SourceCodeMM to Xirup transformation	XSM transformedModel

Table 26 Tools interfaces description for the architecture recovery fragment using MoDisco

Furthermore, the MOMOCS Suite provides its own discoverer (called "J2EE extractor") which directly generates XSM models from J2EE source code. In this case, no intermediate steps, thus interfaces, are required.

Architecture recovery method fragment described so far only consider the most frequent scenario of model generation from source code. However, in some modernisation processes, it is also required to generate models from knowledge unformed sources like textual documentation (MS Word documents, etc), execution logs, etc. In those cases, additional models can be extracted from those sources using Objecteering Scope Manager (OSM) that produces UML models that can be converted into XSMs using the UML2XSM transformation and the transformation tool.

Architecture Recovery (OSM)		
Objecteering Scope Manager	**Input**	**Output**
	Unformed sources (MSWord doc, execution logs, etc)	UML2 model
Interface	*importModel (UML2 model): XSM*	
XSM TT	**Input**	**Output**
	UML2 model UML2 to Xirup transformation	XSM transformedModel

Table 27 Tools interface description for the architecture recovery fragment using OSM

8.1.2 Automatic architecture modernisation

This XIRUP method fragment is implemented by the MOMOCS tools: XSM Editor and XSM TT. XA has created a XSM using the XSM Editor of MOMOCS suite Eclipse workbench. This XSM is stored within the user Eclipse workspace. Then the XA opens a XSM TT view in the same Eclipse workbench and applies a user-defined transformation (previously created using the XSM TT or recovered from the KBR) to the former XSM which is taken from the same shared Eclipse workspace. Therefore, the interface between the XSM Editor and the XSM TT is the common Eclipse workbench and workspace through which XSM files are shared.

The transformed XSM resulted of applying the user-defined transformation within the XSM TT is saved back into the shared workspace, where the XSM editor can open it again in order to show the transformed XSM to the XA. See a sketch description of this process and interfaces in the following table.

Automatic architecture modernisation		
XSM Editor	**Input**	**Output**
	-	XSM
Interface	Shared Eclipse workspace transformModel (XSM model, ATL transformation): XSM	
XSM TT	**Input**	**Output**
	XSM model user-defined XSM transformation	XSM transformedModel
Interface	Shared Eclipse workspace *openModel (XSM transformedModel): void*	
	Input	**Output**
XSM Editor	**Input**	**Output**
	XSM transformedModel	-

Table 28: Tools interface description for the automatic architecture modernisation fragment

As mentioned in the XIRUP methodology walkthrough, other transformation engines can be used in substitution of XSM TT. In case of oAW, Epsilon and Kermeta transformation engines, since they are also Eclipse plugins, the interfaces with the XSM Editor in this XIRUP method fragment are the same, so the afore described process is still valid. The only consideration is that transformations compatible with the XIRUP metamodel and those tools should be provide by the XA.

The XIRUP methodology is partially supported by the MOMOCS Tool Suite. In particular, the reuse of XIRUP artefacts is supported by the KB Repository Tool

(KBR). However, KBR may not satisfy some of the user's requirements, because of which it may be complemented or replaced by another third party tool. For instance, within the INDUSTRIAL case study, the artefacts stored within the KBR should also be stored within the TeamCenter tool, a Siemens data storage and management system (CMS) similar to KBR. In this case, KBR artefacts should be exported to the TeamCenter tool. This exportation is done through the Eclipse workspace and the file system, using the KBR export facility. The artefact is selected in the KBR Explorer view and exported into the Eclipse workspace. From TeamCenter, the artefact is imported (loaded) from the file system.

TeamCenter should not only be used in this architecture modernisation method fragment, but also in other XIRUP fragments that produce reusable intermediate or final artefacts: XSMs, transformation, source code, etc.

Architecture Recovery (TeamCenter)		
KBR	**Input**	**Output**
	- -	XSM ATL transformation
Interface	*Eclipse Workspace | File system* *exportModel (XSM model): XSM* *exportModel (ATL transformation): ATL*	
TeamCenter	**Input**	**Output**
	XSM ATL transformation	- -

Table 29 Tools interface description for the architecture recovery fragment using TeamCenter

8.1.3 Model Evaluation

The Model Evaluation method fragment can be implemented using the MOMOCS XSM Editor, MOMOCS KB Rep , MS Excel, Automation Designer or Enterprise Architect.

In case XSM Editor or Knowledge Base Repository are used no specific interface are needed.

In case specific UML case tools are used (for example Automation Designer or Enterprise Architect) the following interface is used:

Model Evaluation		
XSM TT	**Input**	**Output**
	XSM Model XIRUP to UML2 transformation	UML2 model
Interface	*openModel(UML2Model): void*	
Third Party ToolUML2 Case Tool (e.g. Objecteering CASE tool or Sparx Systems Enterprise Architect)	**Input** UML2 model (xmi or uml extension required depending on specific tool)	**Output**

Table 30 Tools interface description for the model evaluation fragment

In the case of MS Excel, it is possible to exploit one of its feature which allows opening XML files: since both XSM and UML models managed by the MOMOCS Suite are EMF compliant files, they are actually pure XML files. This let them be directly opened without any intermediate step by MS Excel itself. Table below describes this situation.

Model Evaluation (MS Excel)		
MOMOCS Suite	**Input**	**Output**
	XSM Model	XSM Model or UML Model generated by TT Transformation (both are XML files)
Interface	*openModel(Generic XML file): void*	
MS Excel	**Input** XML file	**Output**

Table 31 Tools interface description for the Model evaluation fragment using MS Excel

Objecteering Scope Manager provides a flexible requirements modelling framework fully compatible with XIRUP Tool suite for Model Evaluation Fragment. This framework is compound of editors dedicated to entering and managing requirements, goals, business rules and terms.

