DG III Industry- IT Programme

MISSION - ESPRIT PROJECT  N° 29 656

| | | |
|---|---|---|
| Title | : | **Deliverable D24**<br>**Final Report from MISSION Project** |
| Document type | : | Report |
| WP/Task | : | 5.1. Technology Transfer and Dissemination |
| Document ID | : | MISSION-5-1-20011112-CU-D24-1-9 |
| Version | : | 9 |
| Date | : | November 12, 2001 |
| Status | : | D24 Issued |

| | | |
|---|---|---|
| Organisation | : | LU/CU |
| Editors | : | K. Popplewell (Coventry University)<br>J. A. Harding (Loughborough University)<br>M. Rabe (Fraunhofer IPK) |
| Authors | : | J. Arroyo (Sisteplant)<br>B. Eberhard (vr-architects)<br>M. Fischer (ProSTEP)<br>G. García de Gurtubai (Sisteplant)<br>J.A. Harding (Loughborough University)<br>F-W. Jaekel (Fraunhofer IPK)<br>J. Morazo (EADS CASA)<br>K.-U. Nonnenmacher (Bosch)<br>K. Popplewell  (Coventry University)<br>M. Rabe (Fraunhofer IPK)<br>D. Weinelt (Bosch) |

| | | |
|---|---|---|
| Distribution | : | PUBLIC |

| | | |
|---|---|---|
| Purpose of Document | : | Deliverable D24 |

| | | |
|---|---|---|
| Document history | : | 2001-04-04: First structure decided on technical meeting<br>2001-07-10: Added Substructure and Responsibilities<br>2001-08-01: IPK Input Update<br>2001-08-29: CU Merged all partners' D24 contributions<br>2001-08-31: IPK Added Chapter 6 Draft<br>2001-10-16: Final Draft<br>2001-11-12: Issued |

## Summary

This document is the final public report on European activities within the MISSION IMS Project. As such all aspects of research undertaken within the MISSION Europe are summarised. Readers wishing to have access to more detail should refer to project partners to discuss the terms under which such detail may be released.

Following Chapter 1, which is an introduction to the MISSION project, Chapter 2 describes the general architecture adopted by MISSION in applying distributed simulation to support the Manufacturing Systems Engineering Process.

Chapter 3 describes the principles of Manufacturing Systems Engineering Integration and Chapter 4 the reference model for integration.

Chapter 5 describes the MISSION demonstration scenario and Chapter 6 details MISSION dissemination activities.

## Content

# 1 Introduction

## 1.1 The IMS MISSION Project

At the beginning of the 1990's there had been a number of estimates about the increase of simulation application in Europe and comparisons to other regions, especially the US. Looking back, most of these prognoses have turned out to be wrong. It is a fact, however, that the great leaps forward in simulation development always followed leaps in industrial development. This applied to material flow simulation which really took off when material flow complexity increased in manufacturing and logistics systems. In a similar way flexible simulation systems have been induced by the development of Flexible Manufacturing Systems in the 1980s.

As a consequence we expect a new great leap forward in the field of simulation, resulting from the development of globally distributed or virtual enterprises. The IMS MISSION project aims to support simulation applications of this type through a specific new platform (Rabe 2000).

This report documents the major results of the MISSION, as far as they are public. Further information can be received from the IMS MISSION web (www.ims-mission.de). Companies interested in MISSION work are invited to join the Open MISSION Interest Group (OMIG, see chapter 6.4).

The MISSION projects consists of four major work packages:

- a requirement definition (closed)
- the development of an integration infrastructure
- the development of application components
- test cases and demonstrators
- test cases and demonstrators.

The software and internal mechanisms are the intellectual property of the consortium. However, the interface description is public, and third parties are encourage to enrich their software with MMP interfaces. The necessary documents are:

- D13 MMP Interface Description and
- D18 MMP Enterprise Design Moderator Interfaces and Messages.

During their work, the MISSION consortium has found several generic interface elements, which will need standardisation in the future. These elements are documented within

- D21 Input to standardisation working groups

All public documents are available at the MISSION web site.

## 1.2 Infrastructure

Figure 1.1 shows a sketch of the architecture implementation for the MMP. The general goal of MISSION is the support of the Manufacturing System Engineering (MSE) process by integrating the tools which could be employed in various aspects of the process. However, in order to enhance and accelerate the process, they must specifically refer to both, the usage of simulation, and the integration of simulation tools with other software tools. The suggested architecture therefore incorporates both the MSE and the distributed simulation of the manufacturing processes (Jäkel and Arroyo 2000).

Its purpose is to allow sharing information generated by any one component amongst all components that may have an interest in that information.

The intention is that the set of software agents which could be linked through this RTI, is open. Other agents could be attached to extend the range of manufacturing systems engineering activities integrated. So the set shown in the figure is representative only.

The communication will mainly be realised by the MS-RTI and the MSE integration infrastructure. The decision was to use a mechanism developed by the United States Department of Defence's (DoD).

Some years ago the situation at DoD was similar to the situation of modelling and simulation of global distributed enterprises. Many simulation models have already existed. The necessary effort in developing new simulation models increased. At the same time the available man power for the development decreased. So, it became more and more important to reuse existing systems and existing information in a distributed environment. Moreover, it became important that it was possible to substitute systems within a runtime environment (for example to substitute an old jet by a new version). The DoD initiated a research program to develop an architecture for distributed simulation. One result of this approach is the high level architecture (HLA) (Kuhl 1999).

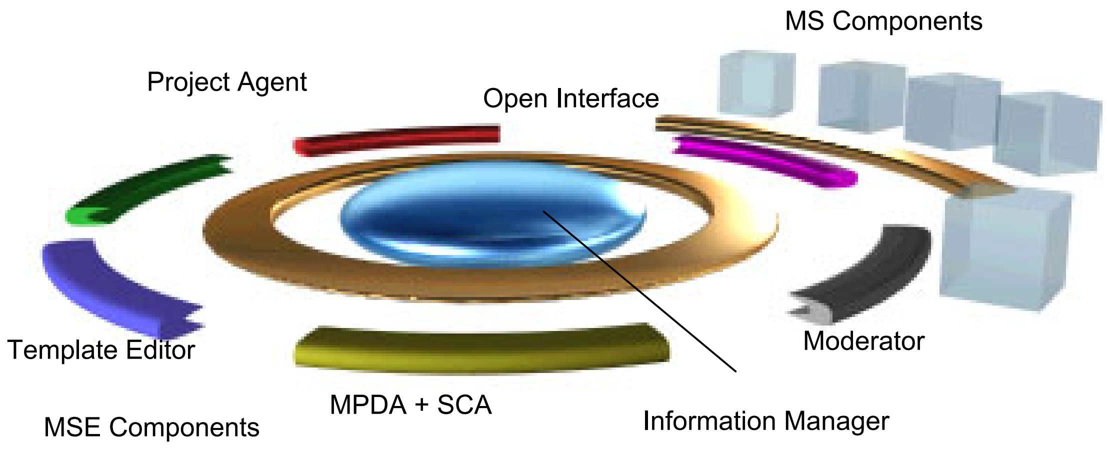


**Figure 1.1: Sketch of MMP Architecture**

*"The DoD's High Level Architecture for modelling and simulation (HLA) can certainly be regarded as the state of the art in distributed simulation. This holds true for the military simulation community per se, since this is the origin of HLA."* (Schulze 1999).

The HLA is the glue that allows to combine computer simulations into a larger simulation. The HLA supports the combination of different simulations into a single, comprehensive simulation.

The HLA satisfied many requirements for distributed simulations e.g. time synchronisation, communication between independent simulation models etc. A disadvantage of the HLA is the dependence of the interface description to the HLA-RTI from the simulation model. An idea how to solve this problem includes the description of the Simulation Manager.

## 1.3 Application Components

The application components within the MISSION platform can be classified into

- Template Libraries and
- Manufacturing System Engineering (MSE) Design Agents

The template libraries are based on the reference model concept developed by Fraunhofer in the period of 1994-1998 (Mertins et al.,1996, Rabe et al,1998, Wenzel, 2000).

### 1.3.1 Template Libraries

All templates are modelled, independently, from a special simulation tool. Templates that are already available in simulation tools refer to these tools. There might be more than one simulation template for the same purpose implemented in different simulation systems. Supporting software allows the selection of templates from the library, specialisation to meet the current project requirements and specifications how simulators must interact.

### 1.3.2 MSE Design Agents

MSE Design agents are software advisors supporting modelling and application within the MISSION modelling platform. A design agent runs autonomously and co-operatively with some communication functions in the distributed environment in terms of simultaneous engineering. It advises and navigates users with intelligence gained from artificial intelligence technology and knowledge collected from experienced engineers.

The concept of design agents is open, i.e. companies from outside the MISSION consortium are encouraged to develop and market their own design agents compliant with the MISSION platform. However, a few design agents will be essential and delivered together with the platform. One of those agents is the Simulation Manager.

### 1.3.3 Simulation Manager

The Simulation Federation Manager uses the information from the template library to construct a distributed simulation model. It further supports the design of simulation experiments, creation of FEDEX (Federation Execution) and executes the design. The results of experiments are predictions of the Manufacturing Process Performance, which the simulation manager makes available to other components through the Manufacturing System Engineering Integration Infrastructure.

The definition of the connection of different simulation models requires a configuration of the involved simulation systems. The HLA – RTI has such a configuration possibility based on an ASCII file. This RTI configuration file includes class names and attribute names of objects, which are exchanged between the simulators. The relations between the simulators default values and object instances are not described in this file. Therefore the simulation manager generates specific federate configuration files (FCF) in addition to the FEDEX which includes extended information.

A configurable interface for simulation models decrease the adaptation costs and increases flexibility, especially for complex simulation systems.

## 1.4 References

Jäkel, F.-W.; Arroyo Pinedo, J.S.: Development of a Demonstrator for Modelling and Simulation of Global Distributed Enterprises. In: Mertins, K.; Rabe, M. (Eds.): *The New Simulation in Production and Logistics. 9th ASIM Dedicated Conference on Simulation in Production and Logistics, Berlin 2000,* pp. 375-384.

Kuhl, F.; Weatherly, R.; Dahmann, J.: *Creating Computer Simulation Systems*. Prentice-Hall 1999

Mertins, K.; Rabe, M.; Friedland, R.: Referenzmodell Fertigungssysteme - Effiziente Simulation bei größerem Nutzen. *10. Symposium Simulationstechnik*, Dresden 1996, pp. 95-100.

Rabe, M.; Wenzel, S.; Ohle, F.: Reference Models for Simulation. In: Bargiela, A.; Kerckhoffs, E. (Eds.): Simulation Technology: Science and Art, *Proceedings of the European Simulation Symposium ESS'98, Nottingham, 1998*, pp. 503-507.

Rabe, M.: Future of Simulation in Production and Logistics: Facts and Visions. In: Mertins, K.; Rabe, M. (Eds.): The New Simulation in Production and Logistics. 9th ASIM Dedictaed Conference on Simulation in Production and Logistics, Berlin 2000, pp. 21-43.

Rabe, M.; Friedland, R.: Neutral Template Libraries for Use in Globally Distributed Enterprises. In: Mertins, K.; Rabe, M. (Hrsg.*): The New Simulation in Production and Logistics. 9. ASIM-Fachtagung Simulation in Produktion und Logistik, Berlin. Eigenverlag, 2000*, pp. 385-394.

Schulze; T.; Straßburger, S.; Ulrich, K.;: Migration of HLA into Civil Domains: Solution and Prototypes for Transportation Applications; *SIMULATION* 73:5, pp. 296-303, Simulation Councils 1999.

Wenzel, S. (Ed.); *Referenzmodelle für die Simulation in Produktion und Logistik*. Delft: SCS 2000, pp. 233-255.

# 2 General Architecture (HLA)

## 2.1 Introduction

The general goal of MISSION is the support of the Manufacturing System Engineering (MSE) process by integrating the tools, which could be employed in various aspects of the process. However, in order to enhance and accelerate the process, they must specifically refer to both, the usage of simulation, and the integration of simulation tools with other software tools. The suggested architecture therefore incorporates both the MSE and the distributed simulation of the manufacturing processes. Figure 2.1 shows the outline of a architecture implementation for the Mission Modelling Platform (MMP).

The Manufacturing System Run Time Infrastructure (MS-RTI) supports execution of distributed simulations. The components of the distributed interface are specified by the Simulation Manager, who also determines which simulation runs are necessary, and accepts output from the runs. It is envisaged that the MS-RTI will permit access to Real Production Systems as operational PPC systems and Shop Floor Control Systems to incorporate their activities into the simulation. Within this platform, time synchronisation mechanisms (independent from calendar time) are essential.

The MSE integration infrastructure provides the mechanism for interfacing software agents that participate in the Manufacturing System Engineering Process. Its purpose is to allow for sharing information generated by any one component amongst all components that may have an interest in that information.

The intention is that the set of software agents, which could be linked through this RTI, is open. Other agents could be attached to extend the range of manufacturing systems engineering activities integrated, so the set shown in the 2.1 is representative only.

The following are prerequisites for a comfortable use of the Mission Modelling Platform (MMP):

- Manufacturing Process - Run Time Interface (MS-RTI)
- Manufacturing System Engineering Integration Infrastructure (MSE Integration Infrastructure)
- Information Manager
- Project Agent
- MSE Moderator
- Template Library
- Simulation Manager.

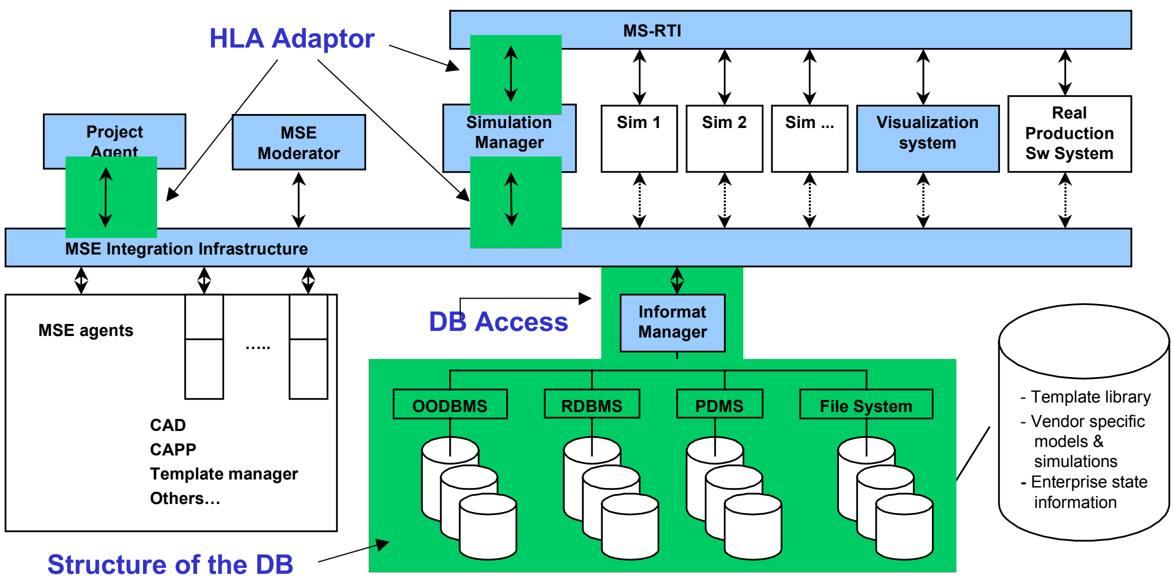


**Figure 2.1: MISSION Architecture Implementation (European Module)**

The communication will mainly be realised by the MS-RTI and the MSE integration infrastructure. The MSE integration infrastructure is realised using TCP/IP and special services provide by the Information Manager. For the MS-RTI  a mechanism developed by the United States Department of Defence's (DoD) is used.

Some years ago the situation at DoD was similar to the situation of modelling and simulation of global distributed enterprises. Many simulation models did already exist. The necessary effort in developing new simulation models increased. At the same time the available budget for the development decreased. So, it became more and more important to reuse existing systems and existing information in a distributed environment. Moreover, it became important that it was possible to substitute systems within a runtime environment (for example to substitute an old jet by a new version). The DoD initiated a research program to develop an architecture for distributed simulation. One result of this approach is the high level architecture (HLA) (Kuhl 1999).

The HLA is the glue that allows to combine computer simulations into a larger simulation. The HLA supports the combination of different simulations into a single, comprehensive simulation.

The HLA satisfied many requirements for distributed simulations for example time synchronisation, communication between independent simulation models etc. A disadvantage of the HLA is, that the interfaces to the HLA-RTI depend of the simulation model. An approach to solve this problem is included within the description of the Federate Configuration Files (FCF) generated by the Simulation Manager.

## 2.2 Configuration of Simulation Scenarios

Unfortunately, within military applications of the HLA each new model will receive new and genuine into interfaces,  typically. Therefore, a flexible interface for simulation models is not required for military applications. This is completely different within civil domains, where the total effort spent to one simulation study is extremely low, compared to defence applications. The dependency of the interface description to the HLA-RTI from the specific simulation model is a critical disadvantage for regular civil applications of HLA.

Within MISSION an approach has been established that allows flexible configuration of interfaces of tools within a simulation scenario. Examples for such tools are

- simulators: Taylor, Arena, etc.
- other tools: Excel, order management systems, etc..

This approach is based on a template library approach and the generation of a federate configuration file for each simulation interface as well as the generation of the federation execution file for the HLA-RTI in parallel (Figure 2.2). The major advantage of this approach is that changing simulation models or adding completely new ones requires only changes of the configuration, but no re-programming. Of course, those interfaces need to be related to a specific application field. Within the MISSION project, this application field is manufacturing systems (including virtual enterprises and logistics).

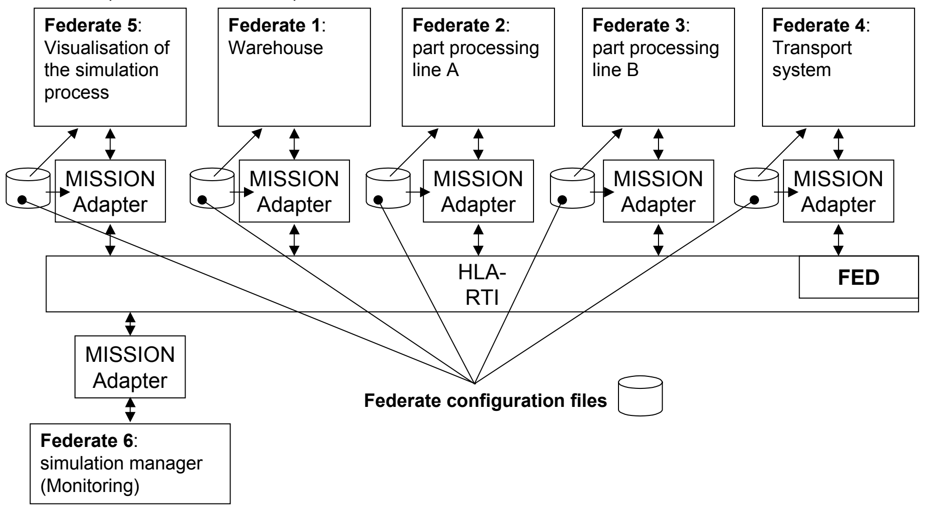

Federate 5: Visualisation of the simulation process
Federate 1: Warehouse
Federate 2: part processing line A
Federate 3: part processing line B
Federate 4: Transport system
MISSION Adapter
MISSION Adapter
MISSION Adapter
MISSION Adapter
MISSION Adapter
HLA-RTI
FED
MISSION Adapter
Federate 6: simulation manager (Monitoring)
Federate configuration files

**Figure 2.2: Execution of a Simulation Scenario**

## 2.3 Template Library

A central structure within the MISSION approach is the template library. Together with the MISSION repository it allows a flexible definition of classes, attributes and objects. The template library consists two kinds of templates:

1. Templates defined to structure the object space within the template library. This template structure includes mechanisms to support template library wizards for easy application.
2. Application templates to derive building blocks which are used for the definition of simulation scenarios.

The application template connects the following different structures consistently (Figure 2.3):

- **referred simulation models and description of the template behaviour**
  One or more simulation models can be referred which satisfy the template specification. This specification can be described in a natural language or in a formal language like XML. The simulation model has to be created within a simulator, which provides an interface to the MMP.
- **description of exchanged objects**
  Exchanged objects are those objects which are necessary to define a communication with other templates.
- **visualisation of the template**
  Each application template needs one or more possible representations within graphical views or animations. Within a first approach, this will be realised by a VRML description.
- **parameter descriptions of the template**
  Each application template has a set of attributes. A subset of these attributes can be directly linked to simulation parameters. Each simulation parameter requires a link to an attribute but not each attribute does necessarily link to a simulation parameter.
- **Definition of connectivity elements**
  One result of the research during the development of the template library approach is that a special kind of exchanged objects is necessary to describe the direct ("physical") connections between the templates. These objects are represented as input and output segments of the building blocks derived from the template library.

The template library content is not fixed. Users or groups of users may add classes and attributes at any time to fulfil their project requirements.

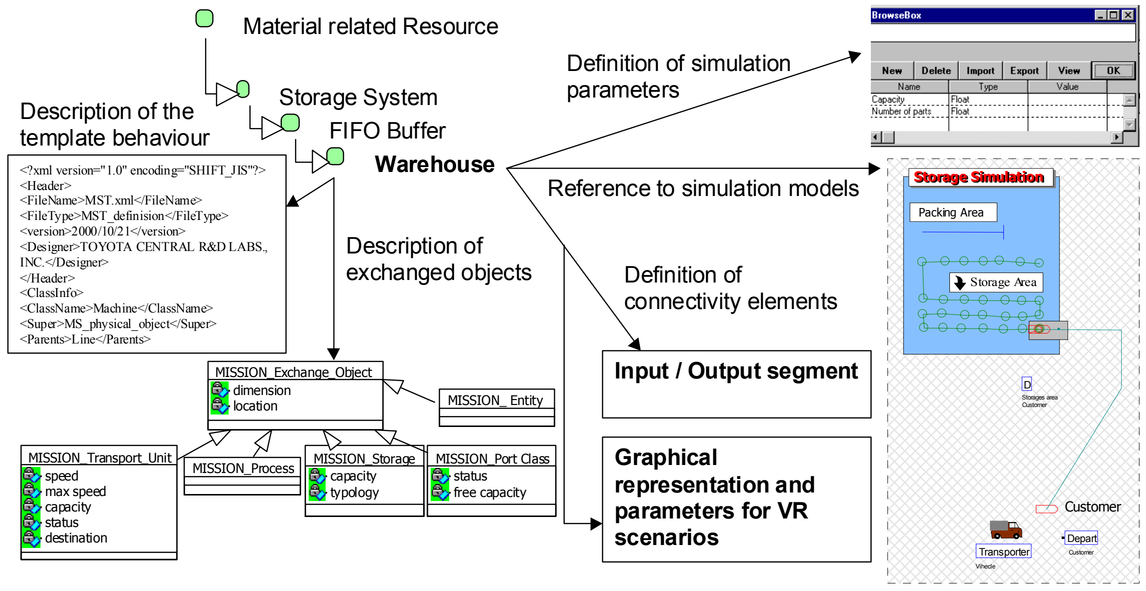

```
<?xml version="1.0" encoding="SHIFT_JIS"?>
<Header>
<FileName>MST.xml</FileName>
<FileType>MST_definision</FileType>
<version>2000/10/21</version>
<Designer>TOYOTA CENTRAL R&D LABS.,
INC.</Designer>
</Header>
<ClassInfo>
<ClassName>Machine</ClassName>
<Super>MS_physical_object</Super>
<Parents>Line</Parents>
```

**Figure 2.3: Description of an Application Template**

The design of a simulation scenario requires an additional construct. Building blocks (Figure 2.4) are used to design the simulation scenario. A template can be seen as a scheme of a building block. Therefore, a building block is defined as an object derived from a template of the template library.

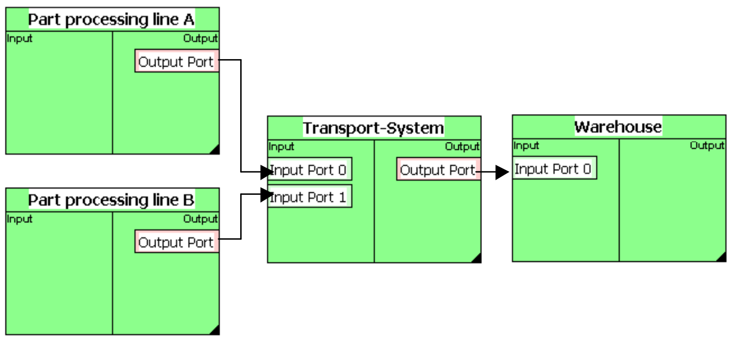
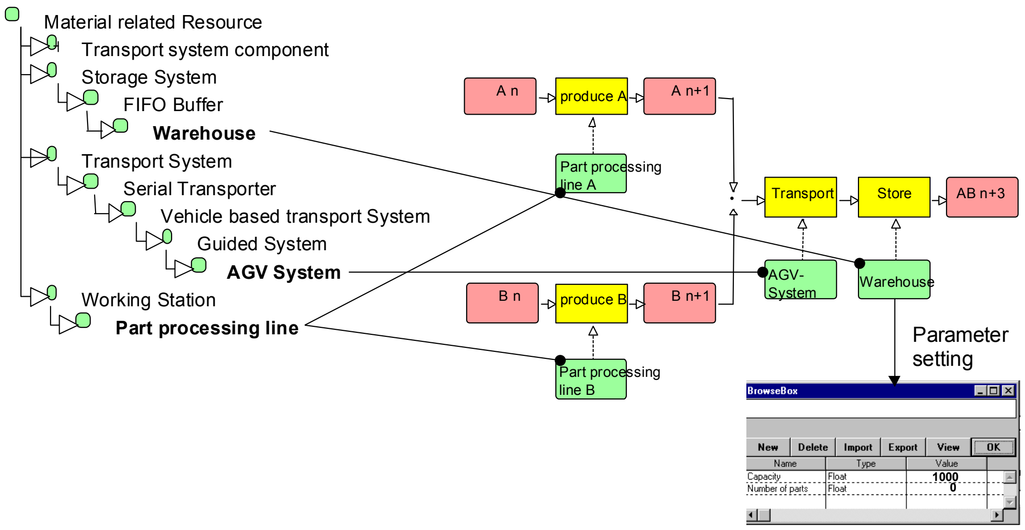


**Figure 2.4: Example for the Usage of Building Blocks**

## 2.4 Simulation Manager

Most functions of the simulation manager are based on the template library approach. It includes views to manipulate template classes, objects, building blocks and manufacturing system (MS) process models. Furthermore it supports the design and execution of simulation scenarios based on the template library mechanism. The MS process model is created during the MSE process and includes all objects needed during the MS process. Especially, machines, transport systems and storage’s with their attributes are specified within the process model. This MS process model can be used later to select a part of the MS process for the simulation (Figure 2.5).

The Simulation Manager uses the information from the template library and the information collected during the MSE process to construct a distributed simulation scenario. Therefore, the Simulation Manager acts as a bridge between the engineering of manufacturing systems and the simulation of such systems, also. It further supports the design of simulation experiments, the creation of FEDEX (Federation Execution) and it executes the design. The results of experiments are predictions of the manufacturing process performance, which the manager makes available, through the Manufacturing System Engineering Integration Infrastructure to other components.



| Name | Type | Value |
|---|---|---|
| Capacity | Float | 1000 |
| Number of parts | Float | 0 |

**Figure 2.5: Process Model and Template Library**

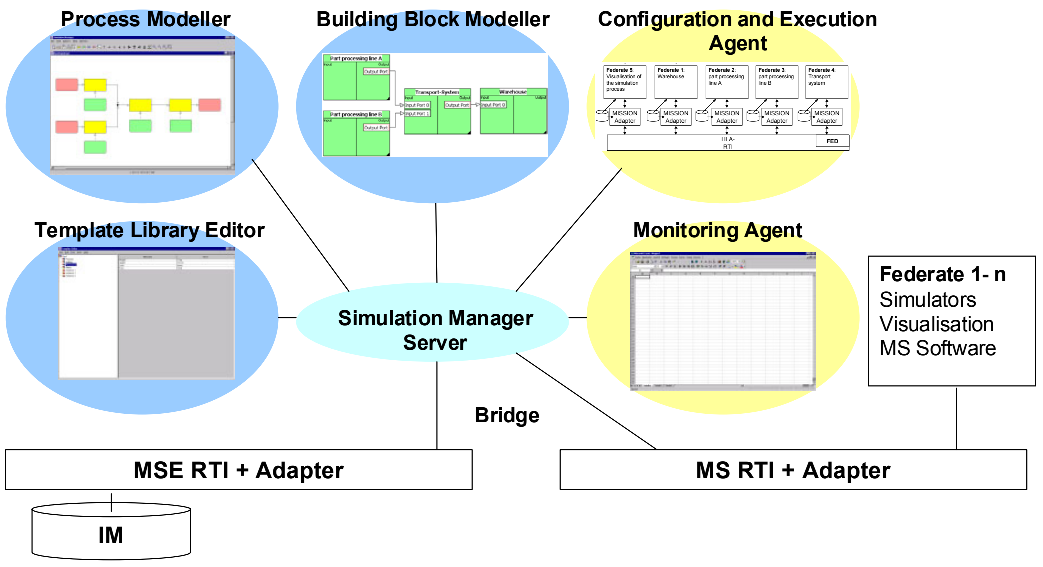


**Figure 2.6: Simulation Manager**

The Simulation Manager is needed to start up, initialise and manage the execution of the different simulations in a distributed simulation execution.

It co-ordinates the following different tasks consistently (Figure 2.6):

- the modelling of a distributed simulation scenario
- the consistent generation of the configuration files
- the execution of the whole simulation scenario
- the monitoring of the different components for detecting breakdowns, deadlocks, etc.
- the design of experiments by changing federate parameters.

## 2.5 Federate Configuration File

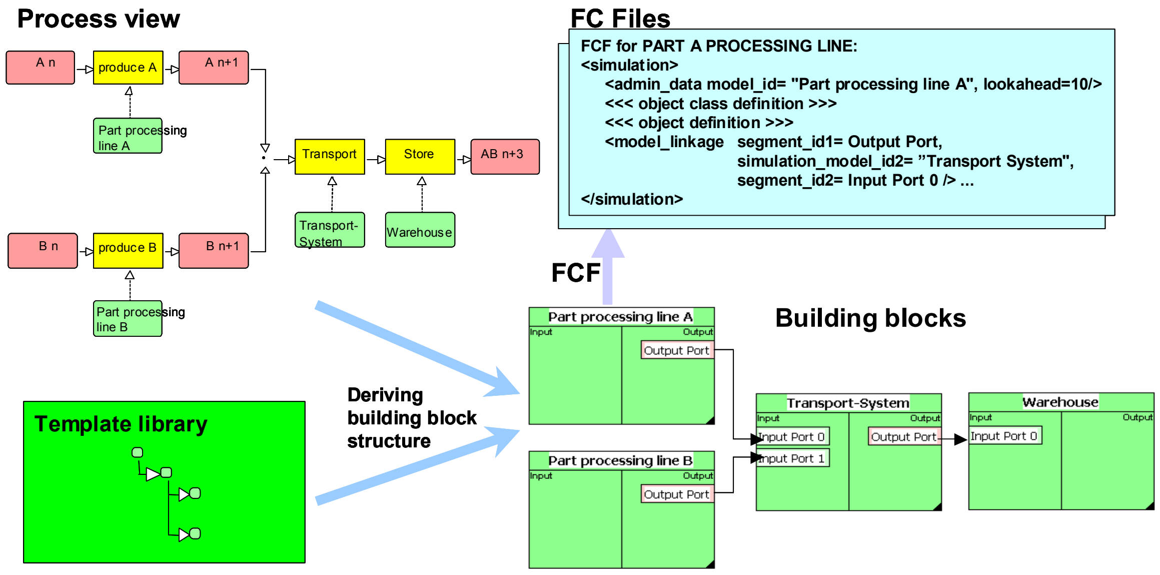
The federate configuration file (FCF) closes the gap between the HLA-RTI-FED-file and the interface definition of simulation models (federates). The description of the information within a federate configuration file (Table 2.1) is defined via XML.
The definition of the object and attribute names within the FCF according the FED-file are necessary, because otherwise a federate does not know the names of the involved exchange objects. The type information is important for a federate to extract information from the objects handled by the RTI. Default values could be defined to substitute an unnecessary simulation model. For example, the speed attribute of a vehicle is defined as an attribute of an exchange object and set by an engine simulator. The default setting of the speed attribute allows to leave out the engine simulator. In the same way, initial values allow to define a start status of a simulation model.

The setting of simulation parameters can be used to tune the simulation models. For example, two part processing lines A and B are used as building blocks. Then the performance of the processing lines can be defined for each building block by setting the performance parameters. So, the same model can be used in different incarnations. An additional requirement is the description of sequences.

| Content of the federation execution definition (FED) file used by the HLA-RTI | Content of the federate configuration files (FCF) used for simulation interface configuration of federates |
|---|---|
| • necessary class names of exchanged object classes<br>• attribute names | • class names of exchanged objects<br>• attribute descriptions of the classes including:<br>+ names,<br>+ type information,<br>+ default values<br>• initial values of attributes or parameters<br>• parameter settings (e.g. max vehicle speed)<br>• information about sequences (process flow)<br>• information about output and input relations |

**Table 2.1: Federation Execution Definition File (FED) vs. Federate Configuration Files**

Based on the information from the template library, the process model and the building block structure the Federate Configuration Files can be generated automatically for each federate (Figure 2.7).



```
FCF for PART A PROCESSING LINE:
<simulation>
    <admin_data model_id= "Part processing line A", lookahead=10/>
    <<< object class definition >>>
    <<< object definition >>>
    <model_linkage   segment_id1= Output Port,
                     simulation_model_id2= "Transport System",
                     segment_id2= Input Port 0 /> ...
</simulation>
```

**Figure 2.7: Generation of the Federate Configuration Files (FCF)**

## 2.6 Principle Connection of Simulation Models

The principle connection of simulation models is explained using an extremely simple example. The example shows the connection of three different simulation models (M1, M2, M3). Each model acts as a separate federate and the FED file includes only the class A (Figure 2.8). during the starting phase of the simulation the simulation interface of each federate subscribes the class A. Now each federate will be informed, if objects of the class A are created, deleted or changed. Within this process the federate configuration file (FCF) is used for each federate to get the classes and attributes which have to be subscribed. These activities  are only done within the simulation interfaces. The simulation models are not involved at this stage.

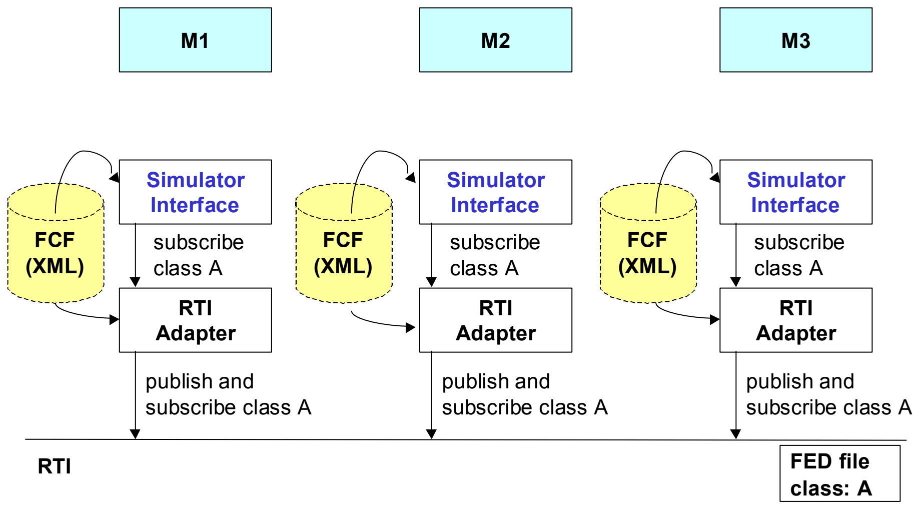


**Figure 2.8: Start of the simulation scenario**

In addition the federate identifier is extracted from the federate configuration file. Each federate within the described federation has such a unique identifier. This is used e.g. to indicate the next federate within a sequence of simulation models.

When each simulation interface has been successfully connected to the runtime infrastructure (RTI), the simulation run starts. In this example the federate M1 creates an entity X of the class A. At the same time the simulation interface of the federate M1 creates a related RTI object for the entity X. This RTI object inherits the attribute "location" automatically from the class A because for every exchange object within this approach the attribute "location" is defined. This attribute represents the location or the next location of an exchange object. The location attribute includes a reference to the actual or next federate there this object exists. The identifier is just the unique Id of a federate.

So, after the object is registered to the RTI each simulation interface checks whether the location was set to their location. But the object is registered with the location M1. Therefore the other federates can ignore this object, at the moment (Figure 2.9).

Now the entity X passes the simulation model M1. If changes of the entity X occur during the simulation within the simulation model M1, then it is possible to register them to the RTI. However the example has only input and output connections between the simulators. So a change of the RTI object is only necessary after the entity X has passed the simulation model.

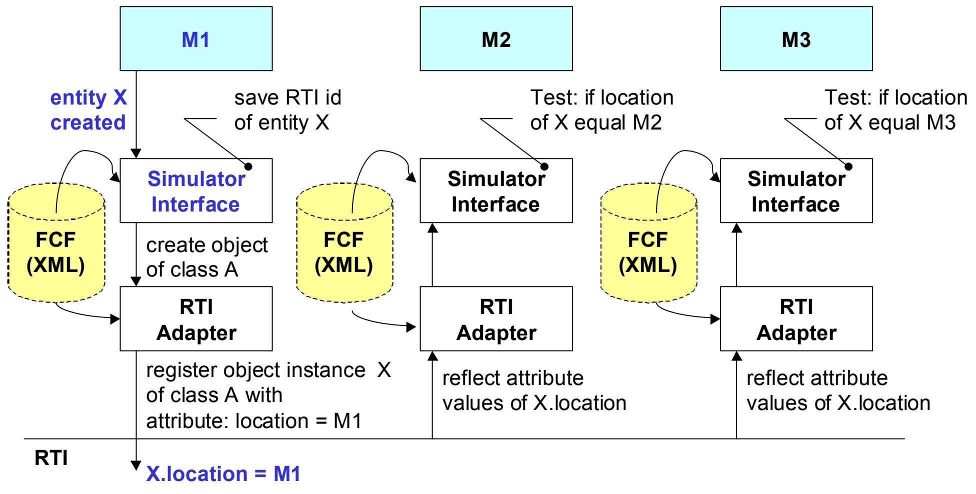

M1
M2
M3
entity X created
save RTI id of entity X
Test: if location of X equal M2
Test: if location of X equal M3
Simulator Interface
FCF (XML)
create object of class A
RTI Adapter
register object instance X of class A with attribute: location = M1
Simulator Interface
FCF (XML)
RTI Adapter
reflect attribute values of X.location
Simulator Interface
FCF (XML)
RTI Adapter
reflect attribute values of X.location
RTI
X.location = M1

**Figure 2.9: Creating of an entity**

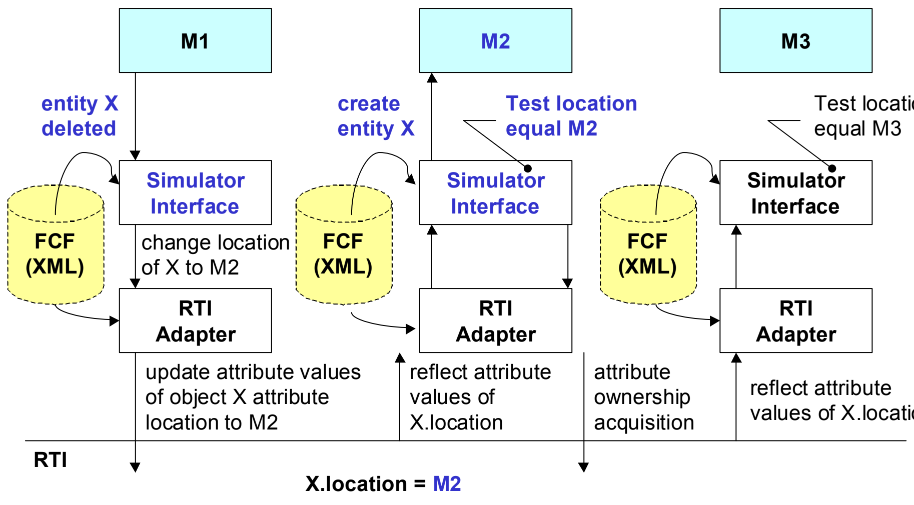


**Figure 2.10: Exchange of an entity**

When the entity X finishes its life within the simulation model M1, several activities occur within the simulation interfaces. All necessary information is taken from the entity X. This information will be registered as an update of the object X to the RTI. Note the object X is still alive. The simulation interface can read the new location of the object X from the federate configuration file. So at the same time the update of the location attribute to M2 will be registered to the RTI (Figure 2.10), also.

Now without any delay the simulation interface of M2 recognises the new value of the location attribute. It compares the new value of the location attribute with its owns. After the simulation interface realises that the new value of the location attribute is M2, then it initiates the creation of the entity X within the simulation model M2 using the content of the RTI object X.

This process will continue until the entity X leaves the simulation model M3.

## 2.7 MISSION Modelling Platform Data Model

### 2.7.1 Introduction

This section describes the specification of the MMP data model, which is the result of discussions between the European MISSION partners.

The focus of WP2 was to define and implement an enterprise data model for the simulation application area. The data model will be implemented in an Integration Platform.

The specification is based on the requirements of the WPs 1, 2, and 3. Additionally it is influenced by approaches to enterprise modelling by the partners (IPK – Integrated Enterprise Modelling, LU – Factory Data Model) as well as experiences in product data standardization (ProSTEP – EXPRESS, STEP).

Emphasis has been placed on the following principles:

- the specification of basic elements
- a generic way to specify the data model ( to enable implementations to satisfactorily meet the specification)
- detailed elements are used if their necessity is known
- concepts are presented if the necessity is known but the detailed requirements are not yet known

### 2.7.2 The Scope of the MMP Data Model

Each application system that participates in the MSE process uses its own data model because it is designed for the application optimally. But the communication between different applications needs a common agreement – an agreement which is valid for all MSE applications, i.e. all MSE and Project Agents, MSE Moderator, Simulation and Information Manager.

The MMP data model should be a common data model for the data objects that will frequently be shared by the MSE components that use the MSE Integration Infrastructure to exchange data.

#### *2.7.2.1 Principles*

##### *2.7.2.1.1 Implementation Independence*

The MMP data model is an implementation independent specification. This independence is necessary because of the various implementation techniques that may be used. The final decision has not yet been made as to which single method or data base software system should be used (see MISSION-2-3-20000303-SISTEPLANT-1-1).

Some of the requirements described in chapter 3.2 reduce the possibilities of choice (e.g. the inheritance requirement). But in general the proposed data model should be able to be implemented with most available data base systems - however with different efficiency in programming and use by applications.

##### *2.7.2.1.2 A Generic Approach*

The development of the MMP data model was strongly influenced by the advances made during the MISSION project. This process is not finished. Some details will be changed over the whole project life cycle. Therefore the data model has to be an open model. That means extensions as well as adjustments should be possible.

A generic data model that specifies the basic data elements and provides a way of extending the model, best satisfies this requirement. Because of this, many detailed data objects, which are known from the MISSION application scenarios or from the experiences of the partners, were not transferred to the data model.

In this way the MMP data model is an open specification. It can be adjusted to suit many different applications. On the other hand, time and effort spent in implementations can be reduced during the initial experimental tasks of MISSION. The results of these tasks can be used to guide further work and implementations, which can use a more detailed data model based on the experiences of the MISSION project.

#### *2.7.2.1.3 Shared Information*

The MISSION project covers many application areas like simulation, enterprise modelling, design, control and libraries. The data model is not able to and must not cover all these areas completely. It has to focus on those data, which are to be exchanged between individual applications.

The data model supplies two ways to exchange data:

- explicitly over data elements with defined context
- implicitly over documents in different formats, whose contents are only described verbally.

The latter can be interpreted only by the original systems or by means of special converters.

#### *2.7.2.1.4 Abstract Data Model*

The documented MMP data model is an abstract data model. Influenced by the first and second principles above, no aspects of implementation are considered. The reasons for this are that neither the implementation method (relational or orbject-oriented) nor the selected software system should influence the design of the data model.

A consequence from it is that during the implementation phase additional specifications will have to be done.

### ***2.7.2.2 Requirements***

#### *2.7.2.2.1 Application Scenarios*

The MISSION application scenarios describe the scope from the application point of view. They specify the parts of the whole world that are  significant to the MISSION project. Therefore the data model must be able to describe at least these sections.

#### *2.7.2.2.2 Inheritance and Polymorphism*

As stated previously, the data model is an open one, which can be improved during the application programming phase. It is meaningfuland necessary that previously defined features are preserved, so they can be used later on in the refined model. This corresponds to an object-oriented approach to data modelling.

If class hierarchies are used, for example common associations can be made in the parent classes. These relationships would then be inherited by all sub-classes of the parent classes.

In the same way methods, specified for a parent class, can be inherited by the child classes. The methods may be overridden by the child classes (polymorphism). By making use of polymorphism, the user can be given appropriate information, without needing to know all the details of the class hierarchies.

#### *2.7.2.2.3 History Mechanism*

It is insufficient to know that the value of an object in the model has changed. It must also be possible to interrogate the model or the objects to find out what the change is, e.g.  to find out what the current value is and what the previous value was.

A complete history mechanism is not included in the MMP data model because it is too complex a task. But there are two components which are parts of such a mechanism:

- each MISSION item may have a time stamp, thus a temporal sequence can be specified
- historical sequences can by specified by common relationships

In addition,  an implementation of the data model should use any support provided by the implementation system in respect of this topic.

#### *2.7.2.2.4 Decomposition and Abstraction Support*

In certain situations it is useful to be able to consider the enterprise as a whole, whilst in other situations more detailed and particular parts of the enterprise need to be examined. For example, it is necessary to be able to consider all the resources or activities within a factory as a whole, or to be able to examine part of the factory, such as a cell, in greater detail.

The MMP data model supports this requirement by using open class structures. This class structure is allowed to be extended by the applications, because it is not meaningful as well as not possible to specify all hierarchy levels of all objects completely. An example is given by predefining the class Facility and its sub-classes Enterprise, Factory, Shop, Cell, and Station.

The MP data model is an expandable class structure.

##### *2.7.2.2.5 Association Mechanism*

Relations from one object to other objects are needed.

That is the reason for the generic class Relation. Each MISSION item may reference any other items via a relationship. The relationships may be specialized or may be grouped using the type attribute.

##### *2.7.2.2.6 Partial Models*

Template libraries are provided to help users build new models faster. Templates can take two forms,

- they can be subsets of classes from the data model or
- they can be populated or partially populated instances of objects from the data model.

A template library contains a collection of different templates, catalogued to facilitate appropriate template selection. A partial model is

- a model or design of part of an enterprise, or
- a model or design created for a limited, particular purpose, or
- a model or design in the course of preparation, so it contains some limited information, but this is not yet complete.

##### *2.7.2.2.7 What-if-scenarios*

Designers must be able to work and experiment on parts of the design before commiting additions and changes to the current version of the enterprise model.

Therefore the classes Project and Project change as well the Approval are introduced.

##### *2.7.2.2.8 Object Versioning*

A versioning concept like those in the PDM area is not included. Such a consept would require that all objects (e.g. MISSION items) shall be represented by at least one version and any relation from one object to another one has to be related via the versions. There is not such a strong requirement in MISSION at this moment. The requirements in MISSION should be satisfied by the usage of the Relation class.

##### *2.7.2.2.9 Documents*

The MMP data model will be the data model of a repository implementation. This repository will be the physical storage for the shared data of the MSE processes. In Additional to the objects (specified in the class structure of the data model) documents exist, for example the documentation of a project, the native files of a layout design, drawings, CAD models, different representations and views of objects, and so on. Some of them will be shared in their original format too. Thus it is necessary to store them as files  or to reference them at their storage location.

##### *2.7.2.2.10 Specification Method*

At the beginning of the MISSION project it was intended to specify the MMP data model in EXPRESS and its graphical representation EXPRESS-G because EXPRESS is the only one standardized data modelling language.

In the course of the project the position of the majority of the partners has been changed. They prefer the Unified Modeling language UML. Thus the diagrams in this deliverable are UML diagrams. They are created using the system Rational Rose 98.

### *2.7.2.3 What is in Scope?*

The class structure of the MMP data model (Figure 2.11) gives a first, short overview of the elements of the data model. It does not include any attributes and associations. More details are presented in the next figures. They are subdivided into different groups to support different views  of the data model.

Strong emphasis must be placed on the fact that the data model is an open one and can be extended by implementations. The proposed data model represents a framework for MISSION implementations.

### *2.7.2.4 What is Out Of Scope?*

#### *2.7.2.4.1 Application Specific Data that are not Shared*

Each application system connected to the MSE Integration Infrastructure generates a lot of data. But only those data are considered by the MMP data model that are shared by at least two partners of the MSE Integration Infrastructure. Any other is out of scope.

They can be store using the classes Black box or Native file. But this is only a storage mechanism. There is no context for their interpretation by other components.

#### *2.7.2.4.2 Interface Specification*

The MMP data model specification includes no interface specifications. This is the subject of other MISSION research.

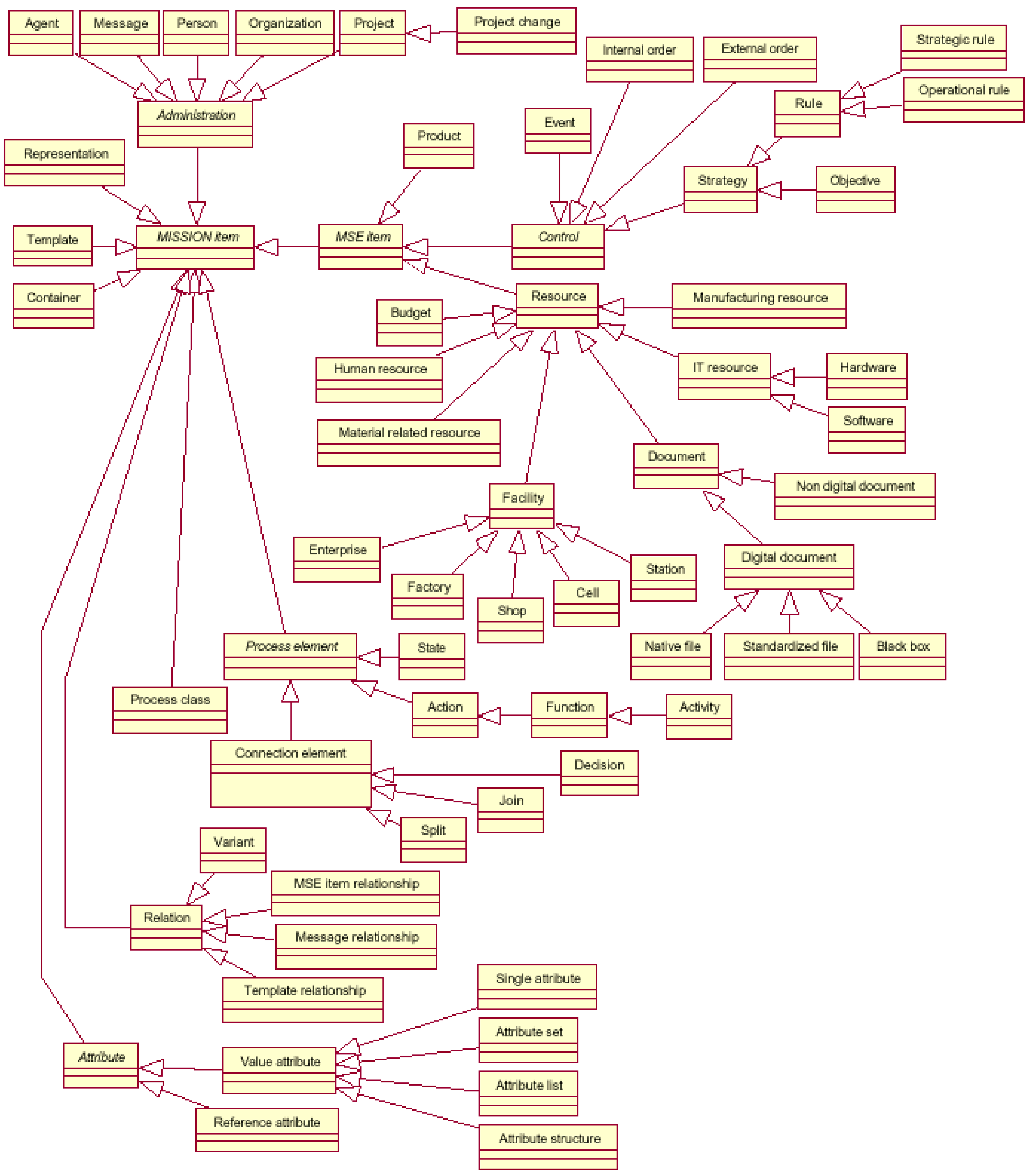


**Figure 2.11: Class structure of the MMP data model**

### *2.7.2.4.3 Basic Elements*

The root class of the MMP data model class structure is the class MISSION item. It specifies attributes and associations that are inherited by all of its sub-classes.

The development state of a single piece of data can be specified by an 'approval status'. The attribute 'time stamp' (should be set automatically by the Informatiion Manager component) and this allows a detailed designation of a time sequence if required.

Later on specified sub-classes will require additional attributes. It can be specified by using 'attributes', but if it has to be inherited it should be specified in the sub-class directly. Then the Attribute class concept should be used as a framework.

Each MISSION item may have one or more representations as well as relations to other MISSION items.

### *2.7.2.4.4 Administration Data*

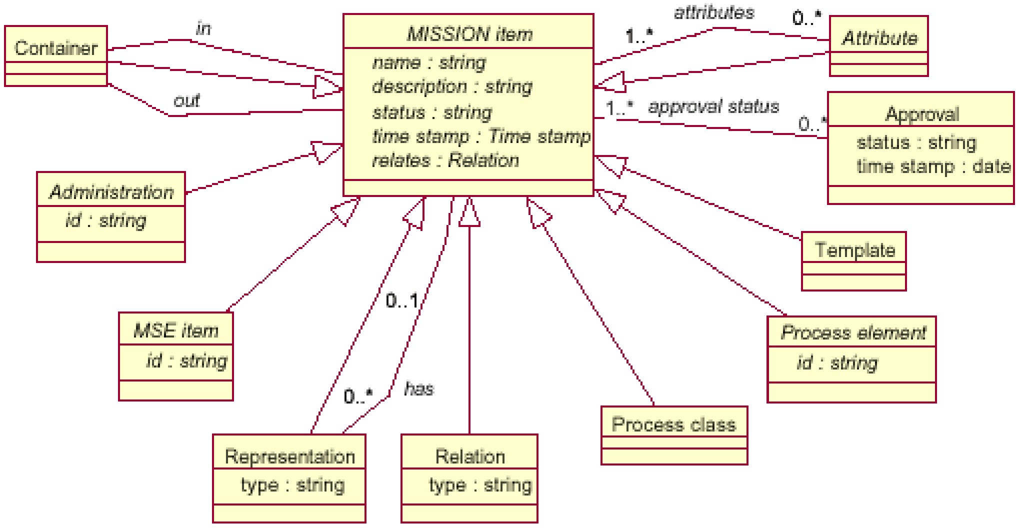


**Figure 2.12 : MISSION item class structure**

The repository of the MSE Integration Infrastructure requires some administrational data to organize the MSE processes. This group of data classes is summarized in the class Administration.

It allows the description of different projects and project stages, persons and organizations as well as agents that are involved in the MSE processes. In this context Person and Organization are not used as a resource in the enterprise model but in an organizational context to identify them as responsible for something inside the MSE implementation system. Agents are the application systems linked to the MSE Integration Infrastructure.

The class Message supports the retention of messages which are of interest for a longer time or that are marked as not delivered (may be the recipient is not online).

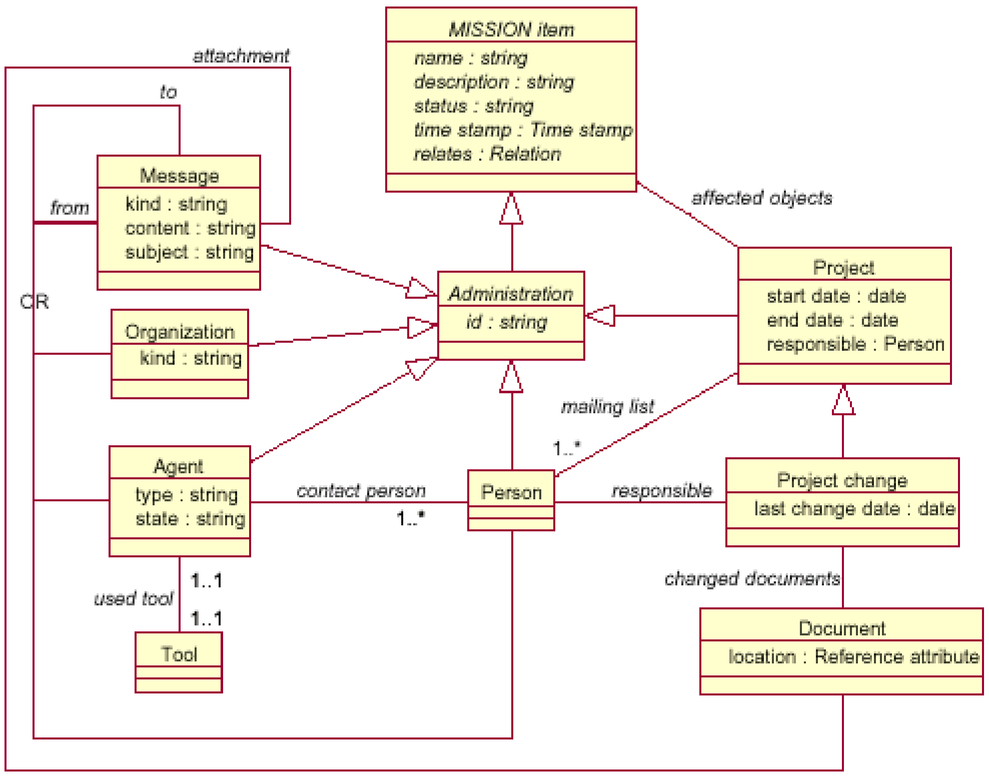


**Figure 2.13 : Administration class structure**

### 2.7.2.4.5 *Enterprise Model*

The class MSE item as the basic element and its sub-classes (see Figure 2.14) shall be used to specify enterprise models.

The class Product represents all objects that are processed and/or produced by processes. This class is not specified in detail in this document. This specialization will be a task of the further MISSION documents and is mainly influenced by the MISSION scenarios.

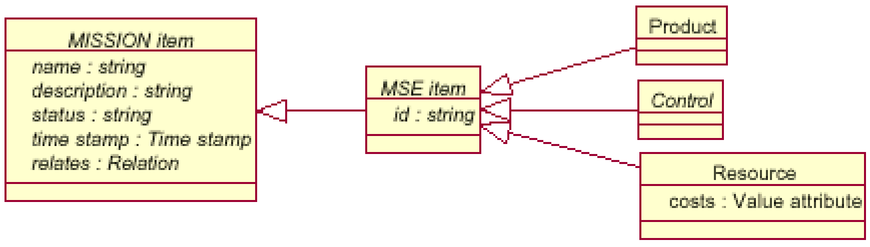


**Figure 2.14 : MSE item class structure**

The class Control involves all MSE elements that have a controlling functionality in the MSE processes (see Figure 2.15). These are

- (internal and external) orders to do something,
- events that influence a MSE process and
- strategies for 'How something has to be or may be done?' as well as 'What is the reason for something?'.

Note: The class Strategy redefines the scope of the inherited attribute 'controls'.

Resources in the MSE process summarize all elements that are required to do a process or process step in a specified way (see Figure 2.16). Thus any physical (persons, machines, tools, facilities) and non-physical (documents, budget) things represent a resource in the MSE process if they are necessary to realize a process (step).

Documents have an extraordinary role in this context. They can be a resource itself as well as an additional information to any other MISSION item. In general documents represent a specific view of something e.g. drawings, CAD models, tables of data, documentations, ond many other more. The digital documents are of special interest in this context. They can be classified into different groups referring to the possibility to interpret their content. This classification is focused on the format used to store the information in a document file. Three main groups can be distinguished

- standardized (in the sense of data exchange: STEP, IGES) or quasi standardized (common used software: WORD, EXCEL, HTML, graphical formats (like BMP, GIF), and other)
- native files, i.e. using the internal file format of the generating software system
- black boxes with no knowledge about the content

All these groups are represented by sub-classes of the class Document.

Note: It is possible to store detailed information of a document inside the MSE repository (driveways out of a plant layout drawing). But there is no mechanism proposed to recover all data after changing it.

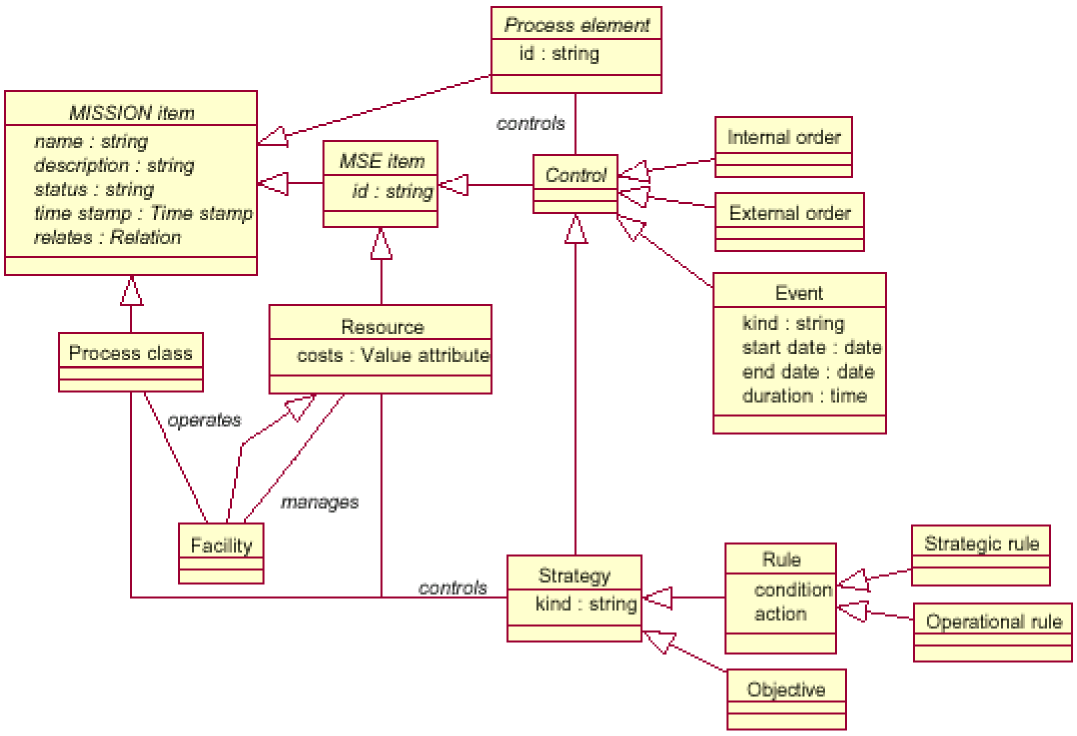


**Figure 2.15 : Control class structure**

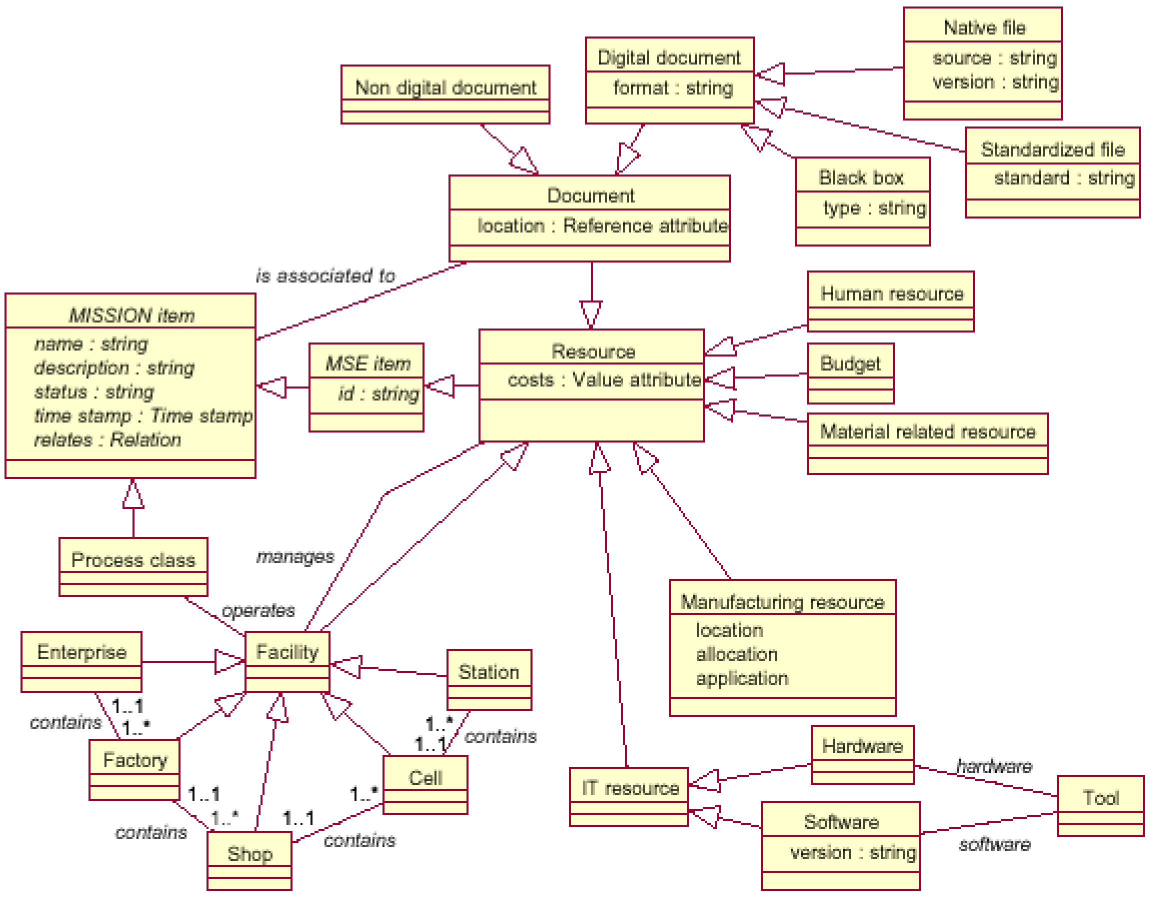


**Figure 2.16 : Resource class structure**

### *2.7.2.4.6 Template Library*

Template libraries are an important component in the MSE process. They are specified using elements of enterprise modelling as well as elements of MSE process modelling. Additionally they may have a relation to the simulation aspects. There may be implementations for different simulation systems.