Model Evaluation (OSM)		
UML2 Objecteering Scope Manager tool	**Input**	**Output**
	XSM OFP	OFP requirements, dictionaries, goals, business rules.

Table 32 Tools interface description for the Model evaluation fragment using OSM

8.1.4 Quality Analysis

The quality analysis method fragment can be implemented using the MOMOCS XSM Editor and the UML2 Objecteering Metrics tool (a third party one). The XA uses the XSM Editor to save a XSM as an OFP project into the Eclipse workspace. Then, the XA launches the UML2 Objecteering Metrics tool and opens the OFP project from the Eclipse workspace directory in the file system. UML2 Objecteering Metrics tool generates the metrics and shows the results to the XA.

Quality Analisys		
UML2 Objecteering Metrics tool	**Input**	**Output**
	XSM OFP	OFP ModelMetrics

Table 33 Tools interface description for the quality analisys fragment

8.1.5 Verification

This XIRUP method fragment is implemented by the MOMOCS Suite tools without the need of additional interfaces: XSM Editor and OCL Checker which are integrated within the Eclipse workbench and both work on the same XSM loaded within the Eclipse workbench.

8.1.6 Code generation

Code generation method fragment is participated by the MOMOCS Suite tools: XSM Editor and the XSM TT, and by the third party tool UML Objecteering. XA has a XSM

obtained by modelling with the XSM Editor or applying transformations through the XSM Transformation Tool and saves it within the shared Eclipse workspace. This model is opened by the XSM TT (working integrated with the XSM Editor within the same Eclipse workbench) from the workspace, in order to apply an available (recovered from the KBR or created from scratch by the XA) XSM2UML2 transformation to obtained a UML2 representation of the former XSM. This UML2 model is saved to the workspace.

Then the XA launches the UML Objecteering tool, and opens the UML2 model (a file having XMI extension) from the Eclipse workspace directory within the filesystem.

Finally, UML Objecteering tool generates the source code corresponding to the opened UML2 model, which is saved back to the file system. See this process depicted in the table below.

Code Generation		
XSM Editor	**Input**	**Output**
	-	XSM
Interface	Shared Eclipse workspace *transformModel (XSM model, XSM2UML2 transformation): UML2*	
XSM TT	**Input**	**Output**
	XSM model XSM2UML2 transformation	UML2 model
Interface	Eclipse workspace and file system *generateCode (UML model): SourceCode*	
	Input	**Output**
UML Objecteering	**Input**	**Output**
	UML2 model	Source code

Table 34 Tools interface description for the code generation fragment

Source code produced by this method fragment can be compiled in one or more execution units (as required) during the building phase. Those units are them deployed into some execution environment, such as, for instance, Siemens PCS 7 and SPPA-T3000, WinCC.

8.1.7 Simulation and testing

Simulation and testing method fragments are not enough supported by available tools so the interfaces between the MOMOCS Suite and those tools cannot be clearly

defined. In any case, and considering the previous method fragments, we can foreseen any potential integration between the MOMOCS Suite and third party tools through a file sharing interface using the Eclipse workspace and the shared filesystem.

8.1.8 KBR Note

In the different XIRUP method fragments discussed in this document, whenever it is required, only the XSM Editor and the XSM TT interact with the KBR, always through the common Eclipse workbench and workspace interface, so no iterations between

9 Annexes

9.1 Acronyms and Glossary

Abbreviation	Description
AST	Abstract Syntax Tree
CRM	Customers Relationship Management
EWFM	Enterprise WorkForce Management
FDD	Feature Driven Development
MDE	Model Driven Engineering
MOMOCS	MOdel driven MOdernisation of Complex Systems
PIM	Platform Independent Model
PSM	Platform Specific Model
RUP	Rational Unified Process
SME	Small Medium Enterprise
SPEM	Software Process Engineering Metamodel
TBMS	To be modernised system
XIRUP	eXtreme end-User dRiven Process
XSM	XIRUP System Model

Formatted: Bullets and Numbering

9.2 Reference Documents

[DOW] MOMOCS – Description of Work

[D11] D11 - Methodology Requirements

[D12] D12 – Methodology Requirements: Industrial and Telecom Use-Cases

[D13] D13 – State of the Art

[D21] D21 – Supporting Tools Requirements

[D53] D53 – Process Modernization Tool

[KumWel92] Kumar, K. and Welke, R. J. 1992. Methodology Engineering: a proposal for situation-specific methodology construction. In *Challenges and Strategies For Research in Systems Development*, W. W. Cotterman and J. A. Senn, Eds. John Wiley Information Systems. John Wiley & Sons, New York, NY, 257-269.

[SPEM] OMG Software Process Engineering Metamodel http://www.omg.org/technology/documents/formal/spem.htm

[WFALL] Royce, Winston 1970, "Managing the Development of Large Software Systems", Proceedings of IEEE WESCON 26(August): 1-9.

[V] Mark Hoffman & Ted Beaumont: Application Development: Managing the Project Life Cycle, Mc Press

[SPI] Barry Boehm. 1988. A Spiral Model of Software Development and Enhancement.http://www.sce.carleton.ca/faculty/ajila/4106-5006/Spiral%20Model% 20Boehm.pdf

[RUP] Ivar Jacobson, Grady Booch, and James Rumbaugh. 1999: The Unified Software Development Process, Addison-Wesley Professional

[Szyperski 2002] Clemens Szyperski: Component Software: Beyond Object-Oriented Programming. 2nd ed. Addison-Wesley Professional, Boston 2002

[XP] Beck, K. 1999. Extreme Programming Explained: Embrace Change. Boston, MA: Addison-Wesley