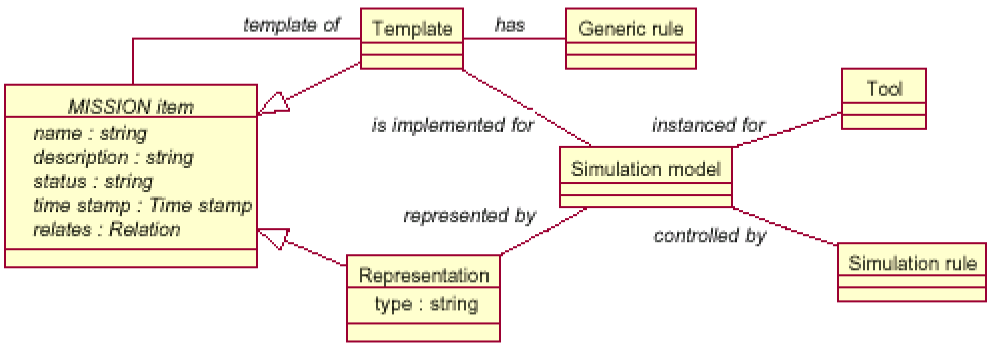


**Figure 2.17 : Template class structure**

### *2.7.2.4.7 Specification of Relationships*

The relation from any MISSION item to any other (set of) MISSION items is a basic requirement. Thus relations can be grouped. Some special relationship are pre-defined. They are characterized by the equality of data types of all in the relationship involved MISSION items.

The sub-class Variant represents a 1:1 relation between two instances of the same data class. It can be used for example to specify relations like 'is replaced by' or 'is an alternative for'. At the same time it is the compromise to a complete versioning concept.

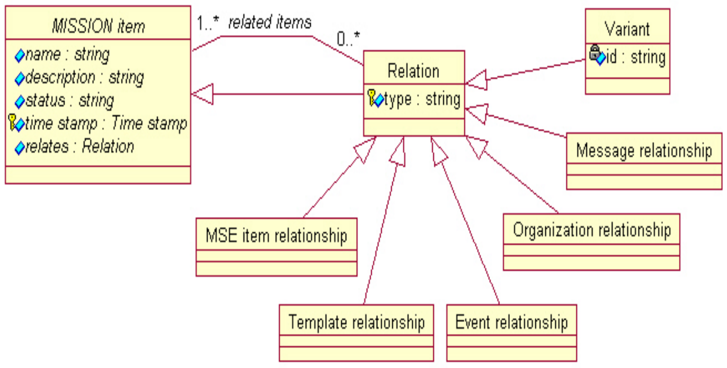


**Figure 2.18 : Relation class structure**

### *2.7.2.4.8 Attribute Specification*

The proposed concept of attributing MISSION items is a generic approach. Later on during the implementation phase of the MISSION project it has to be specified in detail by referring to the used implementation method and database system. The concept includes:

- references to other MISSION items as well as to the location where data are stored outside the repository
- valued attributes with different complexity (simple, sets, lists, structures)
- valued attributes with units.

Because units cannot be restricted to SI units, since physical quantities are not the only values which may be utilised, Unit relationship was introduced. This construct allows reference to be made to other Units, tables (for instance with actual exchange rate of currencies), formula, or textual descriptions.

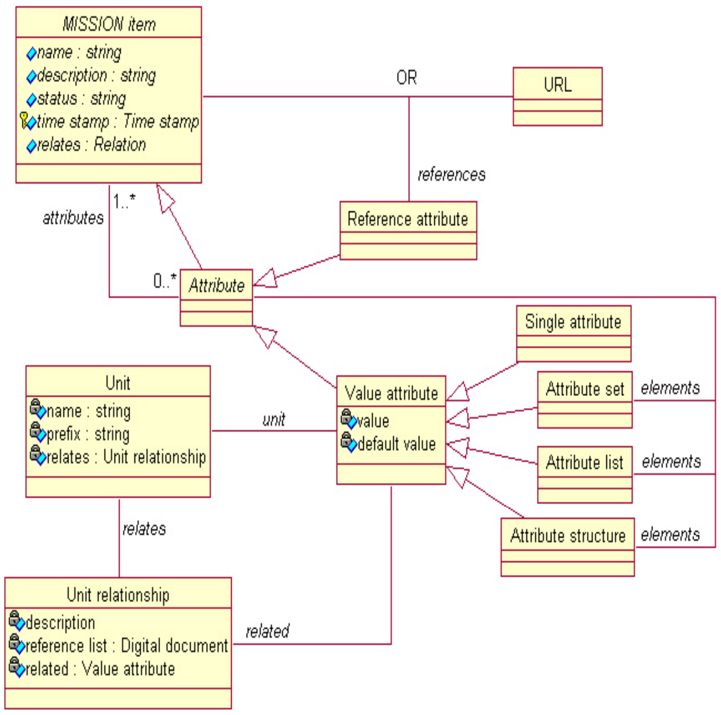


**Figure 2.19 : Attribute class structure**

### 2.7.2.4.9 *MSE Process Modelling*

Within a process model processes or process steps are specified by the class Action. The Action is related to a process class. Because it is not possible to describe all existing process classes, the process class structure has to be defined dynamically. The result is a link to a Process class as attribute of the action.

The class State specifies different stages of an object during its life cycle (requirement, design, production, usage, recycling). Additionally there are relations between the various stages of different objects.

The transformation from one determined (beginning) state to another determined (ending) state is described by the class Function.

Finally, the Activity specifies the Control, which controls the execution of the Function, and the Resources, which are in charge of executing the Function.

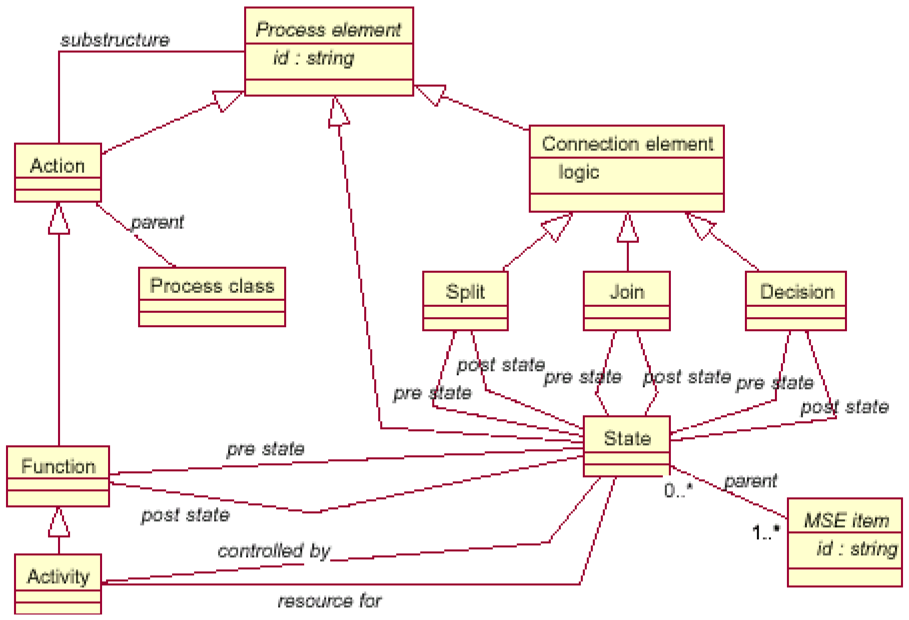


**Figure 2.20 : Action class structure**

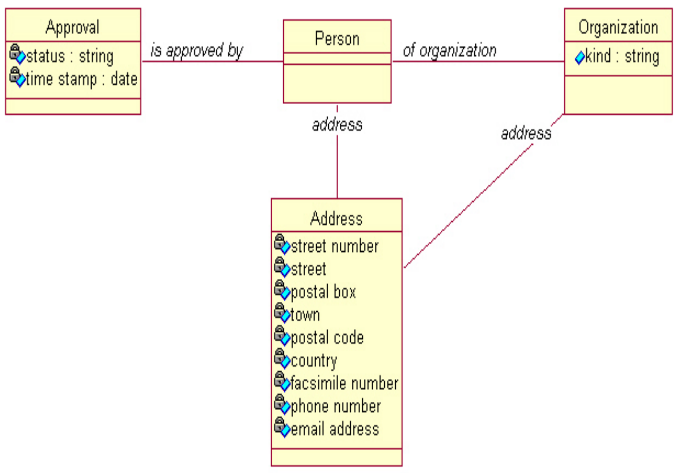


**Figure 2.21 : Approval class**

# 3 Manufacturing System Engineering Integration

## 3.1 Information Manager

### 3.1.1 Introduction

The MMP Data Management Module consists mainly of a Design Agent known as the Information Manager and associated databases and filesystems. This agent store all the information provided by other agents, so it can be used in the Mission platform.

The Information Manager should be able to store the MMP data model. The Requirements for this data model were:

1. It should be able to describe the parts of the whole world that are significant to the MISSION project.
2. It should allow multiple inheritance. The data model could be improved during the application programming time.
3. It should preserve a history mechanism. At least, it should be able to keep on the current and the previous value of an attribute.
4. It should support decomposition and abstraction, so open class structures were needed.
5. It was needed an association mechanism, so relations from one object to other objects were needed.

With these requirements an object oriented approximation for the Information Manager was needed. This does not mean that the database should be object oriented.

Steps followed to achieve this result have been:

1. Database Selection
2. Implementation of the data structure in the selected database (Oracle)
3. Construction of the Information Manager access methods.

### 3.1.2 Database Selection

#### *3.1.2.1 Database Characteristics*

A  little review of the different database systems available in the market was performed to see the principal differences between them.

First of all let see the differences between relational and object oriented database systems, especially when Java or C++ programming languages are used to develop object oriented applications.

The relational model is simple but it is very different from the object model. Object Databases employ a data model that has object-oriented aspects like classes with attributes, methods and integrity constraints. They extend the functionality of object programming languages to provide full-featured database programming capability.

Relational database systems are good at managing large amounts of data and for data retrieval but provide little support for data manipulation. The relational model is based on two-dimensional tables where each item is a row and where the relationship between each item is given comparing the values stored in these tables.

Object oriented programming languages are good at expressing complex relations between objects, and the object model is based on the integration of code and data, dynamic extensions of  class hierarchies, flexible data types...

The most important points we could use to choose a database system are:

- Support for Object Oriented Programming
- Simplicity of use
- Simplicity of development
- Extensibility and content
- Complex data relationships
- Interoperability and middleware

- Performance Vs safety
- Distribution, replication and federated databases
- Scalability
- Product maturity
- Legacy people programs and databases, and the universality of SQL
- Software ecosystems
- Vendor viability.

The importance of these characteristics vary with the application that must use the database.

The compared database products were:

- Poet
- Versant
- Oracle8.

#### *3.1.2.2 Final Decision*

Object oriented databases are coming out of use because of their complexity, poor performance and cost. Instead, Oracle has an easy structure, proved performance and low cost.

One important point in the Mission project is the exploitation of the product. It is necessary to carefully analyse the market situation of each database. In Spain, Versant or Poet products are not so well introduced as Oracle's ones. Oracle's database has the best commercial opening in this country. These two reasons have had a major role in making the decision of programming the first implementation of the Information Manager under Oracle.

Despite Oracle's object orientation, it has been decided not to use this functionality because of the following reasons:

- When a retrieval or insertion of information from / to the database is performed, it is necessary to know the class structure of the object. This information may not always be known.
- It is not possible to dynamically add / delete attributes to the classes, which is a specification of the Mission Data Model.
- There is no need to implement methods in the class structure. This simplifies considerably the structure (encapsulation and polymorphism is not necessary).

The implemented solution consists on an object structure built over relational tables. The access to the Information Manager is based on object oriented operations, The Information Manager translates these operations to relational tables. The cost of implementing this mechanism is lower than the problems that could be derived by the higher cost and less commercial future of object oriented databases.

### 3.1.3 Oracle Implementation

#### *3.1.3.1 Oracle General Information*

For the database management module, an Oracle 8i version (release 8.1.5) has been selected. Besides, for every Oracle release, different products are distributed, to cover as many as possible of today's enterprise and personal needs. These versions are:

- Server Edition: It is the standard database, with full connection capabilities.
- Enterprise Edition: It is the Server edition, but with added capabilities, such as multiprocessor working, that may be of interest for big enterprises.
- Personal Edition: It only allows connections from one computer at a time, although there may be several different users defined. It is mainly directed to people who want to have complete Oracle capabilities, but access it always from the same computer.
- Lite Edition: It is a smaller version of the program, designed specifically for portable systems.

Whenever the installation is started, there are also several different software programs availbale. These are:

- Oracle 8i Server: It is the database engine in itself. Of course, must be installed to run the Data Management Module.
- Oracle 8i Client: Software related with access to remote databases. It is not required in this case, as we are using a Type 4 JDBC driver to access Oracle (see chapter 5.3.2: JDBC access to Oracle).
- Oracle 8i Programmer: It adds tools and interfaces to develop applications accessing the Oracle database. It's not necessary for the Data Management Module.

For the Data Management Module, the software to be installed is the Oracle 8i (also known as Oracle 8.1.5) Personal Edition, and only the Oracle Server.

#### *3.1.3.2 Oracle Structure*

Oracle is a relational database. It uses a particular structure that groups elements at different levels, allowing different kinds of relationships between them, as it can later ease understanding of the selected structure solution (see chapter 4.4: Oracle Implementation).

There are some definitions that must be understood:

- Objects: Objects are places in which data is stored. The different kind of existing objects are:
  - Tables: basic information storage unit
  - Indexes: optional structures that improve performance of the system
  - Views: particularized presentation of data. Can be used to restrict access to other user, or simplify end-user's understanding
  - Synonyms: Aliases of other tables or views, belonging to the same or to other scheme.
  - Program units: Stored SQL or PL/SQL procedures that can be executed.
- Schemes: In Oracle, all data is stored in schemes. Each user owns one and only one scheme, so that it can be said that a scheme is composed of all the different objects one user owns. Still, users may have rights to access other user's schemes, so there must be a way to uniquely identify objects that belong to different schemes. To achieve this uniqueness, objects are identified by a SchemeName.ObjectName structure. If no SchemeName is provided, the current user's scheme is assumed.
- Tablespaces: Tablespaces are logical structures in which data is stored. There is no direct relationship between schemes and tablespaces. Different objects of the same scheme may be stored within different tablespaces, while a tablespace may contain objects from different schemes. They may be used to associate similar tables from different schemes
- Datafiles: These are the physical files, in which data is stored. A tablespace may be stored in several datafiles, although a datafile may not contain more than one tablespace.

Security restrictions in Oracle are performed via permission granting. This permissions can be of two different types:

- System privileges: They allow actions to be performed on a type of objects. For example, delete tables, create view...
- Object privileges: They allow certain actions to be performed over certain objects belonging to a different scheme. This way, a user can access, modify or delete objects that have been created by a different user.

### *3.1.3.3 Implementation*

#### *3.1.3.3.1 Object Attributes Information*

Although Oracle presents object-orientation capabilities, they are going to be ignored in the Data Management Module, as it was discussed in chapter 3.5. Still, the user must see the structure as if it was object-oriented, so that there must be some way to implement inheritance, attributes and so on. To achieve this appearance of object-orientation, relevant information must be stored, such as inheritance, attribute types and values... All this information is stored in four tables. The names contain the project name, so that different projects may define different objects without interfering with each other.

- PROJECT_INH: This table has just two columns: CLASS_NAME and PARENT_NAME, and it identifies single relations between classes, so that the inheritance tree can be discovered. Although each row identifies one and only one class-to-parent relationship, each class may appear several times in this table, thus allowwing multiple inheritance to be implemented. In the Oracle implementation, a class with no parents inherits from an empty class named ROOT, to allow us to know when does the inheritance tree end.
- PROJECT_TYP: In this table, all rows have two columns: OBJECT_NAME and CLASS_NAME. OBJECT_NAME is an identifier to distinguish this object from all other objects, while CLASS_NAME identifies the class this object belongs to.
- PROJECT_ODEF: In this table, the type of all defined attributes is stored. It has three columns: CLASS_NAME, ATT_NAME and ATT_TYPE. CLASSNAME is the class this attribute belongs to, while ATT_NAME is the name of the attribute and ATT_TYPE is its type. ATT_TYPE may be one of a given list (“STRING”, ”INTEGER”, ”TIME”...) or it can be a class name.
- PROJECT_OBJ: Finally, this table stores all values of every attribute of each object. It has three columns: OBJECT_NAME, ATT_NAME and ATT_VALUE. OBJECT_NAME identifies the object this attribute belongs to, while ATT_NAME and ATT_VALUE specify the attribute’s name and value respectively.

#### *3.1.3.3.2 Object Relationship Information*

Object oriented databases also allow relationships between objects, that have been used in the Data Model definition. For instance, a Message object is related with two Person objects, one of these is related via a To relationship, and the other one via a From relationship. To store all relevant information, only two tables are needed:

- PROJECT_RDEF: Within this table, all relationships are defined. Data such as relation name (RELATION_NAME column), classes which are being related (SOURCE and DESTINATION) and a concept called cardinality (CARDINALITY) are defined. Cardinality means that an object may be related only to one object (1:1 cardinality), to several objects (1:N cardinality) or several objects to several other objects (M:N cardinality). Relationships are directional. A To relationship between Message and Person is different from a To relationship between Person and Message, and thus the separation of a SOURCE and a DESTINATION class becomes clear.
- PROJECT_REL: In each row of this table, three columns are stored: relationship name (which must have been defined in RELATIONSHIP_DEFINITION), and the names of the two objects that are related. If a relationship has 1:N or M:N cardinality, an object may appear more than one time.

##### *3.1.3.3.3 Security Restrictions*

Oracle provides several security capabilities that are used in order to establish a read-write permission or denial policy. This policy is based in the layer structure of repository objects that is going to be implemented. A division of objects (mainly templates) into three different categories has been stated:

- core Mission objects: These are objects that may be present in every project. They will be distributed with the platform, although later additional releases could be made, should more objects become necessary. For example, these objects could be queue definitions, AGVs...
- company specific objects: These objects are required in a lot of projects of a specific company, while not being supported within the core Mission objects. For instance, they could be car painting machine templates for a car productor...
- project specific objects: At the end, these objects represent objects which are specific to the related project, or to so few projects that the company doesn't find it useful adding them to the company specific objects set.

Any project is allowed to read and write objects stored in its own tables, and to read objects stored in the Mission and Company tables. Mission tables should not be edited, while Company tables should be edited only by an administrator.

### 3.1.4 Information Manager Management

#### *3.1.4.1 Software Preconditions*

The Information Manager has been programmed in Java. As Java does not directly produce executable files, additional software is required to convert the code contained within these files to specific code for the machine the Information Manager is going to be executed in. This task is performed by a Java Virtual Machine (JVM) that is included with several different software packages. For instance, Windows operating systems include a JVM. Still, the best option is downloading the latest version of the JVM from Sun.

Sun's JVM is included within two packages: the Java Development Kit (SDK) and the Java Runtime Environment (JRE). The former allows the user to develop as well as execute Java programs, while the latter may only be used to execute them. Should none of them be available in the machine executing the Information Manager, one of these packages can be downloaded free of charge. The latest versions are located at http://java.sun.com/j2se and are, at the writing of this document, Java 2 Development Kit v1.3 and Java 2 Runtime Environment v1.3.

Provided that the user has no JVM available, and understanding that, in this case, he is not interested in developing Java Software, the best option is the JRE, because of size. The SDK is about 20 MB in size, while the JRE occupies 5 MB.

Should the user already have a JVM, it must be taken into account that tests have been performed using Java 2 software, although there is no reason why the Information Manager should fail while running on a Java 1.1.x environment. Still, with a Java 1.0.x environment, the Information Manager will not work properly, as there were major API changes from Java 1.0.x to Java 1.1.x.

For the Oracle connection, the JDBC 8.1.5 THIN driver has been included. Oracle states that this driver works with versions from Oracle 7.3.4 to Oracle 8i (it seems that Oracle 8i2 is not supported). Still, the Information Manager has only been tested with Oracle 8i (or Oracle 8.1.5, as it is also known). Oracle also states that JDBC 8.1.5 drivers only work with Java 1.0.x and Java 1.1.x environments, although tests have been performed with Java 2 environments with no apparent problems. Should problems arise during the execution of the Information Manager under a Java 2 environment, Java 1.1.8 versions of the JDK and the JRE may be found at http://java.sun.com/j2se.

Apart from the JVM and the Oracle database, an RTI software must be installed, in order to allow programs to communicate with the Information Manager using the adaptor. An adaptor prototype version has been included with the package. This adaptor versiong requires the RTI-NG v1.3 to be installed. This software can be obtained at http://sdc.dmso.mil.

As for the operating system, the software has been tested only under Windows 2000 platforms. The reason for this is that at the developers' site there were no Windows 98 versions of the Oracle 8i database, but Windows NT ones. Still, there is no reason why it should not work under other operating systems, as well.

#### *3.1.4.2 Configuration Files*

To understand the scheme of the configuration files, the Information Manager structure must be understood, and mainly the Information Manager Interface issue (see chapter 5.5: Data Management Module upgrades). All configuration files share the same structure. They consist of a list of attributes and values that are specific to their related program. Names and values are separated by an equal sign. All lines not including an equal sign are taken as comments and ignored. Lines that are too long can be separated into several lines by adding >> at the beginning of each one of the additional lines.

As a first approach, there is one main configuration file, and another configuration file for each one of the different Information Manager Interfaces that are to be loaded.

##### *3.1.4.2.1 Main Configuration File*

The main configuration file is called, in the default example, InformationManager.conf (see conf directory of the installation). In the example, it only has two fields:

```
FEDERATE CONFIGURATION FILE
FC_FILE=http://localhost/mission/FC.xml

INFORMATION MANAGER INTERFACES DEFINITION
oracle=mission.sisteplant.InformationManager.OracleInterface|
>>d:\mission\jdbc\OracleInterface.conf
```

“FEDERATE CONFIGURATION FILE”, “INFORMATION MANAGER INTERFACES DESCRIPTION” and the blank lines are taken as comments and ignored.

The first field, FC_FILE, points to the Federate Configuration XML file, that has already been discussed (deliverable D9, chapter 8.4: Federate Configuration File).This file contains additional data for the configuration of the Information Manager’s HLA adaptor, and is thus separated from the Information Manager configuration data. If this field does not appear, the FC file is chosen to be http://localhost/mission/FC.xml.

All other fields are considered to be Information Manager Interfaces definitions. These have the following structure:

```
InterfaceName=InterfaceClass|ConfigurationFile
```

In the example, we have an interface named oracle, which is handled by the class mission.sisteplant.InformationManager.OracleInterface. For this class to work properly, there’s the need for a configuration file called d:\mission\jdbc\OracleInterface.conf.

InterfaceName is just a name used to represent the interface without having to use the class name, that is, in most cases, quite longer. This name will be used to address this interface.

InterfaceClass is the name of the Java class used to access a concrete storage medium. The name of the class will be used by the Information Manager to load it, so it must be fully qualified. This means that it’s not enough with the real class name (OracleInterface), but that the package name must also be included (mission.sisteplant.InformationManager).

ConfigurationFile is the name of the corresponding configuration file each Information Manager Interface may have.

The Information Manager analyzes these fields and tries to start all of the interfaces. Should one of the interfaces fail to start, the Information Manager would just ignore it and remove it from the list of valid interfaces. If no interfaces can be run, the Information Manager exits.

#### *3.1.4.2.2 Oracle Interface Configuration File*

The Oracle interface configuration file is called, in the example, OracleInterface.conf, and it contains five fields, providing several parameters for the connection:

```
HOST=localhost
PORT=1521
SID=orcl
LOGIN=Scott
PASSWORD=Tiger
```

These parameters are specific of the Oracle Interface implementation. Each interface has its own ones. The decision on the behaviour of the interface to the lack of these parameters is also left to the corresponding interface developer. For instance, the Oracle Interface has a set of predefined values it uses by default, although it could just give an error message and exit.

### ***3.1.4.3 Oracle Access***

#### *3.1.4.3.1 Overview of Relational Database Access from Java*

Access to relational databases is performed via SQL. SQL consists of a series of commands that are used to retrived data from or insert data into these tables. These commands are very straight-forward, like “SELECT”, “DELETE”, “INSERT”... A command with all its additional parameters is called an SQL statement.

There are two ways to use SQL statements: they can be inserted with specific programs distributed within the database package (like Oracle’s SQL*Plus), or they can be used from external code. Access to the database in Mission is performed using this second approach: SQL statements from Java code.

Access to SQL via Java requires special technologies to be used. These technologies are JDBC (Java Database Connectivity) and SQLJ (SQL Java). JDBC allows statements to be dynamically changed, while SQLJ hard-codes these SQL statements within Java code, making it more intuitive, but less flexible. Besides, SQLJ works by converting the SQL statements it finds within the Java code to JDBC requests. For these two reasons, JDBC has been chosen, in order to allow more direct and flexible access.

#### *3.1.4.3.2 JDBC Access to Oracle*

As we have already said, access to the relational database is accomplished by using JDBC. It consists of a set of Java functions, and a driver which translates JDBC commands and sends them to the database and vice versa.

In JDBC, there are four different driver types defined:

- type 1 drivers: a JDBC-ODBC bridge. ODBC (Open Database Connectivity) is another database access technology used, quite similar to JDBC. This driver connects to the database using the ODBC connection. The driver itself does not require any configuration. But ODBC requires additional software to be installed and configured at each client, which is a drawback to this type of drivers. Therefore, type 1 drivers are seldom used, and even Sun discourages their use (although they provide a type 1 driver).
- type 2 drivers: non-100% Java drivers. They use database-specific calls to access data. As with type 1 drivers, there must be client software installed in the client side (see chapter 4.1: Oracle general information). Note that this software requires about 70 MB of hard disk space. Their major drawback is that this client software uses to be implemented in C, making these drivers very platform-dependent. Still, these drivers tend to be very fast and work with older database versions, as this database specific calls have been used for some time.
- type 3 drivers: one of the two 100% Java drivers. In this case, JDBC requests are translated to a database independent net protocol, and translated again to the database specific protocol by a special server. Its major advantage is that all databases are accessed in the same way from the client’s point of view, as the protocol is independent of the used database. But there’s the need for a middleware server, translating this net protocol to database specific requests to be passed to the final server.

- type 4 drivers: also 100% Java drivers. But, this time, JDBC commands are translated to a database specific net protocol, which the database engine understands directly. There is no need for middleware servers nor for additional client-side software to be installed. The major drawback is that performance is lower than with other drivers.

Both type 1 and type 3 are discarded for their use with Mission. Type 1 is discouraged even by Sun, and type 3 requires a middleware server that wouldn't be needed otherwise. So, the only real choices are type 2 or type 4 drivers. These drivers are database specific, because both use proprietary technology (type 2 uses proprietary software and type 4 a proprietary net protocol). These drivers must be provided by each database vendor. Remote access drivers offered by Oracle are available at their web site http://technet.oracle.com/software/tech/java/sqlj_jdbc/htdocs/listing.htm, and are the following:

- JDBC OCI driver: This one is a type 2 driver. It uses JNI to call specific C functions. These functions call OCI (Oracle Call Interface) libraries, which in turn make the corresponding connection to the Oracle database. This type of driver is implemented for Solaris, Windows and many other machines. It requires Oracle 7.3.4 (or better) client installation.
- JDBC Thin driver: The thin driver is a type 4 driver, completely written in Java and, thus, platform independent. It uses Java sockets to communicate with the database, and does not require additional software to be installed in the client side. However, it has a minor additional drawback in that it requires the underlying network to be TCP/IP (requirement that is not difficult to fulfil, nowadays).

For the Data Management Module implementation, the thin driver has been used. It does not require additional client software and almost every working network uses (or can be configured to use) TCP/IP. Besides, 100% Java code allows easier transition to Internet, which is one of the scopes of the Mission project.

#### *3.1.4.4 Information Manager Commands*

To allow access to the Information Manager features, a series of commands have been added to the HLA adaptor. This commands send different messages to the Information Manager which, in turn, analyzes them and returns an error message or the corresponding result message. This commands may be divided into two different groups: class related and object related commands. There are lots of similarities between these two groups, as they contain (almost) the same commands:

##### *3.1.4.4.1 Class Related Commands*

- Create type: It defines a new class before objects of this type can be created. It defines the inheritances, as parent names are provided at this point.
- Delete type: Completely eliminates a class definition and all classes inheriting from this one. All objects belonging to an eliminated class are also deleted and cannot be further accessed. This call should be used with extreme caution.
- Create copy of type: As it was reported in chapter 4.4.3, types are divided into three categories, of which users can only update one: their own project's. With this method, a copy of a higher class category is copied into the project category. This way, the user can update it without altering the copies on the other categories, to which other users have access.
- Add type attributes: It adds attributes to the type definition. Objects of this type may be added a default value for this attribute if one is provided in the call. Otherwise, they do not contain this attribute until a definite value is provided.
- Remove type attributes: Completely eliminates attributes from the type definition. This attribute cannot be referenced afterwards, and values of attributes of this type are completely lost, for all objects.
- Create relation: Allows new relations of the given type to be created. Parameters such as relation name, source and destination classes and cardinality must be provided.
- Delete relation: Completely deletes a relation and all occurrences of it. Note that deleting a relation (or unrelating two objects, see below) does not eliminate the related objects, but only the existing relationship between them.

#### *3.1.4.4.2 Object Related Commands*

- Create object: Creates a new object instance of the given class. No attributes are defined.
- Change object: Updates the attributes of an object.
- Delete object: Deletes an existing object. This object cannot later be accessed.
- Create copy of object: Creates a copy of a Mission core or Company specific object into the project specific category, so that it can be updated.
- Get object: Allows the user to retrieve an object.
- Get name list: Returns a list of all stored objects, or a list of all stored objects of the given type.
- Relate objects: Defines a new relation between two existing objects. The relation type must be defined with Create relation.
- Unrelate objects: Deletes a relation between two objects. Note that, as with Delete relation, this command does not eliminate the related objects, but only the relationship between them.

#### *3.1.4.4.3 Backup Related Commands*

- Recover type: Recovers the backup of a type definition. Note that this does not alter the existing type (or its inexistence), but only reads backup information. To return to the previous state, use Rollback. If no backup copies exist, an error is returned.
- Rollback type: Returns the definition of the given type to the previous state. Note that this command does not recover backups beyond the first one. Even if a type definition has been updated two or more times, rollback can be used only once.
- Recover object: Returns the backup copy of an updated (or deleted) object.
- Rollback object: As before, returns the given object to a previous state. Again, only one backup copy is stored.
- Commit: Validates the current situation of the database by completely eliminating all backup copies of types and objects. This command should be used with extreme caution, as it will be impossible to return to the previous state.

#### *3.1.4.4.4 Command to Message Relationship*

As it has already been said, the Information Manager receives commands as HLA adaptor messages. The following table shows the existing relationship between commands and messages:

| COMMAND | MESSAGE |
|---|---|
| Create type | CREATE TYPE *NEW_TYPE* AS CHILD OF *PARENT1+PARENT2+...* |
| Delete type | DELETE TYPE *TYPE_TO_DELETE* |
| Add type attribute | ADD ATTRIBUTE *NAME* AS *ATTRIBUTE_TYPE* (WITH DEFAULT VALUE OF *DEFAULT*) TO *TYPE* |
| Remove type attribute | REMOVE ATTRIBUTE *ATTRIBUTE_TO_REMOVE* TO *TYPE_TO_CHANGE* |
| Create copy of type | CREATE COPY OF TYPE *TYPE_NAME* |
| Create relation | CREATE RELATION *NAME* OF *SOURCE* AND *DESTINATION* CARDINALITY ?:? |
| Delete relation | DELETE RELATION *NAME* OF *SOURCE* AND *DESTINATION* |
| Create object | CREATE OBJECT *NAME* AS *TYPE* |
| Change object | CHANGE OBJECT *NAME*>ATT_1=VALUE_1+ ATT_2=VALUE2+... |
| Delete object | DELETE OBJECT *NAME* |
| Get object | GET OBJECT *NAME* |
| Get object list (type) | GET NAME LIST (*TYPE)* |
| Create copy of object | CREATE COPY OF OBJECT *OBJECT_NAME* |
| Relate objects | RELATE *SOURCE* AND *DESTINATION* AS *RELATION* |
| Unrelate objects | UNRELATE *SOURCE* AND *DESTINATION* AS *RELATION* |
| Recover object | RECOVER OBJECT *NAME* |
| Rollback object | ROLLBACK OBJECT *NAME* |
| Delete backups | DELETE BACKUPS |

**Table 3.1: HLA Adaptor Commands and Messages**

Words in italic will be changed as needed, and filled with the correct attribute names, types and so on. The words between brackets are optional.

These messages should never be directly sent to the Information Manager, as there will be adaptor commands that take this responsibility. Still, these commands are blocking, they wait for the Information Manager to answer. So, if other behaviour was necessary, these commands could be sent as messages with “Information manager” as a destination and “IM Command” as type. The answer is of “IM Response” type.

#### *3.1.4.5 Data Management Module Upgrades*

The structure of the Data Management Module can be described as in Figure 3.1:

The Information Manager in itself is not subject to additional changes, should connection to another database be implemented. On running the Information Manager, it gets data on implemented databases and interfaces from the corresponding configuration file (see chapter 5.3: Configuration files) and tries to run the interfaces. Each interface then connects to its associated database (or storage medium, in general), without the Information Manager knowing how to achieve this connection.

All the Information Manager has to know is that every interface has, at least, three methods:

- a basic constructor. A function that initializes all variables used by the interface.
- an initializing function. This function receives the name of the configuration file and starts the connection to the storage medium.
- a processRequest function. This function receives the messages that have been sent to the Information Manager (see chapter 5.5: Information Manager commands), processes them and returns the results of the requests.

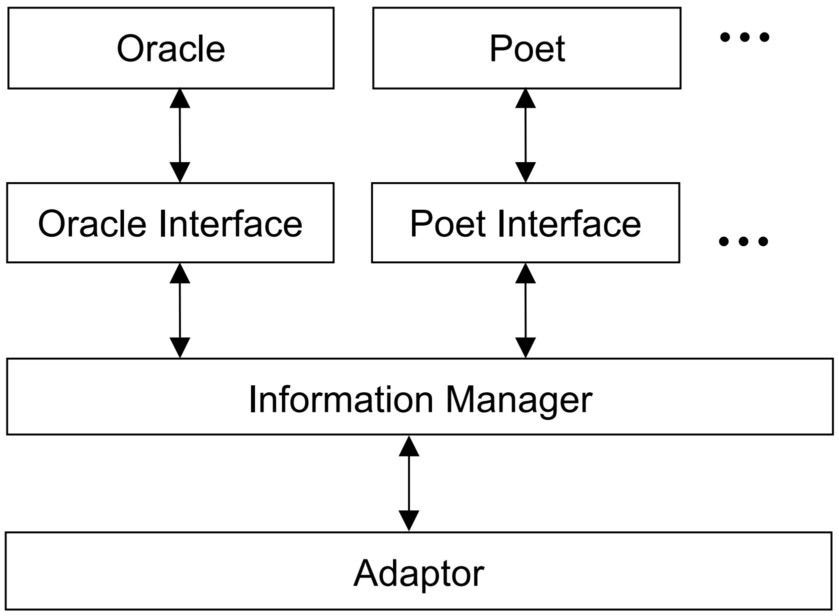


**Figure 3.1: The MMP Data Management Module**

## 3.2 Project Agent

The Project Agent is the first tool to be used in the Mission Platform, because it allows the user to insert the needed data about a project to start it. This program is based on the Information Manager, so all the information used is stored in the Mission database and it is absolutely necessary to have the Information Manager running before start the Project Agent.

This agent is conformed by some screens in order to catch the data about the project specification, customer details, designer of the project, definition of the product and mixes of products, etc. This program uses the Information Manager Adapter to interact with the Information Manager and all the information is stored in the database in the way shown at the deliverable D6 – Definition of the MMP Data Model.

The relation of the Project Agent with the 'Template Library' is shown in figure 3.2 below:
The structure and functionality of the program is defined below. It is primarily a user interface, so the clearest way of understanding it, is to examine the sequence of windows forming the application. These are presented in Figures 3.3 to 3.10 below.

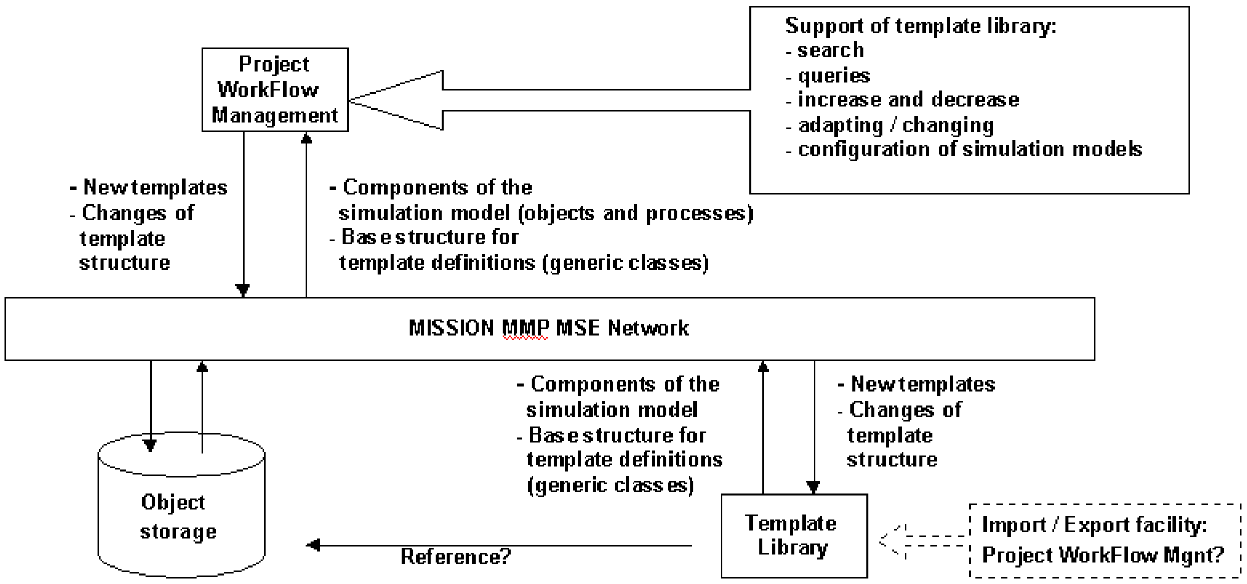
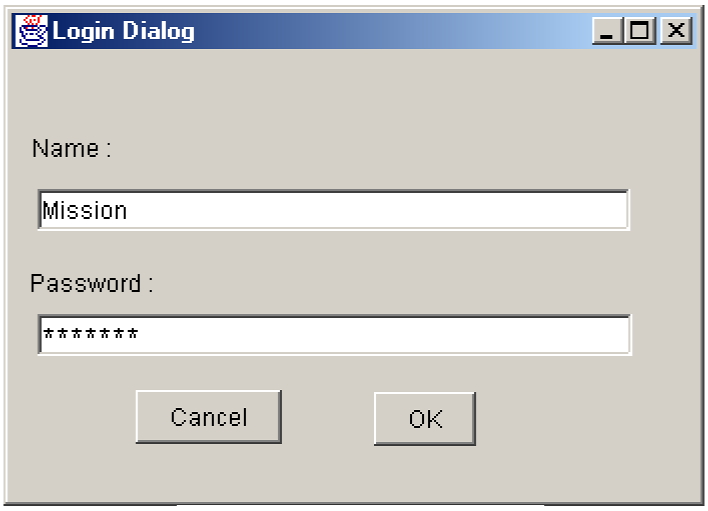

Project WorkFlow Management
Support of template library:
- search
- queries
- increase and decrease
- adapting / changing
- configuration of simulation models
- New templates
- Changes of template structure
- Components of the simulation model (objects and processes)
- Base structure for template definitions (generic classes)
MISSION MMP MSE Network
- Components of the simulation model
- Base structure for template definitions (generic classes)
- New templates
- Changes of template structure
Object storage
Reference?
Template Library
Import / Export facility: Project WorkFlow Mgnt?

**Figure 3.2: Project Workflow Management**



Name : Mission

Password : *******

**Figure 3.3: Login Dialog**

This is the first window the user is going to interact with and it allows this one to enter the application. There must exist a user with a password to get into the Project Agent.

If the user does not fill in the whole two fields, a new window is shown with a warning that allows the user to fill in all the fields.

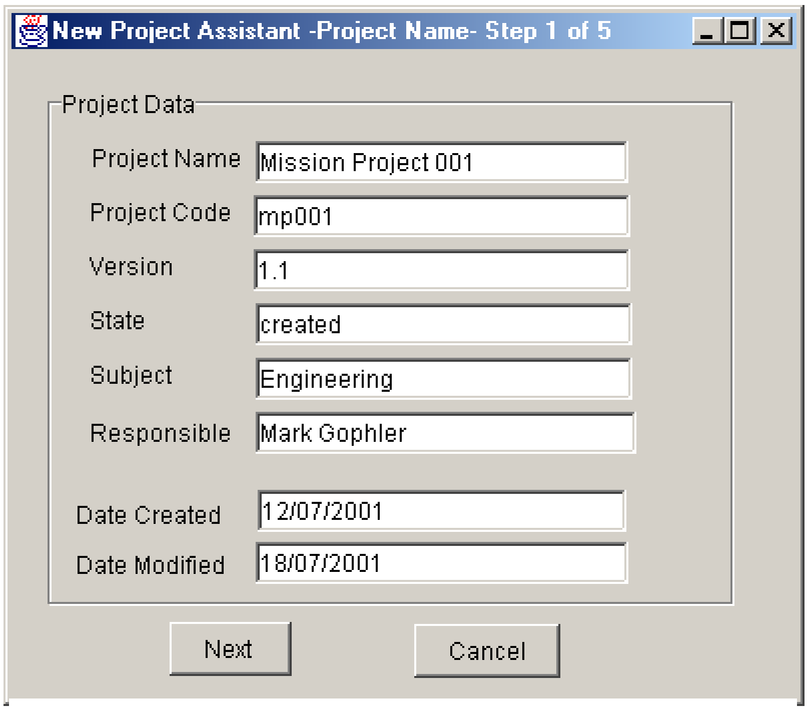
The user must click the ‘OK’ button to accept the written information or ‘Cancel’ button if he wants to exit the application.

Once the user has entered the application, he has to choose between creating a new project or opening an existing one.

First of all, the user has to fill in the fields about the Project data, like project name, project code, state, responsible, etc.

The user must write all the needed information about the project to go on, otherwise a warning window would appear.

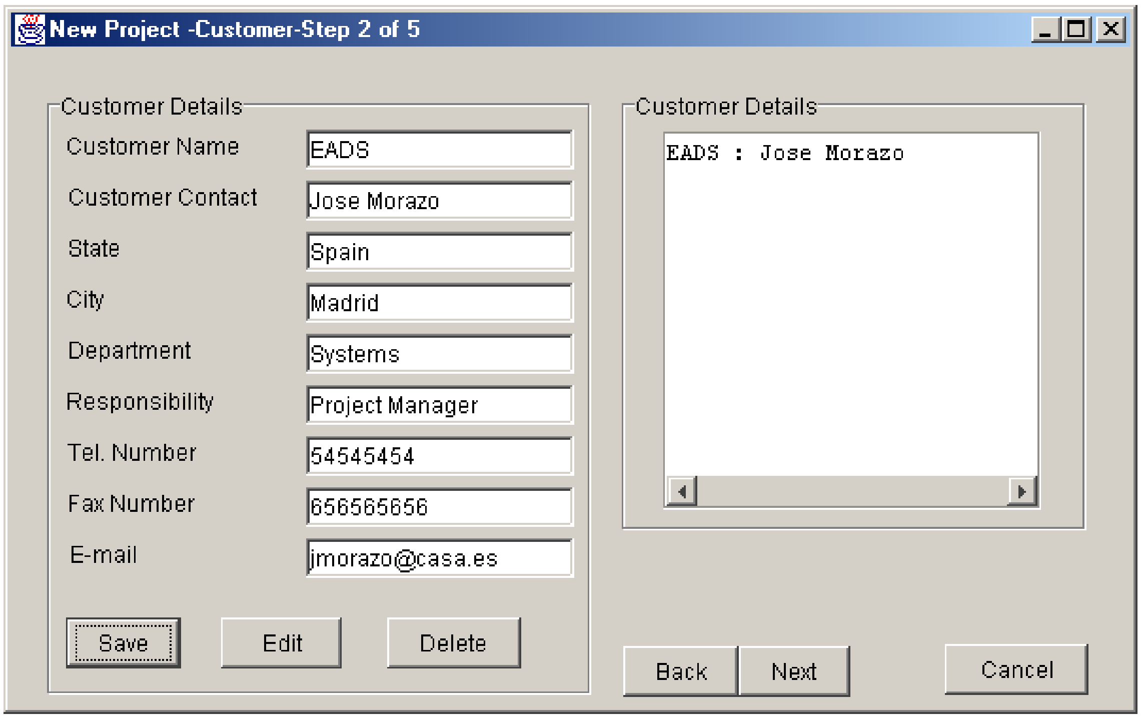
Once the user has filled in the data, he must go on clicking on the ‘Next’ button, or ‘Cancel’ button to exit the program.



Project Data

| | |
|---|---|
| Project Name | Mission Project 001 |
| Project Code | mp001 |
| Version | 1.1 |
| State | created |
| Subject | Engineering |
| Responsible | Mark Gophler |
| Date Created | 12/07/2001 |
| Date Modified | 18/07/2001 |

**Figure 3.4: New Project – Project Data**



|  |  |
|---|---|
| Customer Name | EADS |
| Customer Contact | Jose Morazo |
| State | Spain |
| City | Madrid |
| Department | Systems |
| Responsibility | Project Manager |
| Tel. Number | 54545454 |
| Fax Number | 656565656 |
| E-mail | jmorazo@casa.es |

Customer Details

```
EADS : Jose Morazo
```

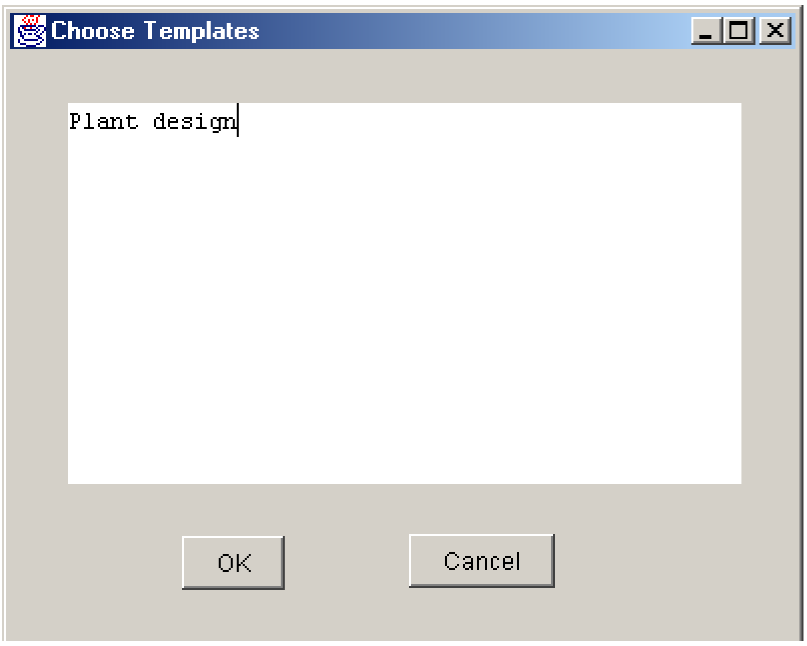
**Figure 3.5: New Project – Customer Data**

Next step is writing the customer details, like customer company, customer contact, etc.

This data can be changed (‘Edit’ option) or deleted (‘Delete’ option), but once the user clicks the ‘Save’ button, the customer organisation name and the customer contact person should appear on the right side of the screen and this should mean that the data has been stored in the database.

The user can also go back to the project data screen (‘Back’ option), go on to the next screen (‘Next’ option) or cancel the process (‘Cancel’ option).

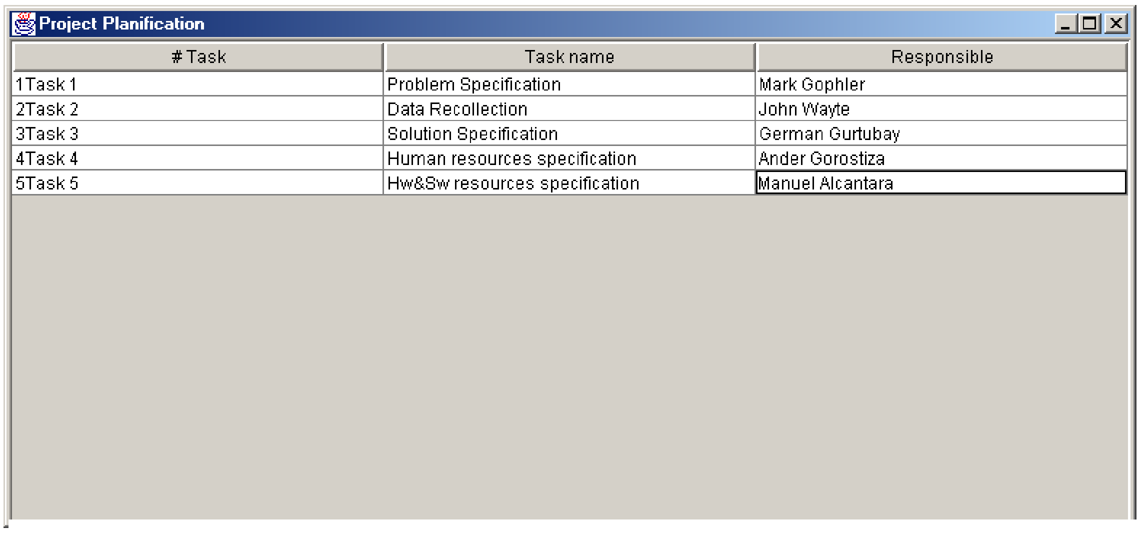
In the same way, the user must fill in the designer information about the project.



Plant design

Figure 3.6: New Project – Choose templates

In this screen, the template for the project is included, but by default the Plant Design is always shown. If the user wants to choose a template, he has to write down its name and this information will be stored.



| # Task | Task name | Responsible |
|---|---|---|
| 1Task 1 | Problem Specification | Mark Gophler |
| 2Task 2 | Data Recollection | John Wayte |
| 3Task 3 | Solution Specification | German Gurtubay |
| 4Task 4 | Human resources specification | Ander Gorostiza |
| 5Task 5 | Hw&Sw resources specification | Manuel Alcantara |

Figure 3.7: New Project – Project Planning

Next step for the administration data of the project is the Planning, that means the user has to write the tasks defined for a project. By default the task name is defined as it is shown in the Figure 3.7, but the user can change the name if needed.

Once the task codes, task names and task responsible are defined, the creation of a new project has been finished.

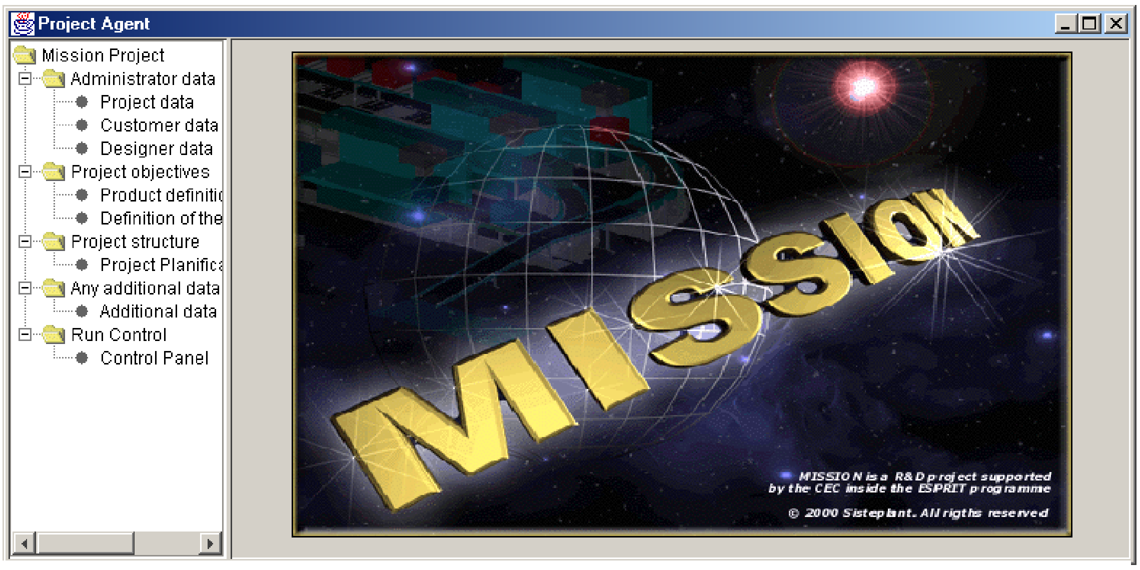


**Figure 3.8: Administration of the Project – Extended Data Tree**

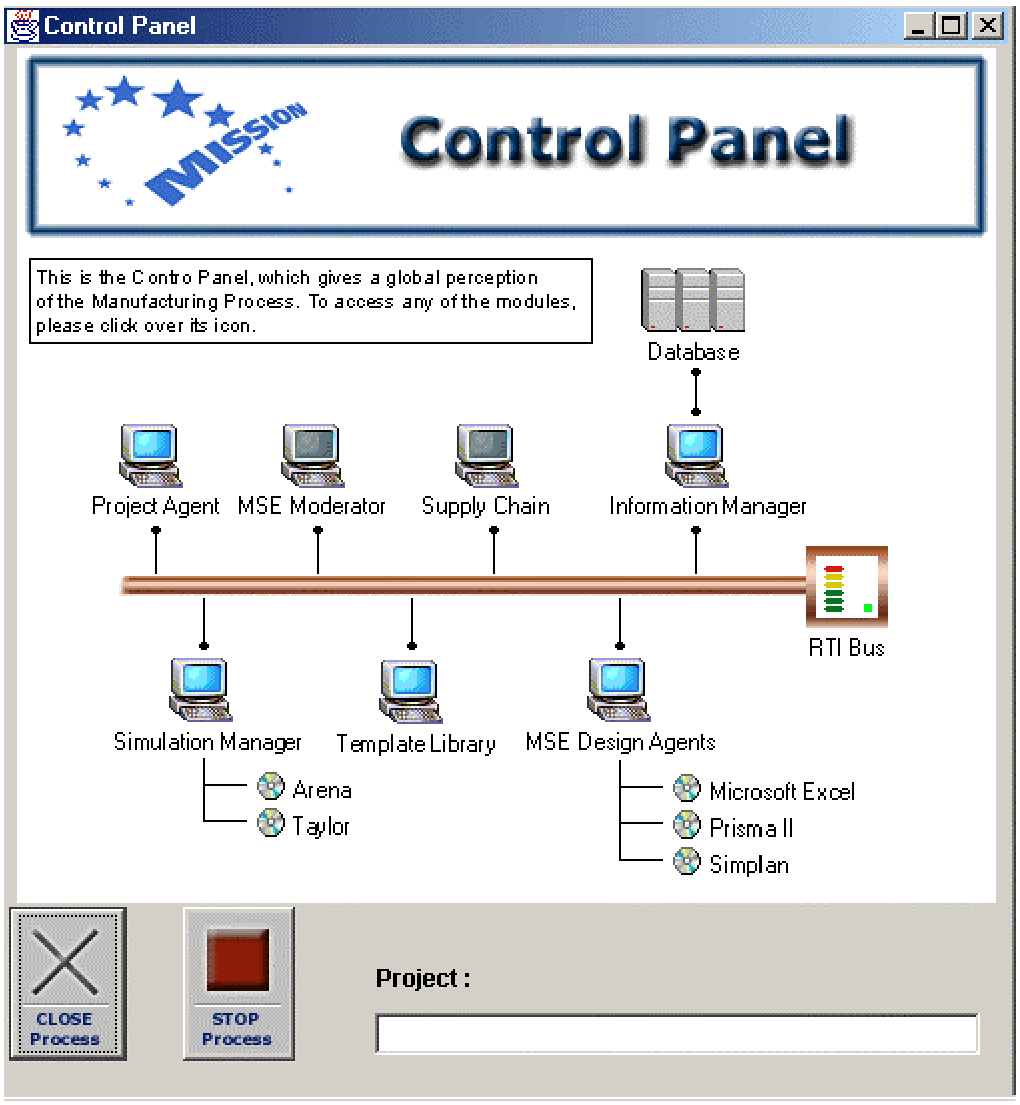
When the user gets into this window, this means that the project administration data has been stored, so the user can go on the Project Agent. This screen consists in a data tree which is used to show the user the project information in a very ordered and useful way.

This is all the information needed about the project to start it. The data are organised by four subjects, each one with more classification items as follows:

- Administration Data: information about the Project, Customer and Designer
- Project Objectives: Information about the Product and Mix of Products
- Project Planning: Information about the phases that define the project
- Additional Data: Any more data to increase the details about the project
- Run Control of the project.

If the user gets into the tree options, views about project data, customer data, designer data, product definition, definition of the mixes, project planning, additional data and control panel should appear.

The most important option in the ‘Extended Data Tree’ is the ‘Run Control’ – ‘Control Panel’. This option is just a confirmation window before the project is going to be run. The project name and project code is shown and there is a button to confirm that the project is prepared to be run.



This is the Contro Panel, which gives a global perception of the Manufacturing Process. To access any of the modules, please click over its icon.

**Figure 3.9: Control Panel**

If the user wants to control the simulation the ‘Control Panel’ process can be used. This screen visualises a project process view, where the user can see the communication among all the agents related to the Mission Platform. The process can be stopped or closed just clicking the corresponding buttons.

As it was said at the beginning, there are two options when a user enters the Project Agent application, one is to create a new project, that has been explained before and the other one is to open an existing project.

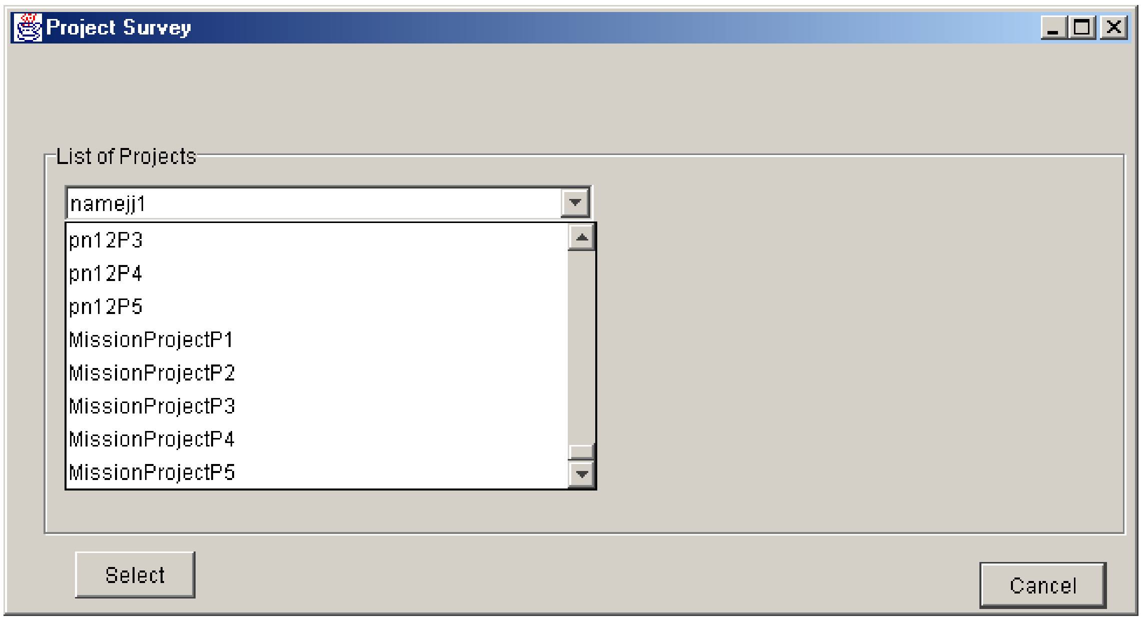


**Figure 3.10: Open existing project and Search projects**

If the user wants to open an existing project, it is only necessary to select the project. In this case, the project would be opened and the Administration tree panel with all the information about the project (the same information seen in Figure 3.8) is shown.

## 3.3 Manufacturing Process Design Agent & Supply Chain Agent

### 3.3.1 Introduction

This chapter describes the Functionality of Manufacturing Process Design Agent (MPDA) and Supply Chain Agent (SCA). These two components are described with each other, because they are very similar.

Like the Template Library Editor (MSE) the MPDA & SCA both use the IM-Adapter.

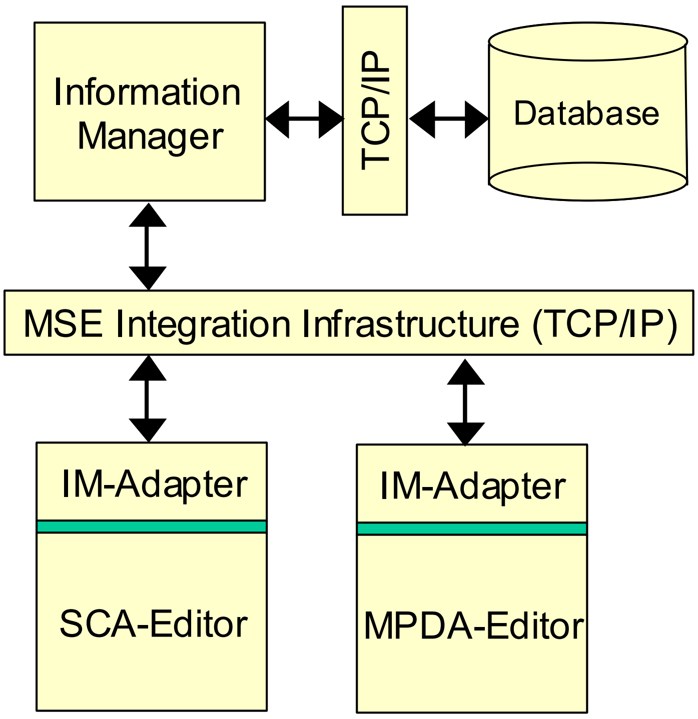


**Figure 3.11 Communication between SCA / MPDA and Information Manager**

### 3.3.2 Requirements

#### *3.3.2.1 Hardware*

The minimum Hardware requirement for a practicable usage of the MPDA / SCA is a AMD or Pentium System with at least 233 MHz and 64 MB RAM.

If you wish to run on the same computer the Information Manager and the database (Oracle) than the user should have a computer with 500 MHz and 256 MB RAM.

#### *3.3.2.2 Software*

The Information Manager Version 1.1 is required for running the MPDA / SCA. As for the IM it is needed to have a Java Runtime Environment 1.2 or greater installed. If you do not have a JRE than you can downloaded one without charge from Sun at http://java.sun.com/j2se/1.3/ or from IBM at http://www-106.ibm.com/developerworks/java/jdk/?dwzone=java for Windows 9X, NT, 2000 and several *nix Versions

### 3.3.3 Installation

The distribution consists of one archive (mpda.zip) for the MPDA which contains the following files:

| Filename | Description |
|---|---|
| mpda.jar | Archive containing the MPDA program files. |
| mpda.bat | Batch file for starting the MPDA. |
| mpda-config.xml | A XML configuration file containing some necessary information for the execution. |
| mpda -config.dtd | The document type definition for the config file. |

**Table 3.2: Content of the mpda.zip file**

And one zip-File for the SCA:

| Filename | Description |
|---|---|
| sca.jar | Archive containing the SCA program files. |
| sca.bat | Batch file for starting the SCA. |
| sca-config.xml | A XML configuration file containing some necessary information for the execution. |
| sca -config.dtd | The document type definition for the configuration file. |

**Table 3.3 Content of the sca.zip file**

For the installation it is only necessary to unzip these files into any directory.

### 3.3.4 Configuring the Program

The configuration file for the MPDA / SCA is a simple XML file (sca-config.xml, mpda-config.xml) which has to be edited by the user before he can run the program. The configuration file has the following format:

```
<Information_Manager>
      <Address>
            [address]
      </Address>
      <Port>
            [port]
      </Port>
</Information_Manager>
```

where

- [address] is the TCP/IP address or a name which can be resolved by a name server where the IM is running at this time and
- [port] is the port where the IM is running at this time.

A valid configuration file is for example:

```
<Information_Manager>
      <Address>
            um183.ipk.fhg.de
      </Address>
      <Port>
            5500
      </Port>
</Information_Manager>
```

The address and port have to be set to point to the computer where the IM is running.

### 3.3.5 Starting the Programs

The IM must be running on the computer defined in the IM section of the configuration file (see 3.3.4), before the MPDA / SCA can be started. The program is invoked by double clicking on the mpda.bat / sca.bat file. It creates a connection to the IM. If this fails you will get a message and the program shuts down.

### 3.3.6 MPDA and SCA User Interfaces

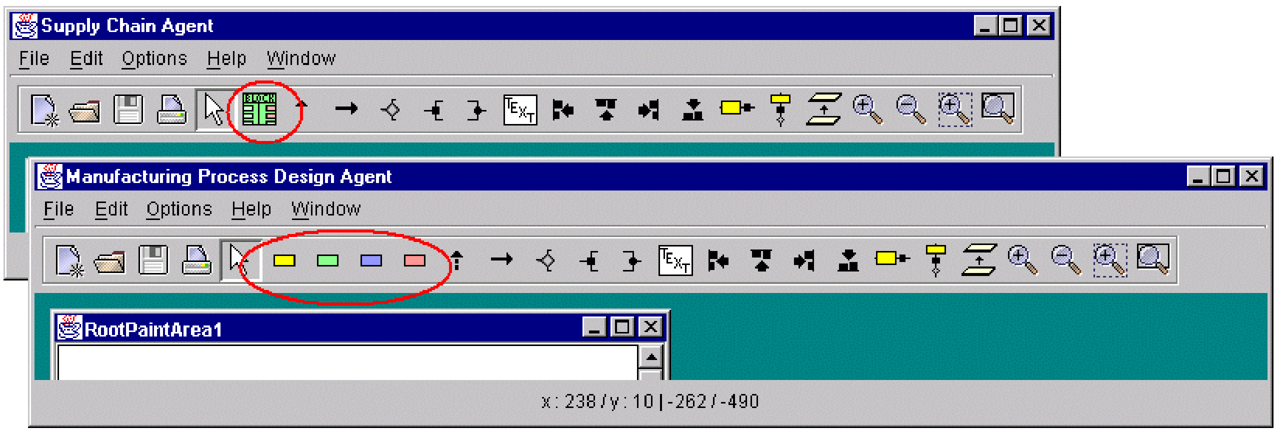

Supply Chain Agent
File Edit Options Help Window
Manufacturing Process Design Agent
File Edit Options Help Window
RootPaintArea1
x : 238 / y : 10 | -262 / -490

**Figure 3.12: MPDA and SCA appearance**

As shown in Figure 3.12 above the two tools are from the look and feel very similar.

### 3.3.7 SCA Features

The task of the SCA is to provide a possibility to design a global supply chain, which will meet the objectives of the MSE project.

- Select the plants to contribute to the manufacture of the product, based on their capabilities to manufacture in appropriate volumes at appropriate times to meet global objectives.
- Design transport networks between plants capable of economically managing the flow of materials in a manner adequate to meet global objectives.
- Determine inventory policy for a global system, defining philosophy (e.g. JIT, make to stock, etc.), shipment batch quantities, order lead times, etc, which will allow the supply chain to meet global MSE objectives.

To perform this task, the user can graphically create and change supply chain models inside his project. To insert templates (building blocks) into this tool, the user can use drag and drop from the Template Editor.

The input and output segments can be connected with lines.

The SCA has the following features:

- opening models
- saving models during editing
- printing a model
- inserting elements
- editing elements
- connecting elements
- aligning elements
- inserting text comments
- zooming.

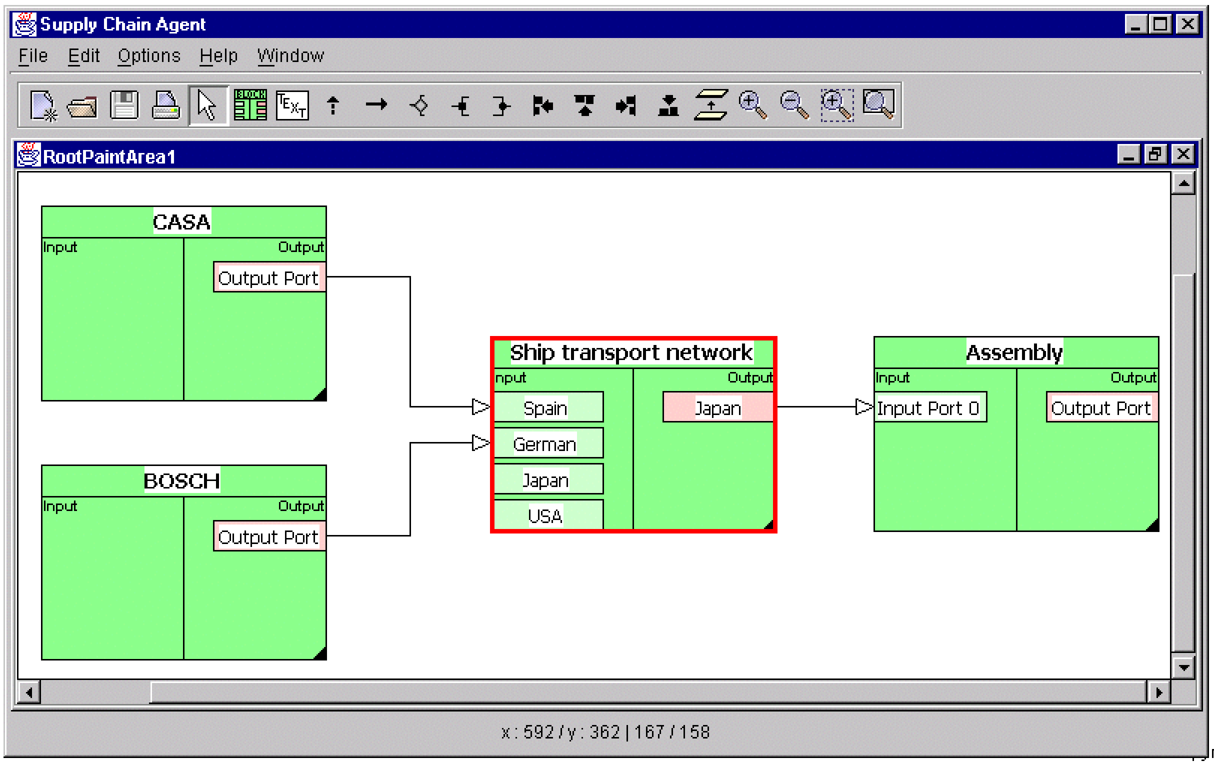


**Figure 3.13: Example for the SCA with an opened model**

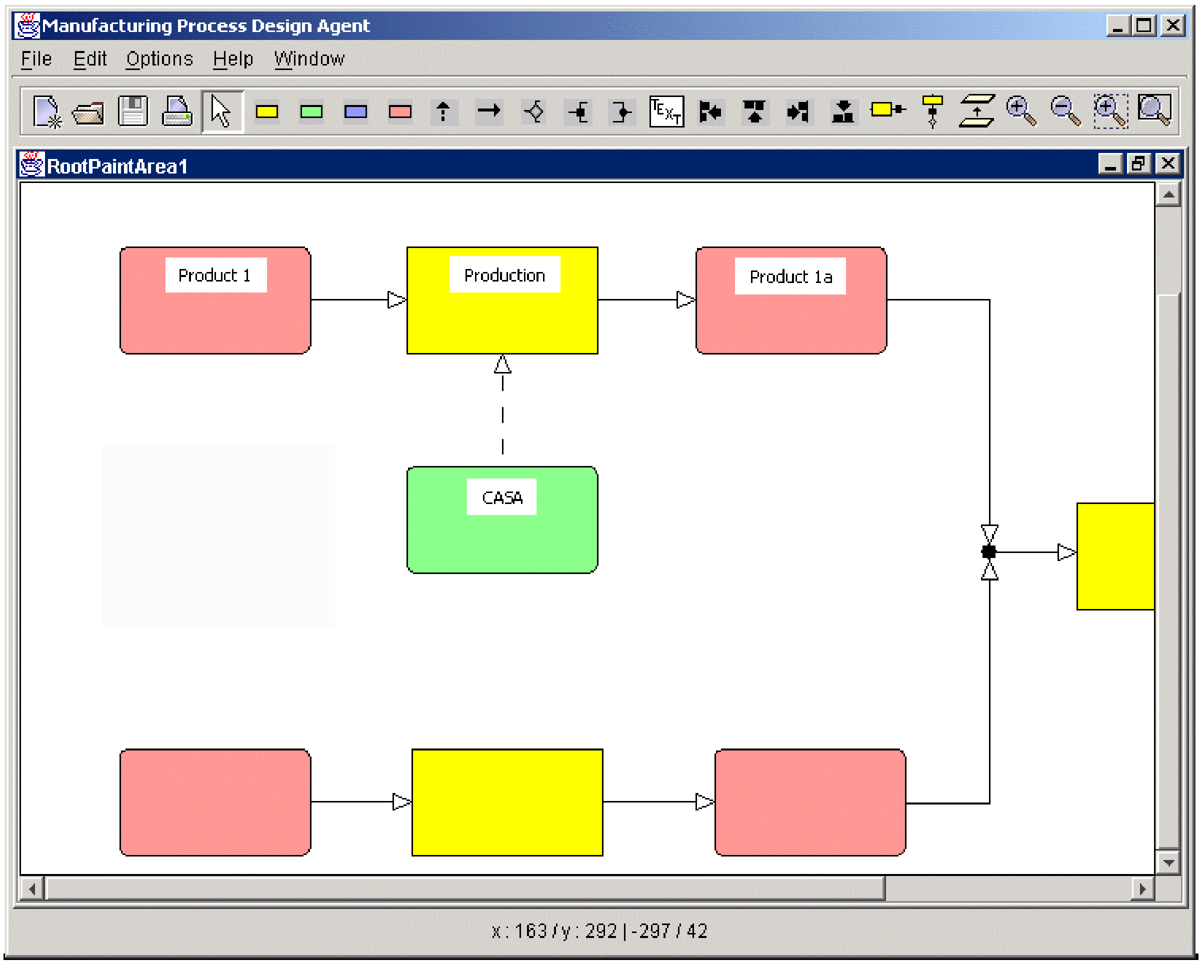

Manufacturing Process Design Agent
File Edit Options Help Window
RootPaintArea1
Product 1
Production
Product 1a
CASA
x : 163 / y : 292 | -297 / 42

**Figure 3.14 MPDA with an example model**

## 3.3.8 MPDA Features

The task of the MPDA is to provide a possibility to model process flows, which will meet the objectives of the MSE project.

To archive this task, the user can graphically create and change manufacturing processes inside his project. MSE items can be dragged from the Template Editor (MSE) into the MPDA.

Each item can be connected with other items.

The MPDA has the following features:

- opening models
- saving models during editing
- printing a model
- inserting elements
- editing elements
- connecting elements
- aligning elements
- inserting text comments
- zooming.

### 3.3.9 Connection to the Information Manager

The MPDA and SCA are connected over the IM-Adapter directly with the Information Manager. This means models storage and loading is done there. Any change the user does in a model is transferred immediately to the IM. At this time editing of the same model at the same date with two clients (two MPDA or two SCA) is not possible because of a fault within the recent IM-Adapter.

### 3.3.10 Interaction with the Template Editor (MSE)

Another possibility to create elements is, to drag them from the Template Editor into the MPDA or SCA. After doing this they will appear in the target window and can be used there. Please, take a look at the description of the Template Editor for more details.

## 3.4 Manufacturing System Engineering Moderator

### 3.4.1 Rationale for a Manufacturing Systems Engineering Moderator

The process of designing a manufacturing system (MS) requires the application of different areas of expertise, including, for example, functions such as process selection, equipment selection, facility layout, performance prediction (perhaps by simulation), and potentially many others. Additionally, information on the strategic requirements (eg. what product(s) are to be made in what volumes, to satisfy what market constraints) from the MS must be available, since this effectively forms the design specification for the MS.

In a very small project it may be that a single engineer or a small co-located group of engineers will be entirely responsible for all aspects of the design process. However in a larger project, and particularly in a project, as envisioned in the MISSION research, where the MS will span several sites which are possibly globally distributed, it is unlikely that a single engineer will carry out all the necessary design functions. Projects of this scale will generally rely on inputs from a team of specialist engineers with various different types of expertise (Figure 3.15). It is likely that the team of project engineers will also be globally distributed.

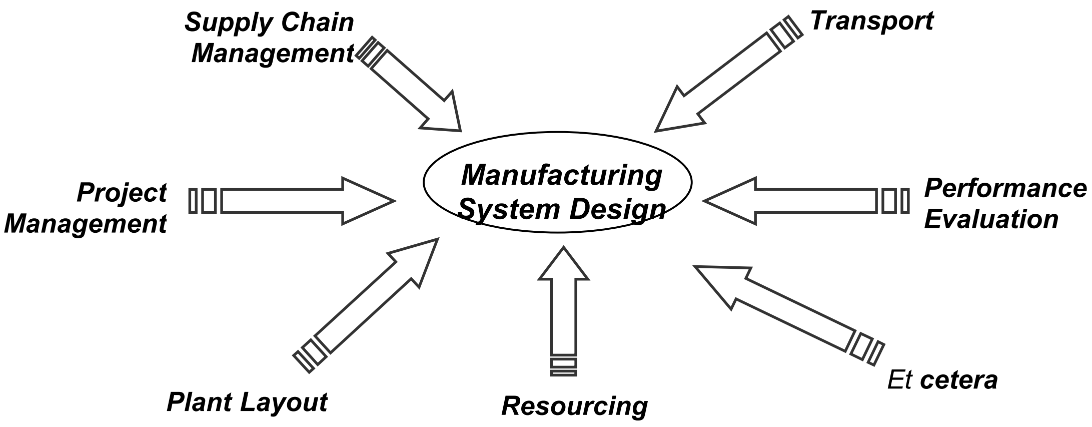


**Figure 3.15. Functions contributing to Manufacturing System Design**

Typically, each MSE design function will be supported by appropriate software tools, and the tools favoured for particular design functions, or by individual engineers, will often be different from those used in other functions or on other sites. In the MISSION project we use the term “MSE Agent” to refer to each combination of engineer(s) and supporting software performing an identifiable function to contribute to the developing MSE design. The developing design is shared between agents, and is held in a common MSE database, structured by a MISSION information model to meet agent needs (see Figure 3.16). Thus, for example, the project management function is fulfilled by a Project Agent which may be expected to include software tools to support both strategic management and project planning, whilst simulation software and statistics packages would be available within a Simulation Agent for performance evaluation. It is important to note that the balance of human and computer intelligence in an MSE agent will vary between MSE agents in a particular MISSION installation, and that the balance may even change within an installation or even during the course of an MSE design project, as new skills or software support are introduced. Few MSE agents will be entirely software based, and no agent can be entirely human as there must at least be an interface with the MSE database. Perhaps the only purely software agent would be the Information Manager which handles all transactions between the MSE database and MSE agents.

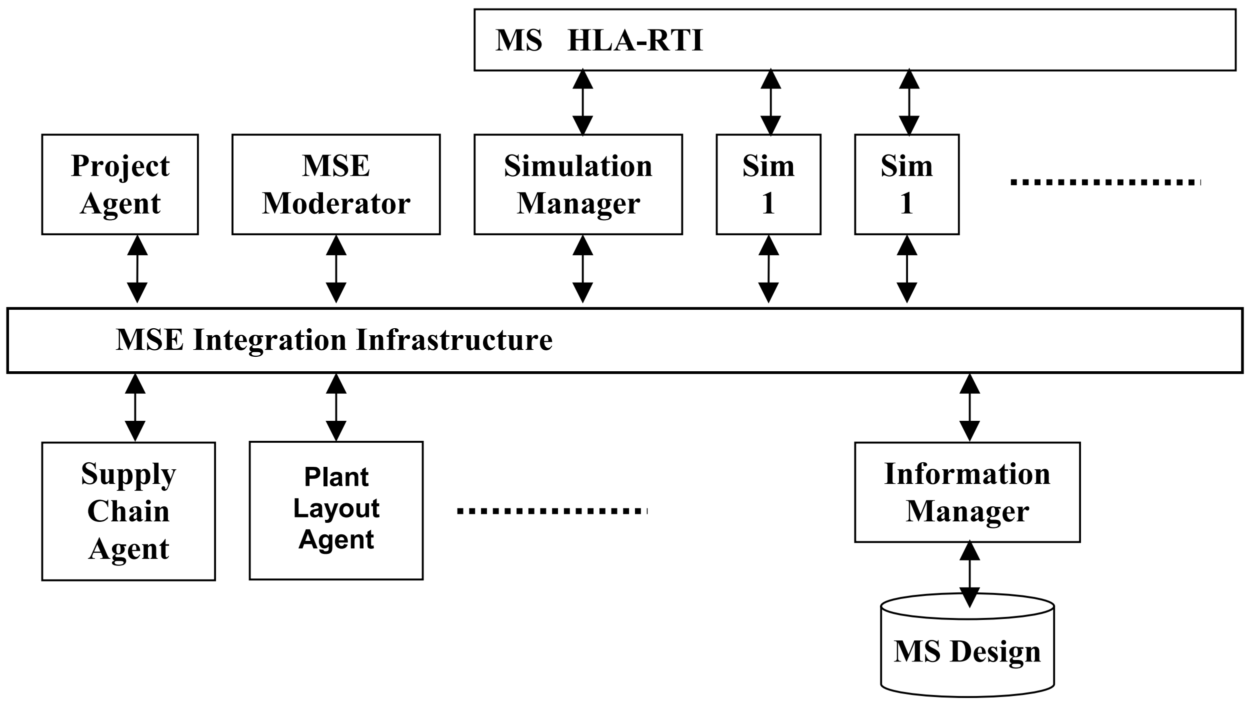


**Figure 3.16. The MISSION Modelling Platform Architecture (Summary)**

Inescapably there are conflicts between the objectives of the different functions, and this may lead to conflicting decisions, that is decisions which are beneficial to the objectives of one function, but detrimental to the objectives of other functions. If such conflicting decisions remain embedded in the final, implemented design the MS may fail to perform as specified.

In a small co-located project team it is conceivable that an experienced team leader will monitor the developing design as the functions contribute to it, and identify the points of conflict as they arise. Thus he can ensure that conflicts are resolved, as soon as possible, so as to best meet the strategic objectives of the project. However as teams become larger, to complete larger projects in shorter times, it is more and more probable that the leader will be unable to identify conflicts as they arise. This is particularly true in the MISSION scenario where the use of distributed teams becomes more common, as global virtual enterprises endeavour to utilise a globally distributed skills-base without incurring the costs of physical re-location of personnel. Intelligent, software support tools can be provided to assist the complex and demanding responsibilities of the project leader, and one such tool is the Manufacturing Systems Engineering Moderator.

The purposes of the Manufacturing Systems Engineering Moderator are to continuously monitor the information model of the developing design, to identify as far as possible each occurrence of a design conflict, and to orchestrate a dialogue between the interested design functions until the conflict is resolved.

## 3.4.2 Requirements

### *3.4.2.1 Moderation Process*

In order to identify and signal design conflict in this environment, the MSEM has the following requirements:

- The MSEM must know whenever a change is made to the MS design. In the context of the MMP, the Information Manager signals the MSEM whenever a change in the MSE design is recorded in the project database. The signal includes information to identify the elements of the design added or changed.
- The MSEM must be able to identify when a design change may cause conflict. The MSEM will therefore retain and apply knowledge about the knowledge used by each of the MSE agents. In doing so it is likely that the MSEM will request through the Information Manager information about design features related to the change, as shown in Figure 3.17.

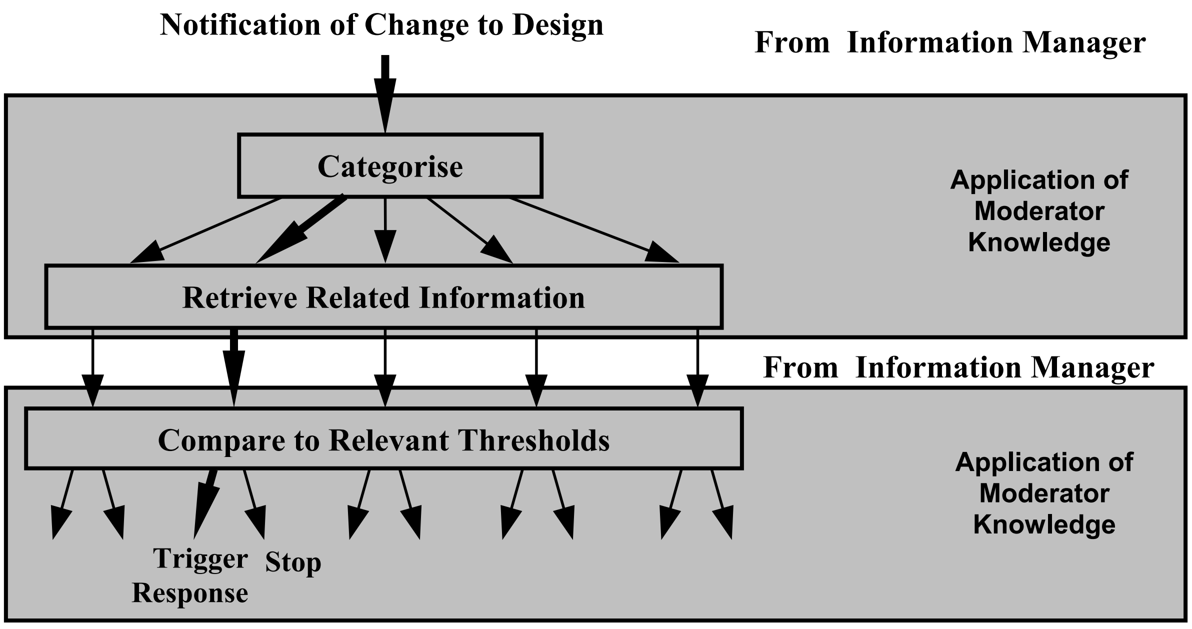

Notification of Change to Design
From Information Manager
Categorise
Application of Moderator Knowledge
Retrieve Related Information
From Information Manager
Compare to Relevant Thresholds
Application of Moderator Knowledge
Trigger Response
Stop

**Figure 3.17: Recognising Potential Conflict**

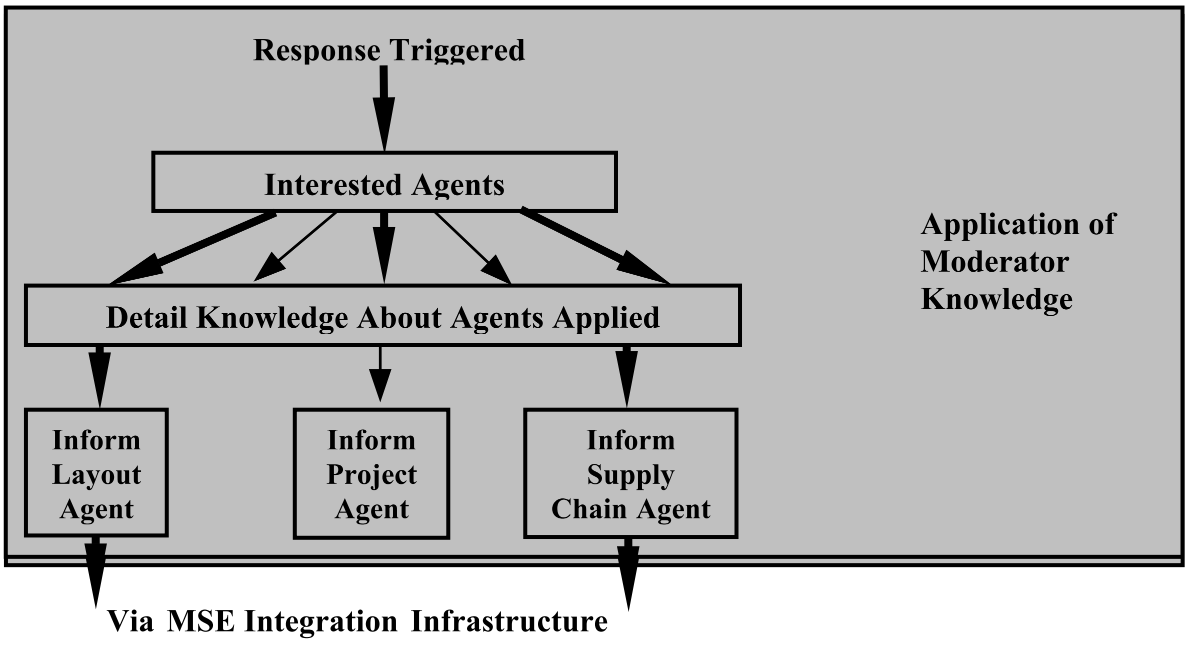

Response Triggered
Interested Agents
Detail Knowledge About Agents Applied
Inform Layout Agent
Inform Project Agent
Inform Supply Chain Agent
Application of Moderator Knowledge
Via MSE Integration Infrastructure

**Figure 3.18: Notification of Interested MSE Agents**

- The MSEM must communicate the detection of possible conflict to all MSE agents which it deems to have an interest in resolving the conflict (Figure 3.18), and when necessary remain in dialogue with these agents until resolution is achieved.

### *3.4.2.2 Knowledge of MSE Agents*

The MSEM is populated with knowledge about the knowledge used by the MSE agents available to the current project.

- The MSEM will know of all MSE agents available to the current project.
- The MSEM will know which MSDD objects are of interest to each MSE agent.
- The MSEM knows how to update its knowledge about MSE agents, either updating knowledge of an existing agent, deleting an existing agent, or introducing a new agent.
- The MSEM will know how to identify changes (additions, amendments or deletions) to such objects which may be of critical to the interests of each MSE agent.
- The MSEM will know how to communicate conflict warnings to each MSE agent.

### *3.4.2.3 Knowledge Acquisition*

The nature of the MSEM is such that its knowledge content must be directly governed by the range of MSE agents incorporated in the MSE team and connected to the MSE Infrastructure. It is thus not possible to develop a hard-coded representation of the knowledge needed for general implementation. The knowledge base will differ between one implementation and another, and indeed it is likely that only the highest level of the structure of the MSEM will be constant between implementations. It is also probable that the knowledge base will need to change as a project progresses, since agents will join the project, requiring additional knowledge to be made available to the MSEM.

- The core knowledge common to all MSEM installations is limited to the structure of general moderation processing, and knowing how to acquire installation specific knowledge.
- Each MSEM installation will be able to acquire knowledge which is generally applicable to the industry and enterprise, including the knowledge of agents available to all MSE projects likely to be undertaken using the installation.

- An MSEM functioning for a particular MSE project will acquire knowledge of the particular MSE agents available to and participating in the project, and be able to update that knowledge as the project progresses.

#### *3.4.2.4 MSE Infrastructure Communications*

- The MSEM will be able to receive messages from the MMP Information Manager informing the MSEM that a change has been recorded in the MSDD. The message will include identification of the changed objects in the MSE Design Database.
- The MSEM will be able to request and receive from the Information Manager, current and previous versions of objects which have changed or related to objects which have changed.
- The MSEM will be able to send messages to any MSE agents involved in the MSE process to inform them of design changes originated by other agents which may impact on their individual areas expertise.

StrTo satisfy the requirements of the Moderation Process, Knowledge of MSE Agents and Knowledge Aquisition defined in section 2, the MSEM has been structured into 3 main modules (see figure 3.19). The knowledge used by the MSEM is all stored within object oriented databases. This approach simplifies maintenance and reuse of the knowledge, and provides great potential for information and knowledge sharing.

### 3.4.3 Structure

#### *3.4.3.1 Knowledge Structure*

To satisfy the requirements of the Moderation Process, Knowledge of MSE Agents and Knowledge Acquisition defined in section 2, the MSEM has been structured into 3 main modules (see figure 3.19). The knowledge used by the MSEM is all stored within object oriented databases. This approach simplifies maintenance and reuse of the knowledge, and provides great potential for information and knowledge sharing.

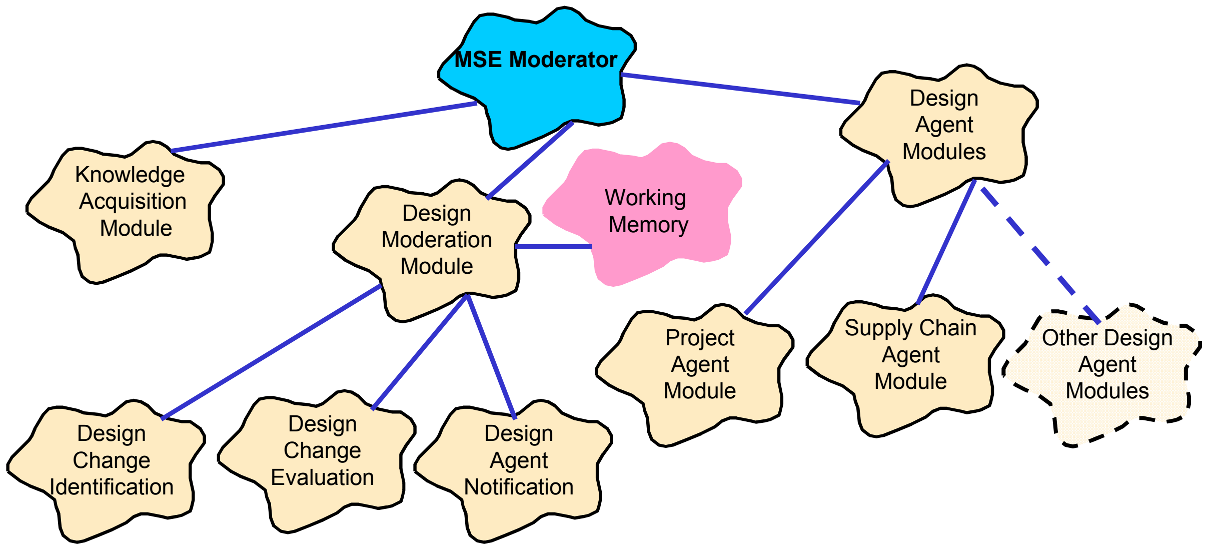

MSE Moderator
Knowledge Acquisition Module
Design Moderation Module
Working Memory
Design Agent Modules
Design Change Identification
Design Change Evaluation
Design Agent Notification
Project Agent Module
Supply Chain Agent Module
Other Design Agent Modules

**Figure 3.19: The 3 Main Modules of the MSEM**

### *3.4.3.1.1 Knowledge Acquisition Module:*

In the prototype implementation of the MSEM, this has been implemented as a program, which may be used to create, delete or modify Design Agent Module objects, and their associated knowledge objects. The resulting objects are stored, as persistent objects in an object oriented knowledge database. Figure 3.20 shows the interface from the prototype MSEM Knowledge Acquisition module, which can be used whenever new Agents join a project, or existing Agents are changed in any way.

### *3.4.3.1.2 Design Moderation Module*

In the prototype implementation of the MSEM, this has been implemented as a program, which orchestrates the interactions of several key objects, which are stored as persistent objects within the object oriented knowledge database. When the Design Moderation Module is activated, it is connected to a knowledge database, which contains the Design Agent Modules, and the MSEM's Working Memory Object. The Working Memory Object is used by MSEM to keep track of changes made to the MSE design, and to record and manage its interactions with Design Agent Modules. Design Change Identification is achieved by the MSEM polling and asking the Information Manager for details of any changes that have taken place. Design

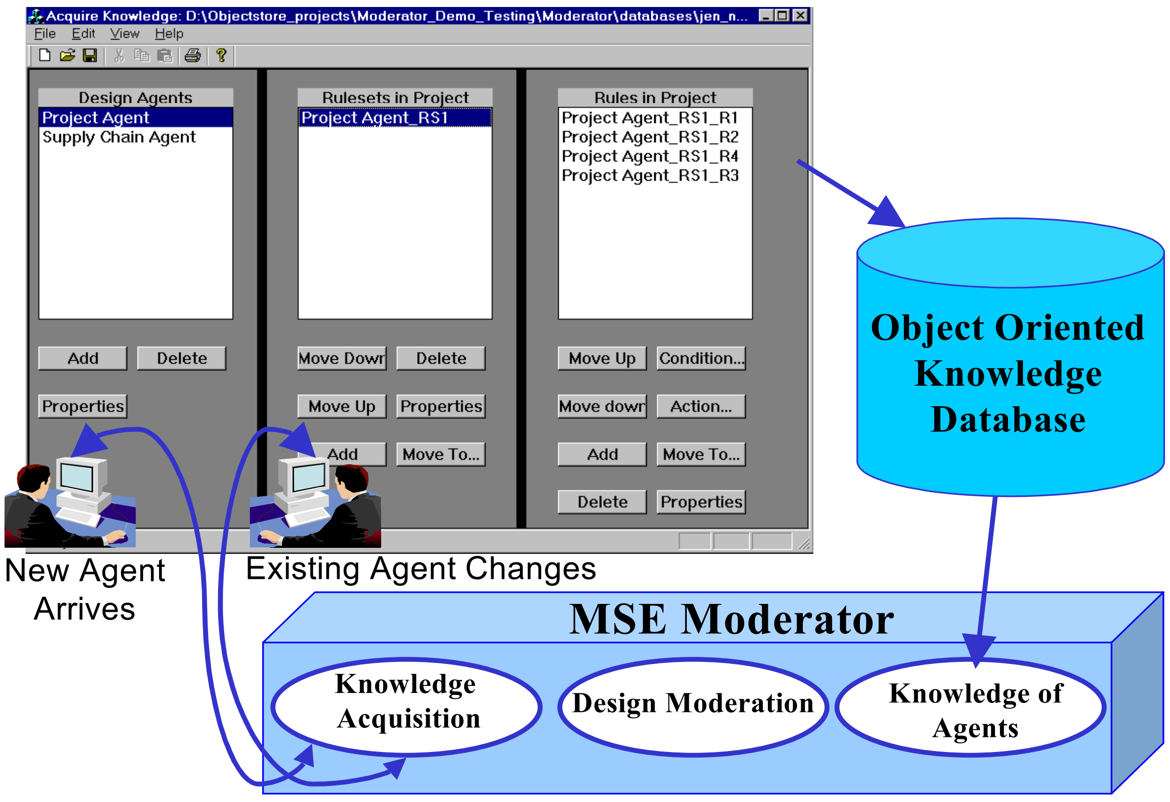


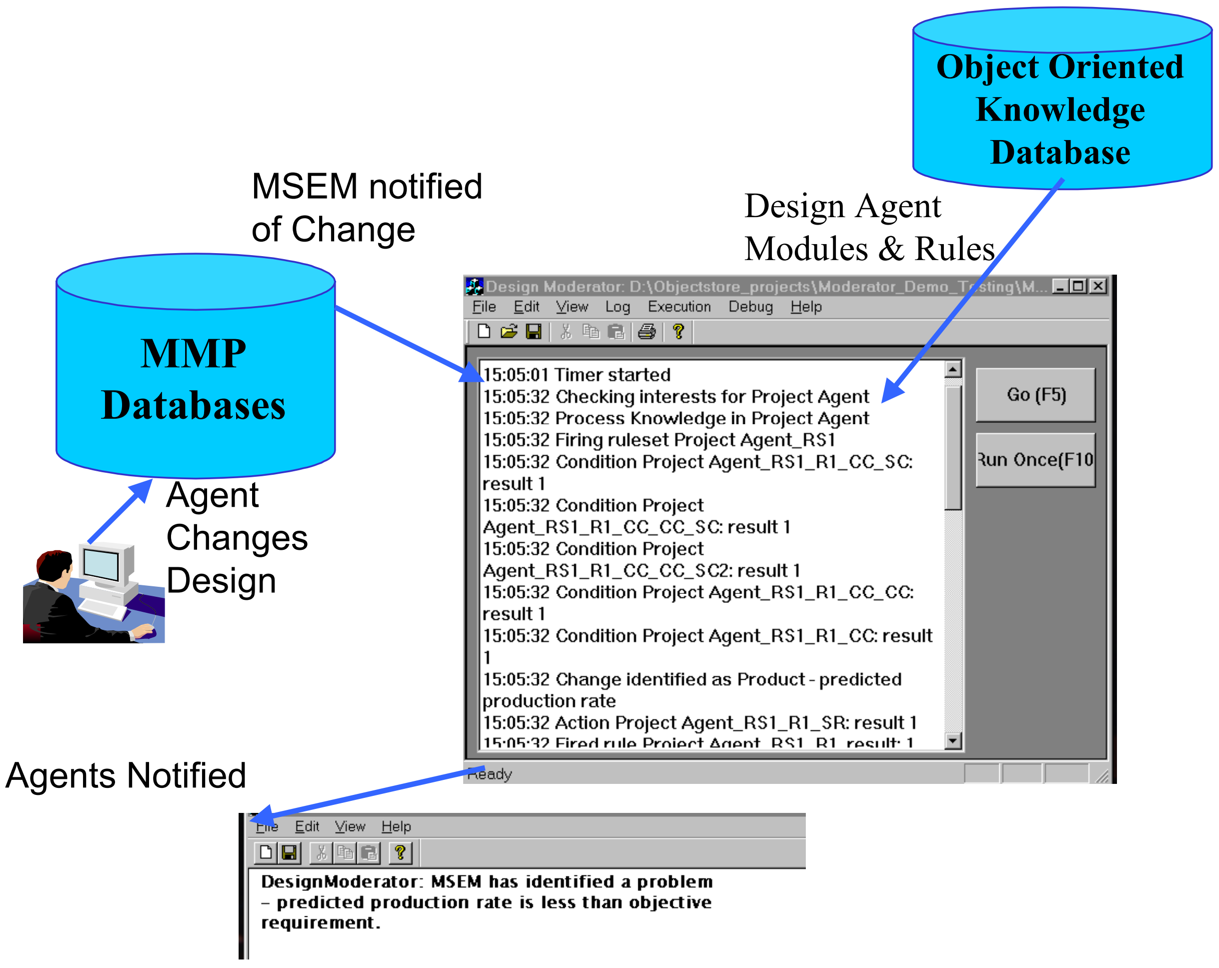
**Figure 3.20: Knowledge Acquisition Module**

Change Evaluation is achieved by the MSEM passing messages to the Design Agent Modules to find out which (if any) of them might be interested in the identified change. When a Design Agent Module is identified as potentially being interested in the change, the MSEM takes the further step of processing all the knowledge it has about the particular Design Agent. This knowledge is also stored as associated persistent objects within the object oriented knowledge database. This knowledge has been structured using the Knowledge Representation Model (KRM) which is described in detail in section 4. When the MSEM processes its knowledge of a particular Design Agent, it may use information relating to the change which has taken place, and information about the interests of the particular Design Agent. It may also gather further information about other elements of the design, via the information manager. Using all these sources of potential information, the MSEM will determine whether Design Agent Notification is necessary. If it is necessary, details of how the Design Agent can be contacted will be extracted from the Design_Agent_Module, and an appropriate message sent to the Design Agent.

Figure 3.21 shows the MSEM working in design moderation mode. When the MSEM is notified that a change has taken place, it checks the Design Agent Modules in its knowledge database to see if any agents might be interested in changes to objects of that type. If interested agents are found, the MSEM processes the knowledge it has associated with them. If the change is found to be significant, the MSEM sends a message to the interested design agents.

#### *3.4.3.1.3 Design Agent Modules*

Each Design Agent Module provides the MSEM with a mental model of a particular Design Agent. Hence, the MSEM will be associated with several (many) different Design Agent Modules, and at any stage of the project, Design Agent Modules may be added, removed or modified. Each model contains some static information about the Design Agent, for example, the name of the Agent, how the Agent can be contacted, what class names of objects the Agent might be interested in, etc. The model also links the Design Agent Module to a knowledge base which provides detailed knowledge for the MSEM to use to determine whether the Design Agent is interested in a particular change. The structuring of this detailed knowledge is complex as potentially it can take many different forms, depending on the expertise and functionality of the Design Agent. For this reason, the prototype implementation of the MSEM has made use of the flexible KRM approach to structure this knowledge.


Object Oriented Knowledge Database
Design Agent Modules & Rules
MSEM notified of Change
MMP Databases
Agent Changes Design
Agents Notified
Design Moderator: D:\Objectstore_projects\Moderator_Demo_Testing\M...
File Edit View Log Execution Debug Help
Go (F5)
Run Once(F10
Ready
File Edit View Help

```
15:05:01 Timer started
15:05:32 Checking interests for Project Agent
15:05:32 Process Knowledge in Project Agent
15:05:32 Firing ruleset Project Agent_RS1
15:05:32 Condition Project Agent_RS1_R1_CC_SC:
result 1
15:05:32 Condition Project
Agent_RS1_R1_CC_CC_SC: result 1
15:05:32 Condition Project
Agent_RS1_R1_CC_CC_SC2: result 1
15:05:32 Condition Project Agent_RS1_R1_CC_CC:
result 1
15:05:32 Condition Project Agent_RS1_R1_CC: result
1
15:05:32 Change identified as Product - predicted
production rate
15:05:32 Action Project Agent_RS1_R1_SR: result 1
15:05:32 Fired rule Project Agent_RS1_R1 result: 1
```

```
DesignModerator: MSEM has identified a problem
- predicted production rate is less than objective
requirement.
```

Figure 3.21: Design Moderation Module

In summary, the requirements identified in sections 2.1, 2.2 and 2.3 have all been satisfied by the prototype implementation of the MSEM as shown below:

*Requirement 2.1.1. is met as follows:*

The MSEM polls and at regular intervals asks the Information Manager for details of any changes that have taken place.

*Requirement 2.1.2 is met as follows:*

The MSEM examines the static information each Design Agent Module in turn, to see if that particular Agent might be interested to changes to objects of the class of object that has been changed. If a Design Agent Module is identified as potentially being interested, the MSEM then goes on to process all the detailed knowledge it has about that Agent, by sending messages to the objects within the Design Agents KnowledgeBase (which is stored in the object oriented database).

*Requirement 2.1.3 is met as follows:*

When the MSEM determines that Design Agent Notification is necessary, it extracts details of how the Design Agent can be contacted from the appropriate *Design_Agent_Module*, and sends a message to the Design Agent, either to the Agent's computer or by email. The prototype MSEM does not remain in contact with the Design Agents to resolve the conflict, and this is an area where further research is required.

*Requirement 2.2.1 is met as follows:*

The Knowledge Acquisition Module of the MSEM should be run, and a Design Agent Module created for all Agents available to the current project.

*Requirement 2.2.2 is met as follows:*

Classes of Objects of interest are stored as part of the static information within each Design Agent Module

*Requirement 2.2.3 is met as follows:*

When the MSEM is operated using its Knowledge Acquisition Module it can either create new Design Agent Modules in a new knowledge database or create new, or delete existing, or change existing Design Agent Modules in an existing knowledge database.

*Requirement 2.2.4 is met as follows:*

Notification of changes is achieved by the MSEM asking the Information Manager for details of any changes that have taken place. Evaluation of each change is made by (always) examining the static information each Design Agent Module in turn and then (if necessary) processing all the detailed knowledge (in the knowledge database) associated with that Agent.

*Requirement 2.2.5 is met as follows:*

Information about how each Design Agent can be contacted is stored as part of the static information within each Design Agent Module. Therefore, if the MSEM needs to communicate conflict warnings to an MSE Agent, details of how the Agent can be contacted will be extracted from the appropriate *Design_Agent_Module*, and the required message sent.

*Requirement 2.3.1 is met as follows:*

The core knowledge for the MSEM simply enables it to process Design Agent Modules. The detailed knowledge which can be associated with a Design Agent Module can take many forms (this is possible due to the KRM approach adopted for knowledge capture/embodiment). In addition, the manner that the detailed knowledge is processed can also be varied, as this is achieved using the process_knowledge method which is a member of the Design Agent Module class. Hence, by specialising Design_Agent_Modules, and the detailed knowledge classes in the knowledge base, which is easily achieved using inheritance and polymorphism, many different types of knowledge can be processessed in many different ways by the MSEM.

*Requirement 2.3.2 is met as follows:*

When the MSEM is operated using its Knowledge Acquisition Module it can either create new Design Agent Modules in a new knowledge database or create new, or delete existing, or change existing Design Agent Modules in an existing knowledge database. As Design Agent Modules are stored in databases, it is possible to share the knowledge between projects or even between installations of the MSEM.

*Requirement 2.3.3 is met as follows:*

When the MSEM is operated using its Knowledge Acquisition Module it can either create new Design Agent Modules in a new knowledge database or create new, or delete existing, or change existing Design Agent Modules in an existing knowledge database.

#### *3.4.3.2 Interface Structure*

In order to meet the requirements for communication with Manufacturing Systems Engineering (MSE) Design Agents and the Information Manager, the Manufacturing Systems Engineering Moderator (MSEM) is connected to the MSE Integration Infrastructure (MSE-II).

As with all agents connected to the MSE-II communication uses an Information Manager Adaptor as described in D24, Section 4.2, and detailed in deliverable D13 “MMP Interface Description”, Document 3 “Java and C APIs”.

The MSEM infrastructure is implemented in C++, and it is therefore necessary for it to use the C implementation of the Information Manager Adaptor (IMA).

*Requirement 2.4.1 is met as follows:*

The IMA receives messages from the Information Manager informing the MSEM of changes to the content of the MS Design Database as MESSAGE class objects (see D13: Document 4 “Information Manager Data Model”, section 4.34). These are retrieved by the MSEM by MissionShellGetNextMessage calls to the IMA (see D13: Document 3 “Java and C APIs”, section 4.2.5).

*Requirement 2.4.2 is met as follows:*

The MSEM requests information from the Information Manager by issuing the MissionShellIMReadObject calls to the IMA (see D13: Document 3, 4.3.13). The Information Manager responds by retrieving the requested MS Design Database objects and returning them to the IMA. The MSEM sees this response in the form of returned values from the MissionShellIMReadObject call.

*Requirement 2.4.3 is met as follows:*

Messages are despatched to MSE agents connected to the MSE-II by the MSEM either an email to the MSE agent or a TCPIP message to a listener programme on the MSE agent’s workstation, using addressing information held in its own knowledge base.

### 3.4.4 Implementation

#### *3.4.4.1 Knowledge Representation Model*

The MSEM is an intelligent, knowledge based system which supports the collaboration and co-ordination of many different types of expertise through a variety of different situations which arise during the manufacturing system design process. In the Mission project, the design of a manufacturing system is achieved through the interaction of a group of Agents, each of which has some specific expertise to contribute to the overall system design. As explained in section 1, an Agent has been defined in the Mission project as a combination of human and software. Hence, at one extreme, an Agent may be an intelligent computing system which requires minimal contribution from a human designer, at the other extreme, the software may provide little or no functionality beyond an interface to the Mission environment, and all intelligence will be contributed by the human designer. Many different types of knowledge and expertise may therefore contribute to the design of a manufacturing system using the Mission environment, and the MSEM should be able to store knowledge of what aspects of the design are important to any or all of these agents. In addition, the particular agents who can contribute constructively to the manufacturing system design process, inevitably vary throughout the lifecycle of the project, as different skills are required from project team members. These points are emphasised here as they are very important when deciding on how knowledge about the agents should be stored.

It is therefore essential that the MSEM can capture and store information and knowledge about the Agents working on a project, in a very flexible, versatile manner. There is no single method which can best capture or store the expertise necessary to enable these Agents to work effectively. It is also important that the knowledge be stored in a manner that ensures its reusability in a variety of situations, by a variety of users or applications over a period of time.

The MSEM has been implemented with a separate knowledge base, based on a knowledge representation model (KRM) which was designed to support distributed team working, as exemplified by design teams in concurrent engineering projects (Harding, 1996). Instances of the KRM may be used to capture different types of expertise within an object oriented database. This approach enables knowledge to be stored in database structures alongside product and manufacturing information, thus enabling both information and knowledge to be shared, and reused, by a variety of applications. Throughout this report, the terms data, information and knowledge should be considered to have distinct meanings. Data relates simply to words or numbers the meaning of which may vary and is dependent upon the context in which the data is used. Information is data which is structured or titled in some way so that it has a particular meaning. Knowledge is information with added detail relating to how it may be used or applied. Thus, in terms of a value line, data is at one end (being least valuable), and knowledge at the other (being most valuable), with information somewhere between.

#### *3.4.4.2 Design of Knowledge Representation Model*

The KRM concept enables software expertise to be represented by one or more modules. In the case of the MSEM, knowledge about individual Design Agents, and the knowledge about what changes are important to them, and what actions should be taken if such changes occur, are stored in classes called Design Agent Modules. Each Design Agent Module knows how to process its own knowledge, as this behaviour is implemented in methods of the class. The processing of knowledge is achieved by message passing between instances of various classes, including Ruleset, Rule, Condition and Action objects. The names used to identify classes associated with the MSEM and Design Agent Modules are very similar to the terminology of standard expert systems, in particular of *production systems*. This is a deliberate use of metaphor, but it must be remembered that in the KRM and the current research, we are working with classes of objects, so instances of these classes have state, behaviour and identity, and therefore the objects provide all the flexibility and power of object oriented systems. By making use of inheritance structures and polymorphism, it has been possible for the similarity of behaviour of certain classes of objects to be exploited even though the implementations of particular classes of objects is significantly different. This point is central to the concepts of this work.

Figure 3.22 shows the KRM class structure used to implement the knowledge in the prototype MSEM. The Moderator_Working_Memory class stores information which the MSEM uses during the design moderation process. This includes details of the changed object, and the design agent current under consideration. Design agent modules were described in section 3.1.3, and the class used to store these objects in the knowledge database is shown in Figure 8. Each design agent module class is linked to knowledge about the design agent, and this is captured in the database as a list of Ruleset objects. Each Ruleset can contain any number of Rule objects, and each Rule is associated with a Condition object and a Resulting Action Object. These can be either simple or compound. A Compound Condition contains a Simple Condition and a Condition, which are connected using either AND or OR, and any Condition can be negated if required. Compound Resulting Actions contain a Simple Resulting Action and a Resulting Action and these are connected using AND.

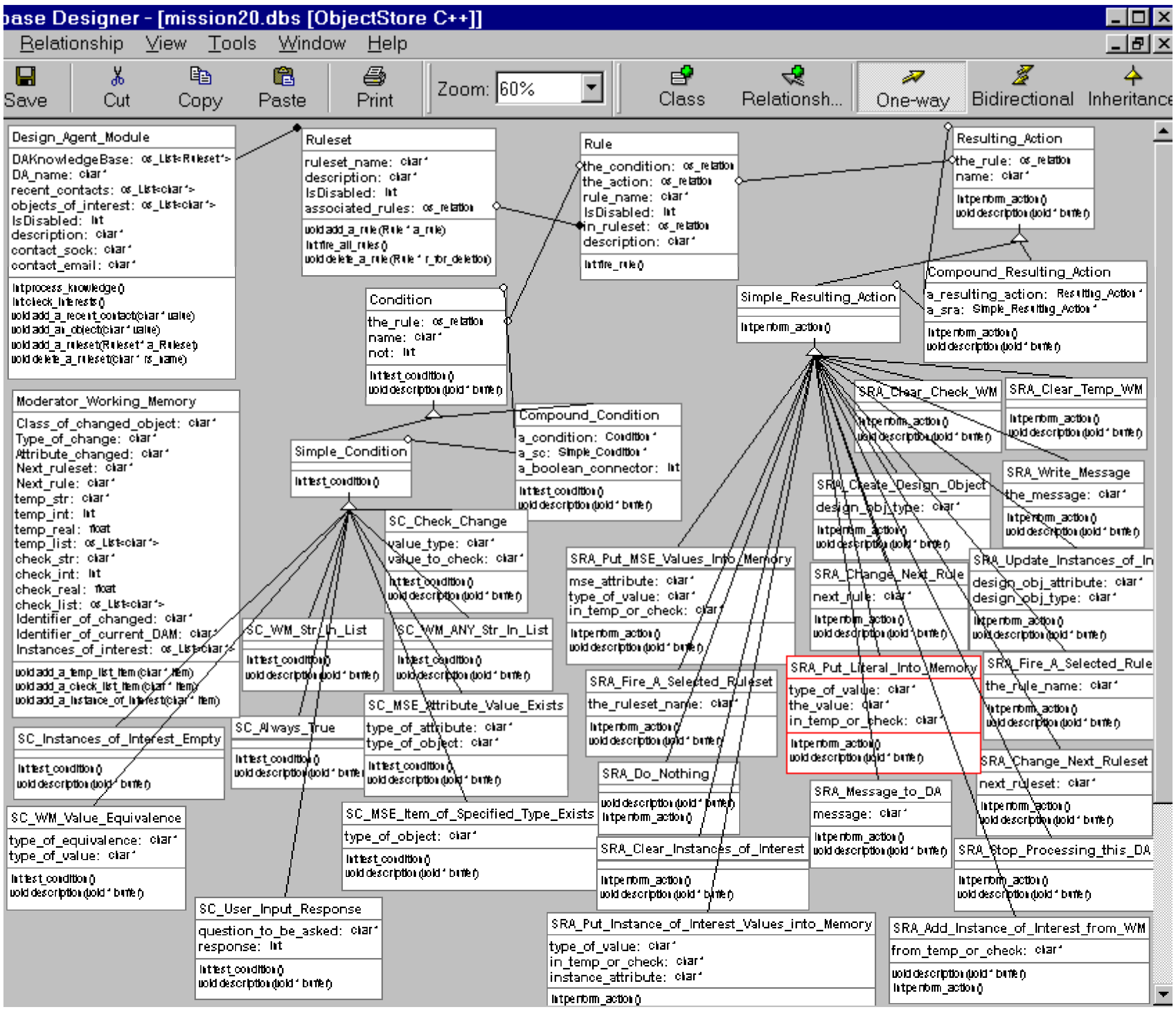

base Designer - [mission20.dbs [ObjectStore C++]]
Relationship View Tools Window Help
Save Cut Copy Paste Print Zoom: 60% Class Relationsh... One-way Bidirectional Inheritance
Design_Agent_Module
Ruleset
Rule
Resulting_Action
Compound_Resulting_Action
Simple_Resulting_Action
Condition
Moderator_Working_Memory
Compound_Condition
Simple_Condition
SC_Check_Change
SRA_Clear_Check_WM
SRA_Clear_Temp_WM
SRA_Create_Design_Object
SRA_Write_Message
SRA_Put_MSE_Values_Into_Memory
SRA_Change_Next_Rule
SRA_Update_Instances_of_In
SC_WM_Str_In_List
SC_WM_ANY_Str_In_List
SRA_Put_Literal_Into_Memory
SRA_Fire_A_Selected_Rule
SRA_Fire_A_Selected_Ruleset
SC_MSE_Attribute_Value_Exists
SC_Always_True
SC_Instances_of_Interest_Empty
SRA_Do_Nothing
SRA_Change_Next_Ruleset
SRA_Message_to_DA
SC_WM_Value_Equivalence
SC_MSE_Item_of_Specified_Type_Exists
SRA_Clear_Instances_of_Interest
SRA_Stop_Processing_this_DA
SC_User_Input_Response
SRA_Put_Instance_of_Interest_Values_into_Memory
SRA_Add_Instance_of_Interest_from_WM

**Figure 3.22: KRM Class Structure used to Implement Prototype MSEM**

The real processing power of the KRM comes through the sub-classes of the Simple Condition and the sub-classes of the Simple Resulting Action. The only thing that the sub-classes of Simple Condition have in common is that they inherit the method, int test_condition (void), which returns either 1 (for true) or 0 (for false), but thanks to polymorphism, the activities which are performed to determine whether 1 or 0 should be returned are very varied. For example, SC_Always_True will always simply return the value 1. In contrast, when the method test_condition is run for an SC_Check_Change Simple Condition object, either the type of the actual change (Create, Modify or Delete), or the class of the changed object, or the attribute of the changed objected, notified to the MSEM is checked against a value created when the knowledge was added to the knowledge base. If the value the value matches, the value 1 is returned by the SC_Check_Change Simple Condition object, but if they do not match, the value 0 is returned. Figure 8 shows several different sub-classes of Simple Condition which have been implemented for this prototype version of the MSEM. Similarly, the only thing that the sub-classes of Simple Resulting Action have in common is that they have a perform_action method. The activities which are carried out in this method are very varied, thanks to opportunities provided by polymorphism. For example the perform_action method of SRA_Message_to_DA is used to send messages to Design Agents. The perform_action methods of other sub-classes of Simple Resulting Action are used to manipulate and test information within the MSEM's working memory, or to change the order in which Rules are processed, or to collect further information from the MMP.

#### *3.4.4.3 Selection of KRM*

The KRM was selected as the medium for capturing and implementing knowledge within the MSEM for several reasons, including

- Flexibility - the KRM allows knowledge to be stored and applied by executing many individually distinct, but interacting pieces of code. Several different artificial intelligence paradigms can therefore be used to model and implement different parts of the captured knowledge. The KRM is therefore very flexible and versatile
- Knowledge is stored as persistent objects within an object oriented database system. Hence the KRM approach takes advantage of the maintenance and administration functionality provided by the database.
- The value of the KRM approach has been identified in earlier projects as it has now been used in several projects, producing software implementations which have been tested and demonstrated, by capturing different types of expertise (Harding and Popplewell, 1996), (Omar et al, 1999), (McKay, 1995).

#### *3.4.4.4 OO Database Selection*

The KRM makes extensive use of object technology, especially polymorphism, in its operation. The KRM has been designed using UML, and is therefore independent of any particular implementation environment. It could be implemented in any truly object oriented database, or in an alternative environment which can support persistent objects, message passing between objects, inheritance and polymorphism.

ObjectStore was selected for the prototype implementations of the KRM in the Mission Project as it is an object oriented database rather than an object/relational database. It also has the benefit of providing excellent support for object collections of various types, which reduced some of the programming needs. Oracle was not used for implementations of the KRM and MSEM moderator as it was judged that additional programming would be required to achieve the required functionality for interactions between object methods, collections of objects and polymorphism.

### 3.4.5 Review of MSEM Demonstration within MISSION

For details of the demonstration scenarios refer to D24 Chapter 5, where conflicts to be detected and reported in demonstration are described in context.

#### *3.4.5.1 Conflicts Demonstrated*

*Stage 2, Conflict 1:* This occurs if the Manufacturing Process Design Agent (MPDA) selects a facility which is incompatible with the transport mechanism defined by the Supply Chain Agent(SCA). For example in the MISSION demonstration MS components are manufactured at Bosch, Stuttgart for subsequent assembly. The SCA could decide to despatch components by boat from this plant. Subsequent selection of the EADS assembly facility in Madrid will cause conflict in transport, as boat transport to Madrid is not feasible. On detection such a conflict is reported to the SCA and the MPDA for resolution.

*Stage 3, Conflict 2:* This occurs if the black box supply chain simulation analysis predicts a supply chain performance inadequate to meet MS design objectives. As a simple example, if predicted annual production rate is less than required annual production rate a conflict may be reported. In a more sophisticated implementation the comparison could be based on confidence limits on predicted annual production rate, based on multiple simulations, and multiple objectives could be considered simultaneously. On detection such a conflict is reported to the Project Agent, SCA and MPDA for resolution.

*Stage 4, Conflict 3:* This occurs when lot sizing for manufacture in a facility does not match the global inventory policy. For example if the SCA has determined a transport batch size and frequency to maintain downstream supplies, whilst details MS design in a facility determines large, less frequent batches. Such a conflict is reported to the SCA and the MPDA as well as facility design agents for resolution.

*Stage 4, Conflict 4:* This is similar to Conflict 2 above, arising when predicted performance of an individual facility MS design does not match the estimates used in the supply chain design (Stage 3) and black box simulation. Such a conflict is reported to the SCA, MPDA and local facility design agents for resolution.

*Stage 4, Conflict 5:* Internal conflicts in the design of an individual facility MS can arise as different internal design agents contribute their expertise locally. Such conflicts need to be reported to the relevant internal design agents for resolution. For example, a change of a machine within the facility, proposed to expand production capacity, may increase the footprint of the machine in the facility layout. In turn this may reduce gangway dimensions and conflict with selected internal transport vehicles. In this case the local process design agent and transport agent must be alerted to the need to resolve the conflict.

*Stage 5, Conflict 6:* Conflicts can arise between the predicted performance of the detailed transport network design cannot achieve the performance estimates used in Stage 3. Such conflict is reported to the SCA and Simulation Agent (SA).

*Stage 6, Conflict 7:* During simulation of the global supply chain it is possible for any aspect of predicted performance to conflict with MS specification. Conflict is reported to the MS design agents concerned.

#### *3.4.5.2 Knowledge Required for Demonstration*

*Stage 2, Conflict 1*

Transport method selected for despatch must be compatible with receiving facility.

Location of despatch transport method in MS Design Database.

Location of receiving facility capabilities in MS Design Database.

That SCA and MPDA have interest in transport methods and must be informed of potential conflict.

*Stage 3, Conflict 2*

Predicted MS performance must, within known acceptable tolerance, match specified MS requirements.

Location of MS performance requirements in MS Design Database.

Location of predicted MS performance in MS Design Database.

That Project Agent (PA), SCA and MPDA have interest in performance of the global supply chain and must be informed of potential conflict with MS requirements.

*Stage 4, Conflict 3*

Knowledge that facility lot sizing and global inventory policy must be compatible in respect of despatch quantities and frequency. Compatibility is understood in terms of known acceptable tolerances.

Location of facility lot size specifications in MS Design Database.

Location of global inventory policy in MS Design Database.

That SCA, MPDA and facility design agents have interest in lot sizing and transport frequencies and must be informed of potential conflict.

*Stage 4, Conflict 4*

Knowledge that facility performance predicted by facility simulation must be within tolerance of the estimates of performance used in black-box simulation of the global supply chain.

Location of facility performance predictions in MS Design Database.

Location of facility performance estimates in MS Design Database.

That SCA, MPDA and facility design agents have interest in predicted facility performance and must be informed of potential conflict with global MS design requirements.

*Stage 4, Conflict 5*

Knowledge of the range of conflicts which may arise in the design of an MS within the particular facility.

Location of potentially conflicting design information in MS Design Database.

The interests of facility design agents.

*Stage 5, Conflict 6*

Knowledge that predictions of performance of detailed transport network design must be within known acceptable tolerance of the estimates used in black-box simulation of the global supply chain.

Location of estimates of transport network performance in MS Design Database.

Location of predictions of transport network performance in MS Design Database.

That SCA and SA have interest in predicted transport network performance and must be informed of potential conflict with global MS design requirements.

*Stage 6, Conflict 7*

All the above knowledge can be re-used at this stage as more detailed predictions of performance are generated.

### ***3.4.5.3 Critical Review of Results***

#### 3.4.5.3.1 Moderation Process

*Requirement 2.1.1:* This is met in full, but see 2.4.1 below.

*Requirement 2.1.2:* The MSEM prototype is fully able to determine action to be followed on receipt of change notification, requesting information from the Information Manager as necessary.

*Requirement 2.1.3:* The MSEM holds within its own knowledge contact details for each MSE agent, including details of email and workstation addresses. Messages alerting the MSE agent to a potential conflict of interest are sent to the agent by email or to a listener programme at the workstation, meeting this requirement in full.

#### 3.4.5.3.2 Knowledge of MSE Agents

All MSE agent knowledge has been captured in the MSEM knowledge base.

#### 3.4.5.3.3 Knowledge Acquisition

The approach adopted for meeting knowledge acquisition requirements is to store MSEM current knowledge in an Object Oriented Database (OODB), whilst including within the MSEM the capability to add to or amend its own knowledge base by updating the database. Furthermore the whole of the MSEM's functionality is implemented as object structured knowledge. Thus knowledge about new MSE agents can be introduced, and, significantly, as an organisation becomes aware of new ways of identifying developing conflicts (usually through bitter experience) knowledge of these can be added. This functionality is achieved through the use of a Knowledge Representation Model. Each organisation implementing the MSEM can embed its own knowledge and experience and then continue to develop this, making full use of its corporate knowledge as a competitive advantage.

This object oriented, database resident, knowledge structure is always open to extension and refinement. Figure 3.23 illustrates the evolution of MSEM knowledge from the generic underlying core functionality. The first level of possibly useful MSEM knowledge is specialised to meet the common needs of an industrial sector, but in truth this is of very limited value until it is specialised at least enough to reflect the manufacturing systems engineering resources in a particular enterprise. At this point it is possible to embed in the MSEM knowledge of the interests of the MSE agents providing these resources. This is then applied when an MSEM is initialised for a particular MSE project: from this point MSEM knowledge is extended for the project itself, firstly by identifying which MSE agents are involved in the project, and subsequently by specialising knowledge of design features critical to the particular project.

It is also possible that as a project progresses the knowledge base of the MSEM in use for that project may evolve. For example the project team may choose to tighten the threshold at which a change in predicted performance of a facility in the supply chain triggers a conflict warning, because that facility is seen to be critical to overall MSE performance.

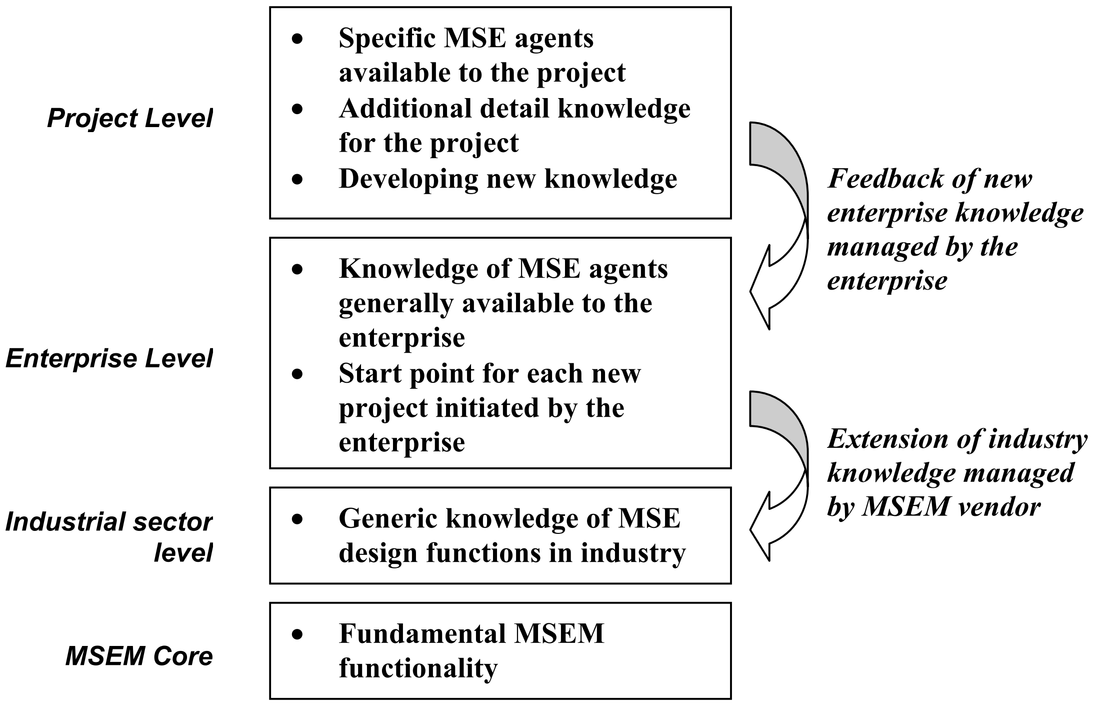

Project Level
Specific MSE agents available to the project
Additional detail knowledge for the project
Developing new knowledge
Enterprise Level
Knowledge of MSE agents generally available to the enterprise
Start point for each new project initiated by the enterprise
Industrial sector level
Generic knowledge of MSE design functions in industry
MSEM Core
Fundamental MSEM functionality
Feedback of new enterprise knowledge managed by the enterprise
Extension of industry knowledge managed by MSEM vendor

**Figure 3.23: MSE Moderator Evolution within an Enterprise**.

Another, rather different example of this might arise if the project team discover a conflict which has not been detected by the MSEM. Here the Moderator's knowledge can be extended to ensure that such conflicts are detected should they recur. Furthermore consideration can be given to the enterprise level Moderator used to start future projects so that new knowledge is retained and propagated throughout the enterprise.

The prototype MSEM implemented within the MISSION project includes a knowledge acquisition module. This is sufficient to demonstrate feasibility of capture of all the above, thus meeting all knowledge acquisition requirements, but a commercial implementation would require a more user-friendly interface. Further research is also required on the methodology of knowledge acquisition for MSE moderation, but this is beyond the scope of the MISSION research.

#### 3.4.5.3.4 MSE Infrastructure Communications

*Requirement 2.4.1:* The MSEM successfully receives messages alerting it to change in the MSE Design Database. However there are performance issues arising from the TCPIP messaging mechanism adopted in the Information Manager (IM). Essentially all messages issued by the IM, regardless of intended recipient, must be processed by the MSEM if only to be ignored as addressed elsewhere. A more specific addressing mechanism, implemented either using HLA-RTI technology, or in some other way allowing the IM Adaptor to reject unwanted messages, is needed to improve performance.

*Requirement 2.4.2:* This requirement is met in the prototype.

*Requirement 2.4.3:* This requirement is met in the prototype.

### 3.4.6 Exploitation

The development of an MSEM has been essentially a research activity to demonstrate feasibility through the implementation of a prototype within the MISSION MMP context. Feasibility has been successfully demonstrated. Three exploitation paths are open.

#### *3.4.6.1 Collaboration with Software Exploiters.*

The development of a commercial implementation of the MSEM must be conducted in collaboration with the MMP software developers, as the MSEM and MMP must communicate closely. LU and CU are willing to participate, on mutually agreeable terms to be agreed, in the implementation of MSEM software within a commercial MMP product.

#### *3.4.6.2 Pilot Implementation in a MISSION Implementation.*

LU and CU would be keen to participate in the implementation of an industrial application of the MMP including an MSEM. Such collaboration would necessarily involve the software exploiters as vendors of the commercial MMP, and an end user company.

#### *3.4.6.3 Future MSEM Research*

Demonstration of the feasibility of an MSEM within the MISSION research leads to future research opportunities, seen by the participating universities as their exploitation of the research.

##### *3.4.6.3.1 Methodologies for Moderation Knowledge Capture*

The process of MSE design is not well understood or documented. In contrast with product design where a number of formal design process models are proposed, no such models exist in the field of MSE design. Work on the Project Agent within MISSION begins to address this issue, but a fuller investigation is beyond the scope of MISSION. Earlier research on a Factory Design Process (Yu *et al.*, 2000) contributes an MSE design model.

There is now a need for research to establish a taxonomy of knowledge relevant to the process of MSE Moderation, which in turn will lead to the definition of an effective methodology for capture and maintenance of moderation knowledge.

##### *3.4.6.3.2 Extension of Knowledge Acquisition Interface*

There is further scope for research into an MSEM knowledge acquisition interface which is both user friendly, and which can be related directly to the knowledge capture methodology outlined in 6.3.1 above.

# 4 Reference Model for Distributed Manufacturing and Simulation

## 4.1 Template Library and Exchange Objects

### 4.1.1 Introduction to the Template Library

Within the MSE process, the template library is mainly used as a flexible knowledge base. For this purpose, attributes can be defined depending on the used design agents. Some attributes are pre-defined according to the requirements of the MMP or according to user requirements. The user has the chance to add templates and attributes of templates. Additionally, the user can use objects of these templates within the MSE process. For example, the developer may decide to use a special AGV. If an AGV template already exists as a subclass of the "transport" template the developer can use it. Otherwise the developer can create a new template class and add the necessary attributes, as far as they are not derived from the transport class. Now the developer can create an object of the AGV class and set the values. Furthermore, the AGV object can be used during the design process. If a simulation model is available for the AGV class and the relevant attributes for the simulation were set, then this AGV object can be used directly within a simulation scenario.

Concerning the simulation process, the template library contains a reference to simulation models for each application template. The simulation model implements the content of the application template. Each model has to be able to execute this partial simulation process by itself. The notation "application template" will be used for templates directly applicable within a simulation scenario.

Additional descriptions are necessary to execute different simulation models in a distributed simulation scenario for a template. It is important to describe the relations between the different templates. In order to enable an automatic generation of simulations based on a given simulation scenario, an object scheme required. This scheme describes the objects exchanged with other simulation models ("Common Exchange Object Model"). Furthermore, the exchanged objects can be divided in two types:

1. Objects including necessary information but not associated with other objects
2. Objects including necessary information and associated with objects of other simulation models (e.g. the port of a warehouse is associated with a net node of a transport system).

In order to avoid modelling all the exchanged objects for each single simulation scenario anew , the template library includes an additional structure of common exchanged objects. This structure can be expanded by the template library administrator following the template library rules. The rules include mainly:

- Clear and unique name space for objects,
- Clear and unique name space for attributes within one inheritance tree,
- Unique semantic structures of each object class and
- Unique semantic structures of each attribute within one object class.

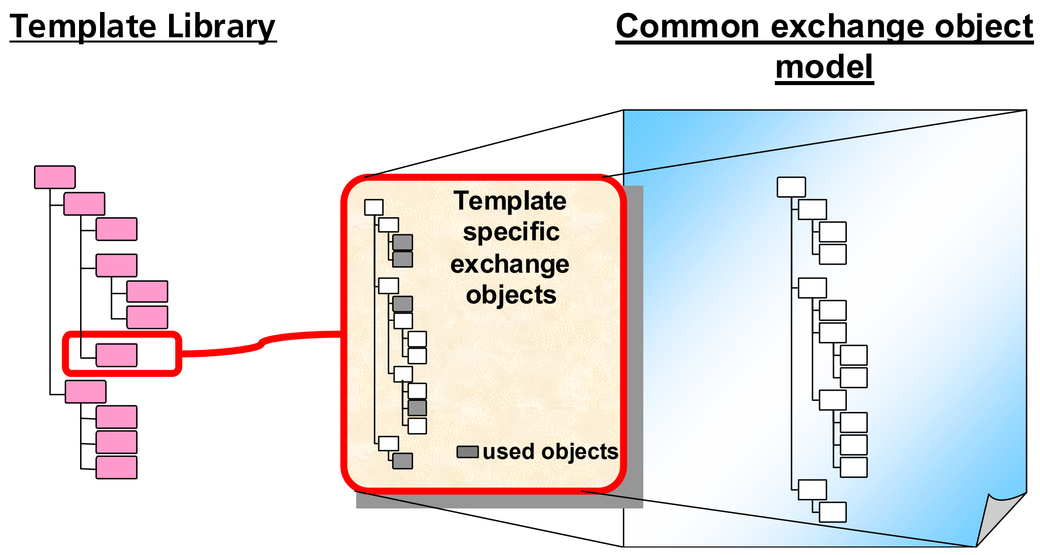

Template Library
Common exchange object model
Template specific exchange objects
used objects

**Figure 4.1: Relation Between a Single Template of the Template Library and the Common Exchange Object Model**

This approach will allow an automatic generation of a common exchange object model for a user defined simulation scenario. Furthermore, it becomes possible to configure a simulation scenario including an automatic generation of HLA-RTI-FED files and federate configuration files.

An additional part of an application template is the graphical representation of the template. 2D and 3D representations can be used for the graphical representation of the template. A possible format for the graphical representation is VRML. VRML allows a modular structure of objects within a 3D animation.

Summarising, a template contains:

- References to simulation models or other applications (one template can containing links to more than one simulation models. But every of this models must satisfy the description of the Template and support the parameters setting and the Exchange Objects),
- Description of exchanged objects,
- Definition of connectivity elements with information about input or output segments like ID and description (for example which data exchange the input/output segment is included and provided),
- Visualisation of the template and
- Parameter descriptions of the template with name, short description, unit and default value.

Now the templates can be used for the definition of simulation scenarios by deriving building blocks from the template library.

### 4.1.2 Advantages of the Template Library Approach

Within different simulation systems mechanisms like templates, modules or classes are already available (e.g. Arena, eMPlant). These mechanisms have been further improved by reference models (Mertins et al. 1998).

The disadvantage of these methodologies was that they are specific for one simulator and currently not compatible for different simulators.

The advantage of the template library is that this approach includes a mechanism to reuse simulation models from different simulators in different simulation scenarios. Furthermore, it allows distributed simulations between different enterprises without the necessity for these enterprises to use the same simulator, nor to exchange the information of their models.

HLA allows the connection of different simulation models via a runtime infrastructure (RTI) called federation. However, it is still a disadvantage, that each federation needs a hard programming of the interface between the federates and the RTI. This is one of the reasons, why the HLA is not used more frequently within the civil area.

The advantage of the MISSION template library approach is that the simulation manager, as a part of the template library implementation, will implement the generation of configuration files for federates participating in the simulation scenario to avoid intense programming.

### 4.1.3 Building Blocks

An application template can be seen as a building block of a huge simulation scenario, or as a predefined component of a manufacturing design like a transport system, a warehouse, etc.. Therefore, a building block in this context is defined as an object derived from a template of the template library.

A simulation study applying the building blocks consists of the following steps:

1. Selection of suitable templates
2. Insertion of a building block which is derived from the selected templates into the simulation scenario
3. Parameterisation of the building blocks
4. Selecting the relevant region of the scenario for the simulation
5. Consistency check and if necessary supplementation of the selected simulation scenario
6. Execution of the simulation scenario.

The template library supports the selection of suitable templates by a search structure and the special indication of application templates. Highlighting the templates already used within the current MSE project will furthermore speed up the selection process. The insertion of the building block into the simulation scenario can be done during the MSE process in order to identify the required processes and resources for the manufacturing system (MS). Then the model and especially the building blocks derived from the templates can be used as an information base for the whole design process. The information about the objects within the manufacturing process model becomes more and more supplemented during the MSE process.

In the next step this manufacturing process model is the base of the simulation scenario. A region of the model or the whole model can be selected as simulation scenario.  Furthermore, a completeness and consistency check is started to identify missing parameter settings, missing simulation models and so on. For each building block, the user has to set the missing parameters, as far as they are required by the simulation model. For this task some supporting wizards are proposed.

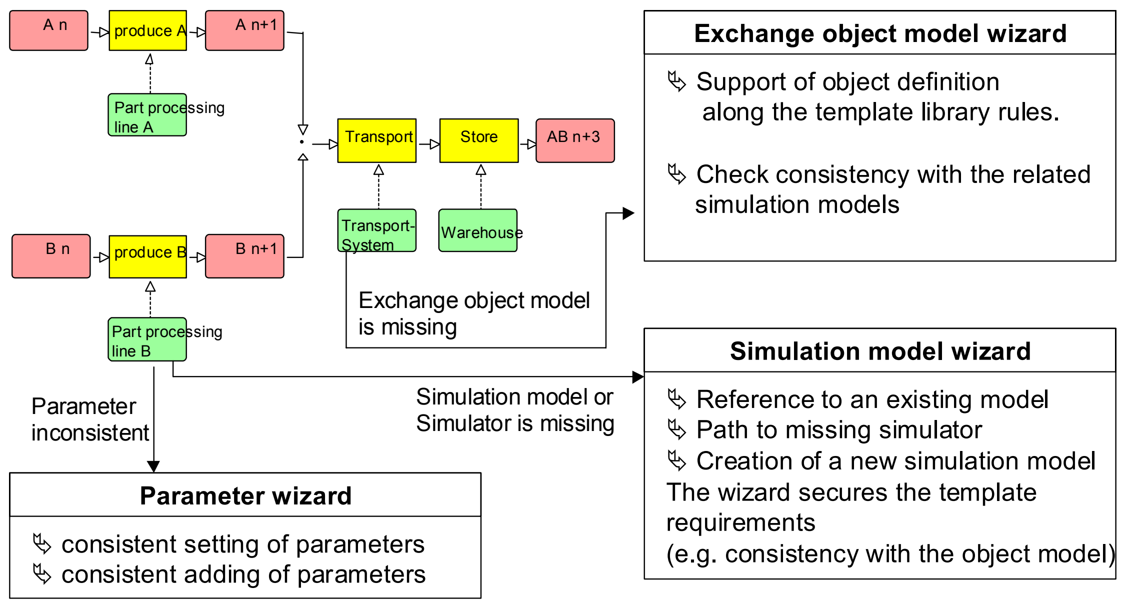

A n
produce A
A n+1
Part processing line A
Transport
Store
AB n+3
Transport-System
Warehouse
B n
produce B
B n+1
Part processing line B
Exchange object model is missing
Simulation model or Simulator is missing
Parameter inconsistent
Exchange object model wizard
Support of object definition along the template library rules.
Check consistency with the related simulation models
Simulation model wizard
Reference to an existing model
Path to missing simulator
Creation of a new simulation model
The wizard secures the template requirements (e.g. consistency with the object model)
Parameter wizard
consistent setting of parameters
consistent adding of parameters

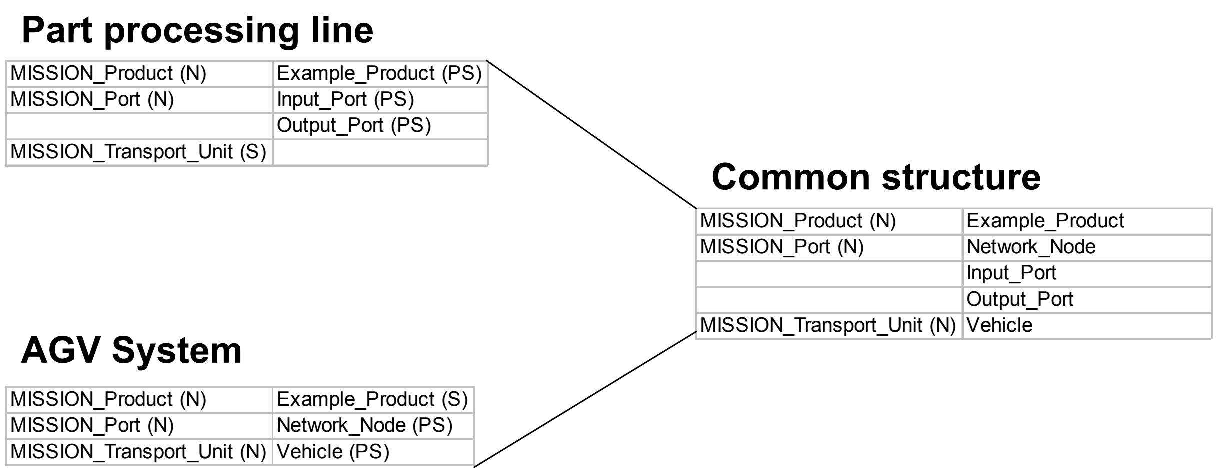
**Figure 4.2: Simulation Scenario Check**

An important task is to connect the connectivity elements of the different building blocks. Each building block has a set of connectivity elements. It is necessary to define the input and output of the building blocks via connectivity elements defined for the building blocks. For example, the user has to associate the output of a processing line with a station of a transport system. When the simulation scenario is complete and consistent, the user can execute it. Three steps will be done during the initialisation of the execution:

1. The generation of a common information model and the generation of the configuration files
2. The generation of the runtime environment of the simulation scenario
3. The start of the runtime scenario.

The common information model of exchanged objects can be generated easily on the base of the exchange object model being part of each template, if the template library rules are adhered to. The different exchange object classes can be integrated into one common class structure based on the common exchange object model of the template. This is done by the following steps:

1. Modelling of a simulation scenario by application of building blocks derived from the templates
2. Supplementation or setting of the necessary parameters of the building blocks for simulation
3. Merging of the exchange object classes and generation of a common structure for the simulation scenario
4. On this base, automatic generation of a FED file to configure the RTI
5. Generation of the federate configuration files from the model of the simulation scenario
6. Conduction of experiments by simply changing the parameter settings.



**Part processing line**

| | |
|---|---|
| MISSION_Product (N) | Example_Product (PS) |
| MISSION_Port (N) | Input_Port (PS) |
| | Output_Port (PS) |
| MISSION_Transport_Unit (S) | |

**Common structure**

| | |
|---|---|
| MISSION_Product (N) | Example_Product |
| MISSION_Port (N) | Network_Node |
| | Input_Port |
| | Output_Port |
| MISSION_Transport_Unit (N) | Vehicle |

**AGV System**

| | |
|---|---|
| MISSION_Product (N) | Example_Product (S) |
| MISSION_Port (N) | Network_Node (PS) |
| MISSION_Transport_Unit (N) | Vehicle (PS) |

**Figure 4.3: Merge of Exchange Object Classes of Different Templates**

### 4.1.4 Application Templates and Exchange Objects

Application Templates and Exchange Objects are topics which are narrowly connected with each other. In the development process new ideas or approaches to the Application Template definition will influence the "Exchange Object structure". So the Application Template and Exchange Object approaches will be updated continuously.

Within the MSE process, the template library is mainly used as a flexible knowledge base. For this issue, attributes can be defined depending on the used design agents. Some attributes are predefined according to the requirements of the MMP or according to the available user requirements. The user has the possibility to add templates and attributes of templates. Furthermore, the user can use objects of these templates within the MSE process.

For example, the developer may decide to use an AGV with the following characteristics (Table 4.1):

| Characteristic | Type | unit | value |
|---|---|---|---|
| **Speed** | **Real** | **m/s** | **1** |
| **turning radius** | **Real** | **m** | **0.6** |
| **Weight** | **Real** | **kg** | **2** |
| **...** | | | |

**Table 4.1: Characteristic of a special AGV**

After this initial situation, if under the transport template an AGV template already exist then the developer can use it. Otherwise the developer can create a new template class and add the necessary attributes, if they are not derived from the transport class. Now the developer can create an object of the AGV class and set the values. Further, the AGV object can be used during the design process. If a simulation model is available for the AGV class and the relevant attributes for the simulation were set, then all AGV objects can be used directly within a simulation scenario.

Additional descriptions are necessary to execute different simulation models in a distributed simulation scenario for a template. It is important to describe the relations between the different templates. If an automatic generation of simulations based on a simulation scenario is wanted, an object schema is necessary including the objects exchangeable with other simulation models. Furthermore, the exchange objects can be divided in two types of objects:

1. Objects including necessary information but not associating to other objects
2. Objects including necessary information and associating to objects of other simulation models (e.g. the port of a warehouse and a net node of a transport system)

To avoid the data modelling approach of exchange objects for each simulation scenario the template library props an additional structure of common exchange. This structure can be expanded by the template library administrator following the template library rules.

The common information model of exchange objects can be generated easily on the base of the exchange object model behind each template of a building block, if the template library rules are satisfied. The different exchange object classes can be integrated into one common class structure based on the common

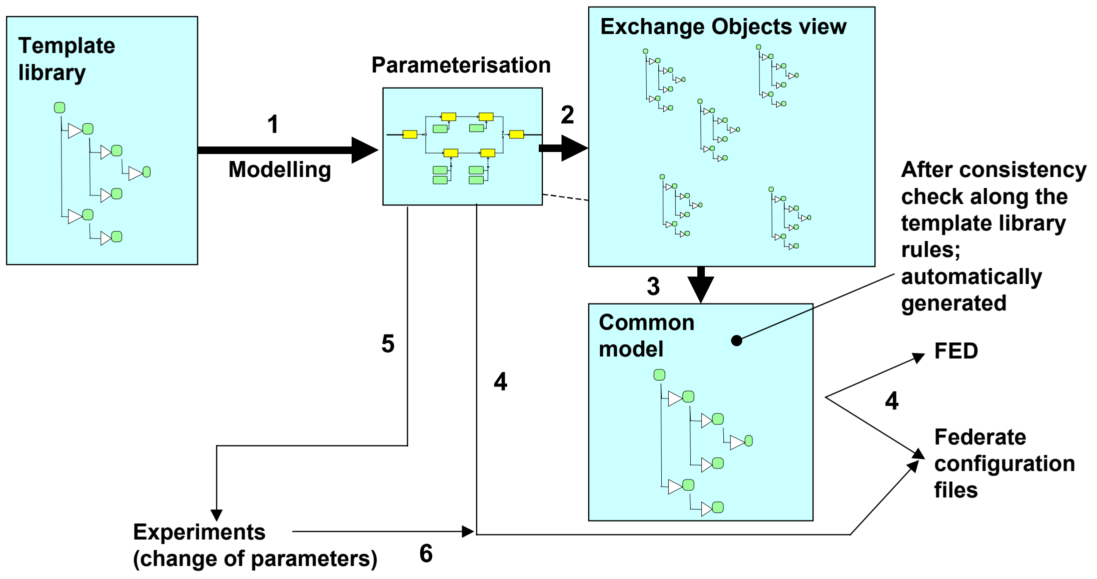

Template library
Parameterisation
Exchange Objects view
1
Modelling
2
After consistency check along the template library rules; automatically generated
3
Common model
5
4
FED
4
Federate configuration files
Experiments (change of parameters)
6

**Figure 4.4: Generation of the simulation scenario runtime configuration files**

exchange object model of the template library.

This approach allows an automatic generation of a common exchange object model of a user defined simulation scenario. Furthermore, it opens the possibility to configure a simulation scenario providing automatic generation of the HLA-RTI-FED file and federate configuration files.

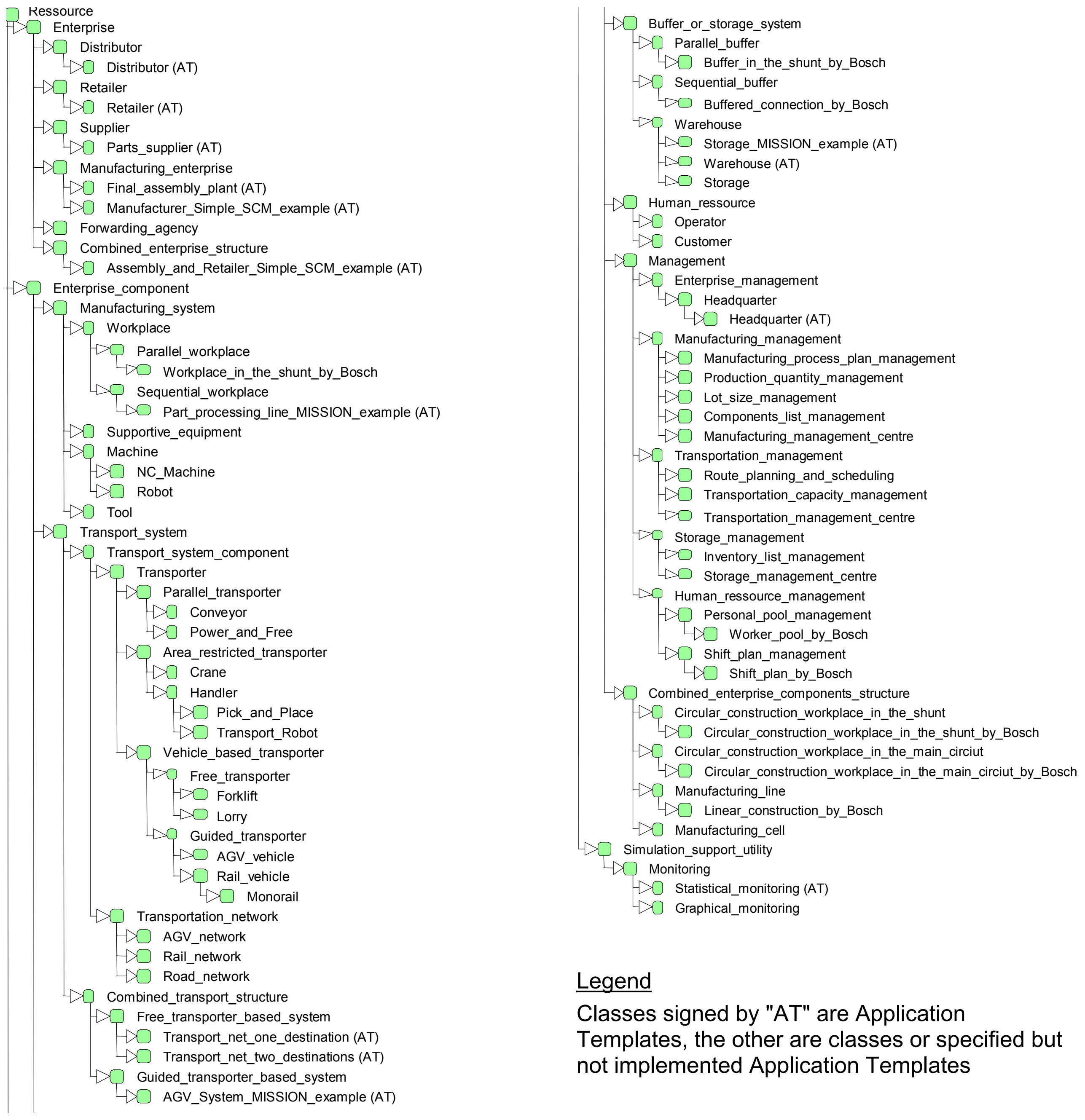
The generation of the runtime environment of the simulation scenario is based on the previous generated federate configuration files. Each simulation model involved within the simulation scenario gets its own federate configuration file including all information necessary to configure the simulator interface for this simulation model. So the next task is the consistent combination of simulator, simulation model and federate configuration file.

Afterwards the HLA-RTI will be started using the previously generated FED file. Now the different components of the simulation scenario runtime environment will be executed. Additionally, Figure 2.2 shows an Adapter to the RTI that allows an easier connection of software tools. The adapter will support the retrieval of class and attribute information described within the federate configuration file. Further it will support the association between class and attribute descriptions within the federate configuration file and the class and attribute names within the FED file. Therefore, it will be easy to retrieve information for an attribute name like value type, default values of attributes, etc (Figure 2.2).

### 4.1.5 Application Templates

The approach seen so far bases on the template definition. All Application Templates are classes within the Template library and contain a reference to simulation model or other application (like Excel), a set of attributes to parameterise the model and a set of links to Exchange Object classes. The consistent combination of template attributes, parameter of simulation models and the exchange-object-model is very important.

An Application Template is independently of the detail degree of the linked simulation model. That means, it can describe a complete enterprise just like a very small detail of an enterprise (machine, vehicle etc.) That’s why, the class structure is orientated to the construction and mode of operation of a manufacturing enterprise. The following Figure 4.5 illustrates an example for the class structure of the Template Library.


Ressource
Enterprise
Distributor
Distributor (AT)
Retailer
Retailer (AT)
Supplier
Parts_supplier (AT)
Manufacturing_enterprise
Final_assembly_plant (AT)
Manufacturer_Simple_SCM_example (AT)
Forwarding_agency
Combined_enterprise_structure
Assembly_and_Retailer_Simple_SCM_example (AT)
Enterprise_component
Manufacturing_system
Workplace
Parallel_workplace
Workplace_in_the_shunt_by_Bosch
Sequential_workplace
Part_processing_line_MISSION_example (AT)
Supportive_equipment
Machine
NC_Machine
Robot
Tool
Transport_system
Transport_system_component
Transporter
Parallel_transporter
Conveyor
Power_and_Free
Area_restricted_transporter
Crane
Handler
Pick_and_Place
Transport_Robot
Vehicle_based_transporter
Free_transporter
Forklift
Lorry
Guided_transporter
AGV_vehicle
Rail_vehicle
Monorail
Transportation_network
AGV_network
Rail_network
Road_network
Combined_transport_structure
Free_transporter_based_system
Transport_net_one_destination (AT)
Transport_net_two_destinations (AT)
Guided_transporter_based_system
AGV_System_MISSION_example (AT)
Buffer_or_storage_system
Parallel_buffer
Buffer_in_the_shunt_by_Bosch
Sequential_buffer
Buffered_connection_by_Bosch
Warehouse
Storage_MISSION_example (AT)
Warehouse (AT)
Storage
Human_ressource
Operator
Customer
Management
Enterprise_management
Headquarter
Headquarter (AT)
Manufacturing_management
Manufacturing_process_plan_management
Production_quantity_management
Lot_size_management
Components_list_management
Manufacturing_management_centre
Transportation_management
Route_planning_and_scheduling
Transportation_capacity_management
Transportation_management_centre
Storage_management
Inventory_list_management
Storage_management_centre
Human_ressource_management
Personal_pool_management
Worker_pool_by_Bosch
Shift_plan_management
Shift_plan_by_Bosch
Combined_enterprise_components_structure
Circular_construction_workplace_in_the_shunt
Circular_construction_workplace_in_the_shunt_by_Bosch
Circular_construction_workplace_in_the_main_circiut
Circular_construction_workplace_in_the_main_circiut_by_Bosch
Manufacturing_line
Linear_construction_by_Bosch
Manufacturing_cell
Simulation_support_utility
Monitoring
Statistical_monitoring (AT)
Graphical_monitoring

Legend

Classes signed by "AT" are Application Templates, the other are classes or specified but not implemented Application Templates

**Figure 4.5: Example for the Template library class structure**

The subclass Enterprise contains all Application Templates which describe *complete*. A combination of these the Templates can form a supply chain simulation scenarios. In difference to a complete enterprise the class Enterprise_component represents substructures of an enterprise, such as Manufacturing_system, Transport_system, Buffer_or_Storage_system, Human_resource and Management. They contain the Application Templates which are necessary to fulfil the manufacturing process.

Instead of the detailed description of a template, it is possible to describe a template by other templates. This allows a modular structure of the templates and of the components of the distributed simulation. For example, a transport system may be described by a routing planning system, a network simulation and a vehicle simulation. The hierarchical inheritance structure of the template library can support wizards to know the necessary elements of a more detailed template because on a higher level the structure can be described more generic (Figure 4.5), e.g. Transport system includes the part vehicle, for the derived subclass AGV system, the wizard can ask for a vehicle as part of the AGV system.

The simulation model or the other application can be built independently from the template, only securing the template description. Afterwards the simulation model can be connected with the template. But the linkage of a simulation model to a specific template has to follow some rules:

- similar behaviour of the simulation models for one template
  The simulation model linked to a template has to follow the template definition. So that all simulation models linked to a template, have a similar behaviour.
- satisfaction of simulation parameters
  The simulation parameters defined for the template have to be supported by the simulation model. On the other side, the parameters of the simulation have to be satisfied by the template parameter definition.
- satisfaction of exchange object classes
  A more difficult rule is the satisfaction of the exchange object class definition of a template. The interface definition of the simulation model must be mapped to the exchange object class definition of the template..
- satisfaction of connectivity elements
  It is necessary that each simulation model of a template supports the input and output relations of the template definition.
- subclass mechanism
  If the previous described rules can not be fulfilled for a new simulation model, then it is possible to create subclasses or siblings of the preferred template. Subclasses can be supplemented. Siblings can be defined new, excepting the definitions derived from their parent class.

The parameters of the simulation model are connected with attributes of the template. These attributes have to be marked as attributes relevant for simulation (e.g. for consistency checks). At the end, it is necessary that each parameter of the simulation model will have a clear relation to a template attribute.

shows some parameter of exemplary Application Templates. It is important that all subclasses inherit the root class attributes or their parent class attributes respectively. So, for example the Application Template “Machine” inherit the parameter ID and Name (from Root). Additional the own parameters are Processing_time, MTBF and MTTR.

| Application Template Class | Parameter Name | Type | Unit | Default Value | Exempl. Value | Description |
|---|---|---|---|---|---|---|
| Resource | | | | | | |
| | All subclasses inherit the root class attributes or parent class attributes respectively | | | | | |
| | ID | String | - | - | | Unique Identifier |
| | Name | String | - | - | Processing_line_part_A | Object name |
| | Description | String | - | - | | Short description |
| Machine | | | | | | |
| | Processing_time | String | h | - | TRIA(2; 3; 4) | manufacturing time per product entity |
| | MTBF | String | h | 3589 | - | mean time between failure (for the complete model) |
| | MTTR | String | h | 2.5 | 0 | mean time to repair (for the complete model) |
| Storage_MISSION_example (AT) | | | | | | |
| | Products_per_package | Integer | Nr | 1 | 5 | Number of Parts in a package (internal package station) |
| | Storage_capacity | Real | Nr | ∞ | 15 | Storage Capacity for Packages |
| | Storage_strategy | String | - | FIFO | FIFO | "FIFO", "LIFO", "Random" |

**Table 4.2: Parameter of exemplary Application Templates classes**

For the Application Template, it has to be decided which objects are necessary for the communication within the distributed-simulation-environment. So, each Application Template refers to one or more Exchange Objects from the common class structure in Figure 4.9. This principle of relation between Application Template and Exchange Objects is described in Figure 4.1.

In difference to the parameters of the Application templates the reference to exchange objects will not be transmitted or inherited. So, each Application Template has its own list of supported and linked Exchange objects.

Furthermore, it is necessary to pay attention to the output and input segments as well as the output and input objects. For example, an output segment of a processing line could be the output buffer. Within a simulation scenario this output segment will be associated with an input segment of an other simulation model (Figure 4.6) e.g. a transport system. The objects, which can pass the segments, have to be described also for output and input segments. Because each output or input segment support only a special number of Exchange Objects class. Additional the direction of information flow, such as Input or Output, is important. So, for example it is impossible to connect an output segment, which generates MISSION_manufacturing_orders, with an input segment of an other building block, which support only

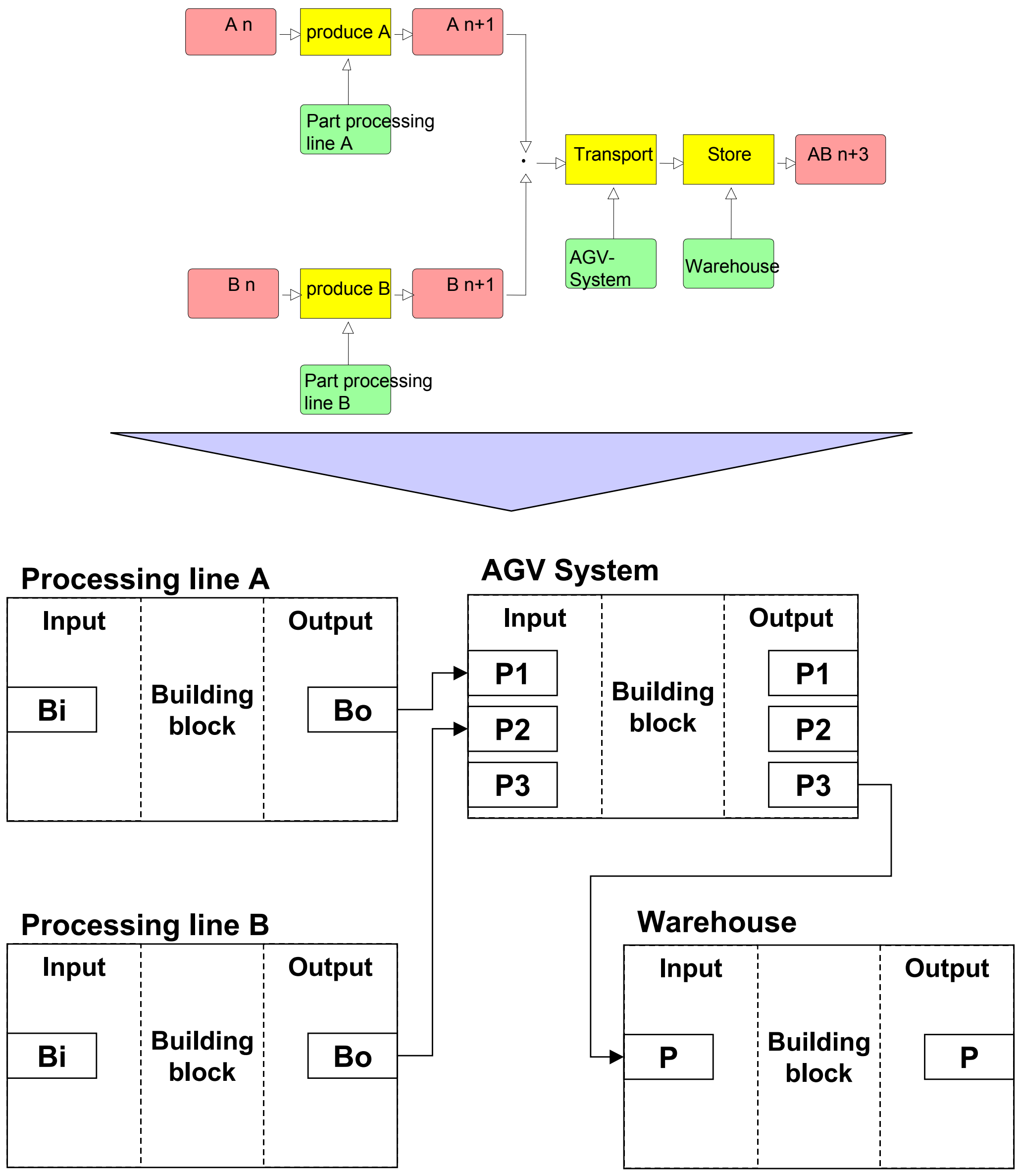


**Figure 4.6: View of the connections between the output and input segments in a simulation scenario**

incoming MISSION_transport_order. With this construction the Exchange Object will generates and publish right, but the input segment can't understand and process the incoming Exchange Object.

The segments and attributes are a part of the exchange object model and the connections between these objects are described within the federate configuration files. For an experiment, the common information model will not be changed. Only the federate configuration files have to be adapted to the experiment. The generation of the runtime environment of the simulation scenario is based on the previously generated federate configuration files. Each simulation model involved in the simulation scenario gets its own federate configuration file including all information necessary to configure the simulator interface for this simulation model.

The behaviour description of the template supports the retrieval of a suitable template and secures a similar content of the simulation models behind the template. The description can be done using a natural language but it seems much better if it can be done using a formal language based on XML. The advantage of a formal language is the possibility to load them directly into a simulator. That does not result necessarily in an automatic generation of simulation models, but it should provide a base to build such models faster.

The next step is the consistent combination of simulator, simulation model and federate configuration file.

When this step is completed, the HLA-RTI can be started using the previously generated FED file. Now the different components of the simulation scenario runtime environment will be executed. An Adapter to the RTI allows an easier connection of software tools. The adapter supports the retrieval of class and attribute information described within the federate configuration file. Furthermore it supports the association between class and attribute descriptions within the federate configuration file and the class and attribute names within the FED file. Therefore, it will be easy to retrieve information for an attribute name like value type, default values of attributes, etc.

An important point of the Template Library concept is the connection of the supported simulation models within a distributed simulation scenario. As said before, the Exchange Objects are the information carrier between the models. They transport the data from one model (Federate) to an other via predefined connectivity elements. These connectivity elements are associated with input or output segments. And in the simplest case one output segment of a upstream simulation model is connected with one input segment of an other downstream simulation model (1:1 connection). If there are no input segment restriction (no limited capacity for incoming Exchange Objects), the data exchange is possible without any problems.

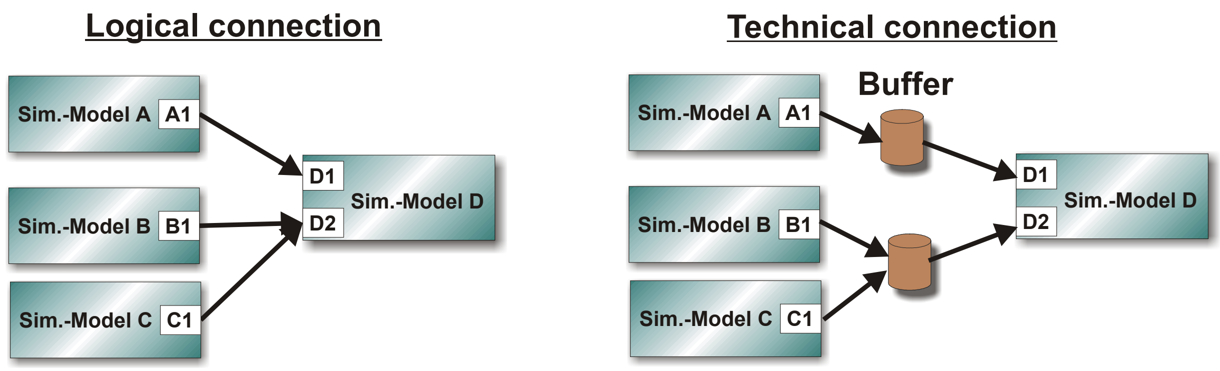


**Figure 4.7: Connection of Simulation models via a buffer (schematic)**

But what happens, if there are two (or more) output-segments linked to one input-segment (n:1 connection) or reversal (1:n connection)? Or, what happens, when the capacity of a segment for outgoing or incoming Exchange Objects is limited? Here a synchronisation of the exchange objects is necessary.

As a result of the discussion between Japanese and European project partners, Figure 4.7 illustrates schematically a solution in form of a *Buffer,* which was created to solve the synchronisation and bottleneck problems. A connection between simulation models ever goes via this Buffer element. The Buffer has a throughput capacity. The throughput capacity indicates how many Exchange Objects can be transferred by the buffer simultaneously. Depending on an internal counter value the Exchange Objects will stopped or passed. This counter information is handled on the RTI. For example, if the throughput capacity of the buffer (associated with PortIn 1 of Federate n+1) is restricted to 4 entities and the Federate n wants transfer 6 entities at the same time, then only 4 entities can pass the buffer. In this moment the internal counter has a value of 6 and therefore it is greater than the capacity. As result two entities have to wait in PortOut 1 of Federate n until the next possibility (time). If the 4 passed entities reach the PortIn1 of the following Federate n+1, then the internal counter is resetting to the new value 2. But if the 4 entities can’t pass the PortIn 1, e.g. because a machine has a malfunction and is blocked, then the entities have to wait until unblocking. In this case the internal value is setted back only, if the Federate n+1 isn't blocked any more. With the same principle, two or more Federates can be connected together via a buffer.

But note that the transfer process doesn’t need any simulation time. That means, during the transfer process the global simulation time doesn’t change. By this way, the consistency of scenario is reached and the bottleneck problem is solved.

Generally the Buffer is a Persistent Object and invisible for the user in the background. Persistent Exchange Objects are created only once in the simulation scenario and then it is existing during the complete simulation. Note that the existence is static, but not the information within these Objects.

### 4.1.6 Exchange Objects

An important issue within a distributed environment is the data, which can be exchanged between the different participants. The template library approach including the "exchange objects" defines a format for the data exchange. This format contains predefined elements to support the data exchange.

Furthermore, a flexible exchange protocol is required. A "federate configuration mechanism" has been developed which extents the "high level architecture" (HLA) to avoid additional programming for each "distributed scenario". The "federate configuration mechanism" is based on the generation of a "federate configuration file" (FCF) and the generation of the HLA-FED file at the same time from the same database. This is controlled by a simulation manager.

The automatic generation of a runtime simulation scenario requires the definition of the exchange object model for a template according to some data modelling rules. These rules must be fulfilled for each template. If the user adds an additional template, which does not fulfil the rules, he has to define a common exchange data model for a simulation scenario by himself. To save flexibility this is possible but not the normal way. In a normal situation, a new template will be integrated in the template library by a template library administrator. The administrator will secure the following rules:

- unique name space of classes
  Each name of an exchange object class of a template, has to be unique within the template library. This can be achieved by a class structure of exchange object classes.
- unique semantic of classes
  The semantics of an exchange object class must be described well and must be unique within the template library. Especially, it is required that two classes with the same or similar semantics do not exist.
- unique attribute definition, rely on inheritance
  Within one inheritance path structure the attribute definitions must be unique. This includes name, type and semantic.
- reliable class and attribute structure
  It is import that an implementation of a simulation model within a simulator can rely on the definitions of exchange object classes within a given template library. That means, the common exchange object class structure can only be supplemented. A reducing of classes or attributes as well as the redefinition of classes and attributes is forbidden. Expecting the class or attribute is not being used within any simulation model of the template library, then they can be destroyed or they can be redefined, if it is necessary.

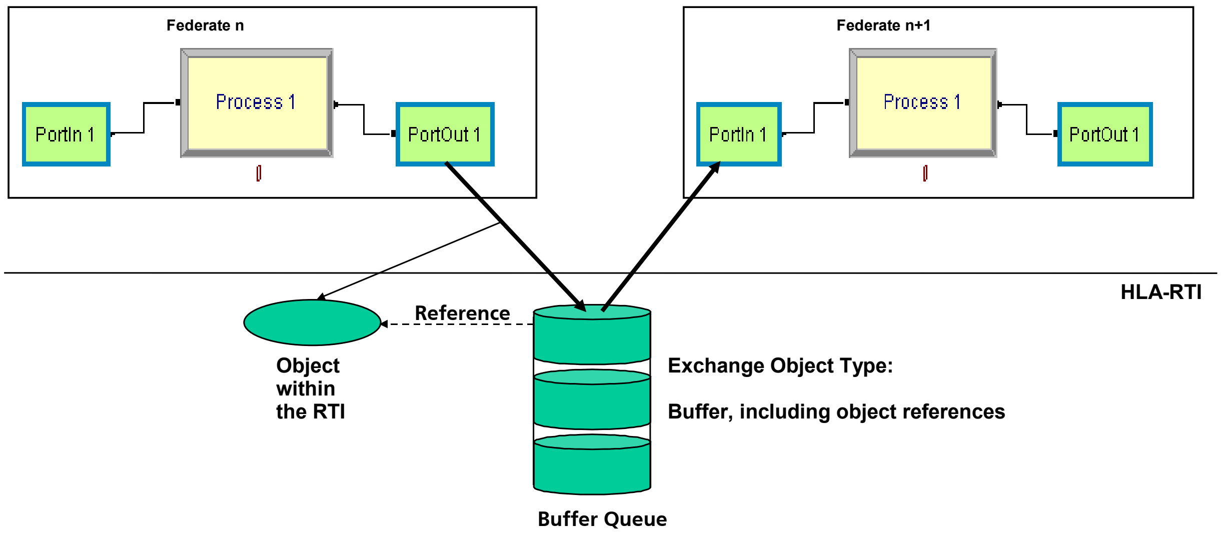


**Figure 4.8: Technical Principle of the Buffer Synchronisation (Bottleneck Problem)**

Like the Template Library the Exchange Objects also have a class structure (figure 4.9). Furthermore, it is not easy to decide if an class specific for exchange objects or for templates. In this context it is important to say, that Application Templates and Exchange Objects both are two different views on the same class structure.

Take for example a “car”-object in a scenario which simulates the metropolitan traffic situation. Within this scenario the “cars” have to exchanged between different simulation models within a distributed simulation environment. But it is also possible, that the “car” themselves is a simulation model, e.g. Mercedes A Class or a Volvo Truck. So, on the one hand the “car” object is an Application Template and on the other hand it is an Exchange Object. But during the MSE process the engineer (or developer) does not know which kind of object is the “car” object. This information gets only interesting, if the process shall be simulated within a distributed simulation environment. Therefore it is necessary in future, that there is existing only one general class structure. And “Application Templates” and “Exchange Objects” are two different views on the same class structure.

Only for technical reasons it was differ in two structures t the moment, each for Application Templates (Figure 4.5) and Exchange Objects (Figure 4.9).

Of course, the Exchange Object class structure has also a hierarchical inheritance structure. So, all subclasses inherit the attributes ID, Name and Description of the MISSION_Exchange_Object_class. In addition to these inherited attributes for example the MISSION_Product class has the attributes Product_type, Entity_ID, Total_cycle_time, Total_processing_time, Total_Transport_time and Total_waiting_time.

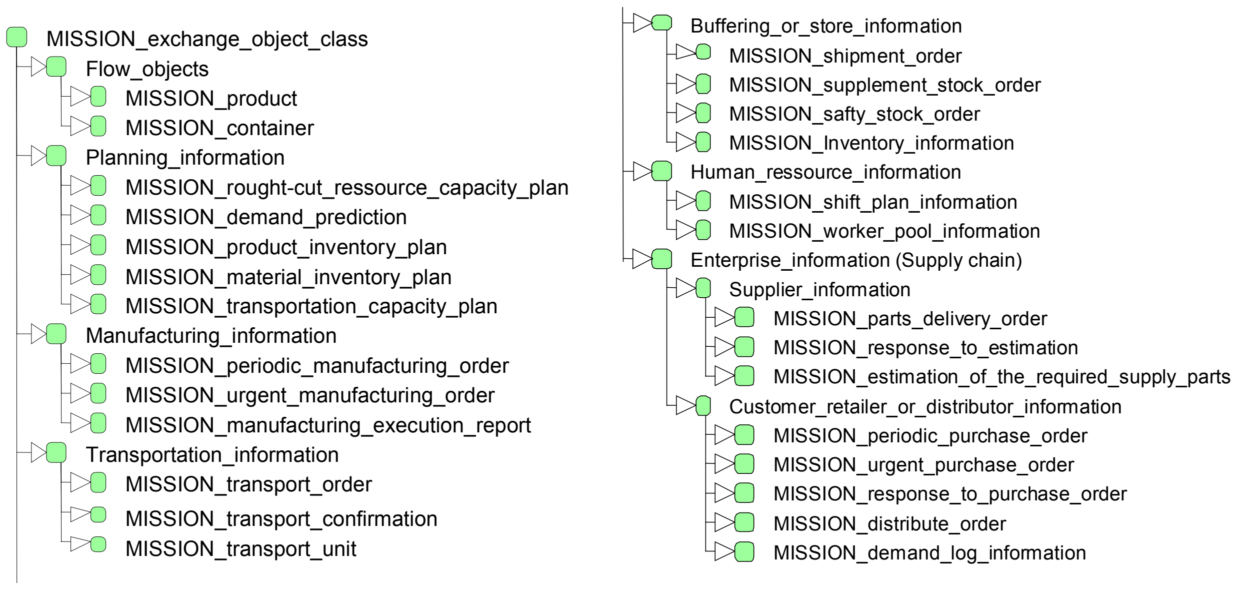

MISSION_exchange_object_class
Flow_objects
MISSION_product
MISSION_container
Planning_information
MISSION_rought-cut_ressource_capacity_plan
MISSION_demand_prediction
MISSION_product_inventory_plan
MISSION_material_inventory_plan
MISSION_transportation_capacity_plan
Manufacturing_information
MISSION_periodic_manufacturing_order
MISSION_urgent_manufacturing_order
MISSION_manufacturing_execution_report
Transportation_information
MISSION_transport_order
MISSION_transport_confirmation
MISSION_transport_unit
Buffering_or_store_information
MISSION_shipment_order
MISSION_supplement_stock_order
MISSION_safty_stock_order
MISSION_Inventory_information
Human_ressource_information
MISSION_shift_plan_information
MISSION_worker_pool_information
Enterprise_information (Supply chain)
Supplier_information
MISSION_parts_delivery_order
MISSION_response_to_estimation
MISSION_estimation_of_the_required_supply_parts
Customer_retailer_or_distributor_information
MISSION_periodic_purchase_order
MISSION_urgent_purchase_order
MISSION_response_to_purchase_order
MISSION_distribute_order
MISSION_demand_log_information

**Figure 4.9: Example for Exchange Object class structure**

In a very early version of exchange objects only a simple location attribute was used. But this is not enough to link exactly one segment of an simulation model with another. So for the root class of exchange objects the following attributes are defined:

- next location
  This attribute indicates the next federate.
- previous location
  This attribute indicates the previous federate.
- next segment
  This attribute indicates the segment within the next federate.
- previous segment
  This attribute indicates the segment within the previous federate.

These attributes are reflected within the federate configuration file in the description of the model linkage.

The inner structure of an Exchange Object consists of two kinds of attributes:

- **Technical description**
  The technical description includes attributes which are necessary to control the flow of such objects. One example for such attributes are the attribute Next_location or Previous_segment (the Previous_segment attribute is not implemented yet)
- **Description of content**
  The description of content includes application specific attributes. These attributes are defined by Type, Default Value and Description

The difference of this both types of attributes can be explained at different views of the location attributes. Figure 4.10 contains a distributed simulation scenario with five models. The models A and B symbolise suppliers and the models D and E are customers or retailers. The model C simulates a transport system which connects the models A, B, D and E which each other. In this configuration an entity (like a  product or package) should go from simulation model B to simulation model D via model C.

For the first transport step from model B to C the *technical* attributes “Next_location” and “Next_segment” of the product-entity are set to “Sim.-Model C” and “C2”. The simulation model B registers the product-entity at the RTI and free the ownership at the entity within the RTI, after model internal deleting the entity. Now, the product-entity will owned by the model C and created at segment C2. However where shall the Simulation model transport the entity? The model C simulates a transport system with different locations or destinations. There are exist 4 possible destinations (with name and ID)!

That’s why, here an additional Exchange Object with a “*description of content*”-attribute is necessarily. Therefore, the model B has to generate a MISSION_Transport_order for model C with the end-destination information of the product-entity (“Sim.-Model D”). The Exchange Object MISSION_Transport_order now contains two different kinds destination attributes. The first attributes are the *technical* attributes “Next_location” and “Next_segment”, which describe the destination for MISSION_Transport_order itself. Like at the product-entity they are set to “Sim.-Model C” and “C2”. (Note, that the MISSION_Transport_order inherits the attributes “Next_location” and “Next_segment” from the Root class.) Additional, the MISSION_Transport_order has the own “*description of content*”-attribute “Destination Name”. This attribute contains the end-destination information for the transport of the product-entity. It is set to “Sim.-Model D”. During the parameterisation process of the model C the information “Sim.-Model D” was associated with an internal path through the transport net. So, the model C can transport the product-entity directly from segment C2 to segment C3.

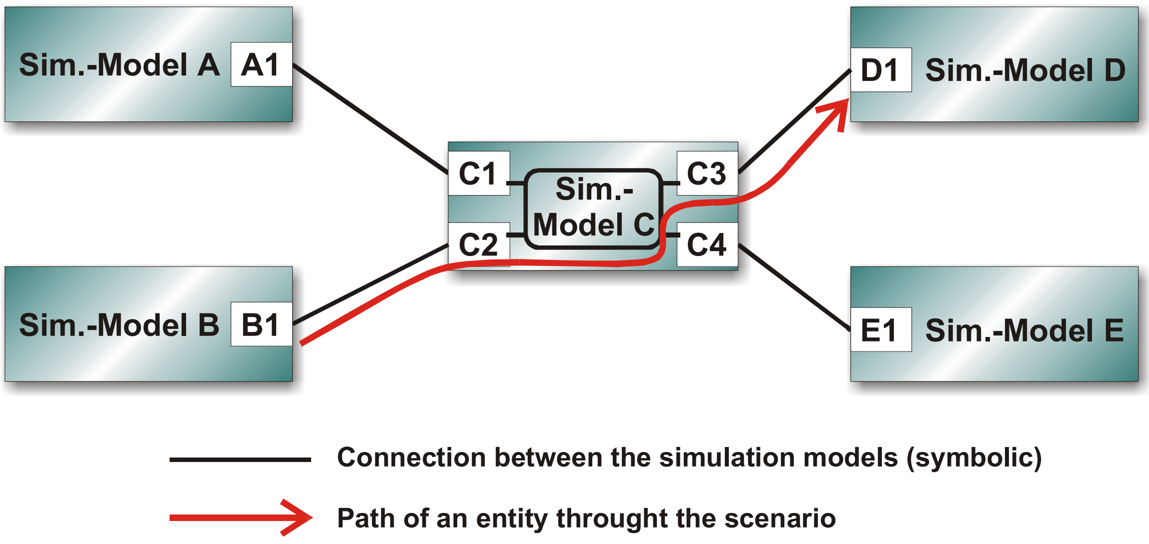

Sim.-Model A
A1
Sim.-Model B
B1
C1
C2
Sim.-Model C
C3
C4
D1
Sim.-Model D
E1
Sim.-Model E
Connection between the simulation models (symbolic)
Path of an entity throught the scenario

**Figure 4.10: Exemplary scenario scheme for describing the different meanings of location attributes**

### 4.1.7 The Template Editor

This chapter describes the functionality of the Template Library Editor implementations for the MSE process (TL-Editor MSE) and the MS process (TL-Editor MS).

#### *4.1.7.1 Technical Issues TL-Editor MSE*

For the connections between the TL-Editor and the Information Manager (IM), the Information Manager Adaptor (IM-Adaptor) is used. The IM-Adapter and the IM are communicating with each other over the TCP/IP protocol instead of using the MS-RTI, this is because the direct connection over TCP/IP is faster as the before used HLA RTI.

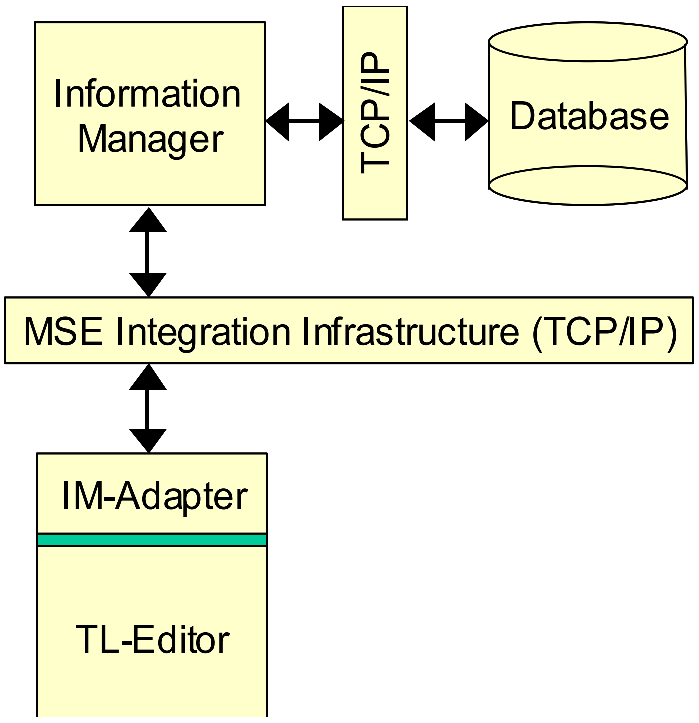


**Figure 4.11: Connection between IM and TL-Editor**

#### *4.1.7.2 Technical Issues TL-Editor MS*

The MS TL-Editor is connected with the Simulation Manager Server (SM-Server) via the Remote Method Invocation (RMI) method of Java. RMI has the advantage of security, multi-threading and object orientation. Especially because of the object oriented aspects a faster and more reliable implementation is possible.

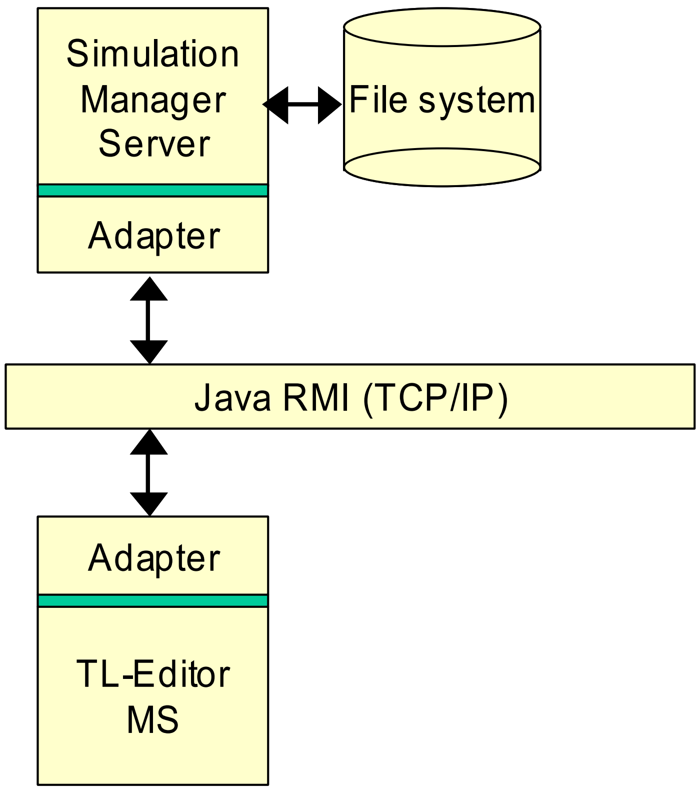


**Figure 4.12: Connection between TL-Editor (MS) and Simulation Manager**

### *4.1.7.3 Implementation Issues*

Like most of the other software developed for the Mission-Project the TL-Editor (MSE) and the TL-Editor (MSE) are implemented in Java, which allows a fast prototyping and the execution on different platforms and with small adaptations the execution in internet browsers.

### *4.1.7.4 Requirements*

#### *4.1.7.4.1 Hardware*

The minimum Hardware requirement for a practicable usage of the TL-Editors is a AMD or Pentium System with at least 233 MHz and 64 MB RAM.

For running the Simulation Manager Server on the same computer 128 MB RAM are recommended.

If the Information Manager and the database (Oracle) should run on the same computer then the system should have at least a CPU with 500 MHz and 256 MB RAM.

#### *4.1.7.4.2 Software*

The Information Manager Version 1.1 is required for running the TL-Editor (MSE). As for the IM it is needed to have a Java Runtime Environment 1.2 or greater installed. If you do not have a JRE than you can downloaded one without charge from Sun at http://java.sun.com/j2se/1.3/ or from IBM at http://www-106.ibm.com/developerworks/java/jdk/?dwzone=java for Windows 9X, NT, 2000 and several *nix Versions.

For the TL-Editor (MS) it is of course necessary to have Java installed. Additionally the Simulation Manager Server is needed (take a look at chapter 4.4 for installation and usage).

### 4.1.7.5 Installation (both Versions)

The distribution consists of one archive (tleditor-mse.zip) for the TL-Editor MSE and one archive (tleditor-ms.zip) for the TL-Editor MS which contains the following files:

| Filename | Directory | Description |
|---|---|---|
| tleditor.jar | ./lib | Archive containing the TL-Editor program files. |
| tleditor.bat | ./bin | Batch file for starting the TL-Editor. |
| tleditor-config.xml | ./config | A XML configuration file containing some necessary information for the execution. |
| tleditor-config.dtd | ./config | The document type definition for the configuration file. |
| ….jar | ./lib | Additional libraries needed by the TL-Editor. |

**Table 4.3: Content of the tleditor-mse.zip file**

| Filename | Directory | Description |
|---|---|---|
| tleditor.jar | ./lib | Archive containing the TL-Editor program files. |
| tleditor.bat | ./bin | Batch file for starting the TL-Editor. |
| tleditor-config.xml | ./config | A XML configuration file containing some necessary information for the execution. |
| tleditor-config.dtd | ./config | The document type definition for the configuration file. |
| ….jar | ./lib | Additional libraries needed by the TL-Editor. |

**Table 4.4: Content of the tleditor-ms.zip file**

For the installation it is only necessary to unzip these files into a any directory.

### 4.1.7.6 Configuring the Program

#### 4.1.7.6.1 TL-Editor (MSE)

The configuration file for the TL-Editor (MSE) is a simple XML file (tleditor-config.xml) which has to be edited by the user before he can run the program. The configuration file has the following format:

```
<Information_Manager>
      <Address>
            [address]
      </Address>
      <Port>
            [port]
      </Port>
</Information_Manager>
```

where

- address is the TCP/IP address or a name which can be resolved by a name server  where the IM is running at this time and
- port is the port where the IM is running at this time (normally it does not need to be changed).

A valid configuration file is for example:

```
<Information_Manager>
      <Address>
            um183.ipk.fhg.de
      </Address>
      <Port>
            5500
      </Port>
</Information_Manager>
```

The address and port have to be set to point to the computer where the IM is running.

##### *4.1.7.6.2 TL-Editor (MS)*

The configuration file for the TL-Editor (MS) is a simple XML file (tleditor-config.xml) which has to be edited by the user before he can run the program. The configuration file has the following format:

```
<Simulation_Manager_Server>
      <Address>
            [address]
      </Address>
      <Port>
            [port]
      </Port>
</Simulation_Manager_Server>
```

where

- address is the TCP/IP address or a name which can be resolved by a name server where the Simulation Manager Server is running at this time and
- port is the port where the Simulation Manager Server is running at this time (normally it does not need to be changed).

A valid configuration file is for example:

```
<Simulation_Manager_Server>
      <Address>
            um183.ipk.fhg.de
      </Address>
      <Port>
            1089
      </Port>
</Simulation_Manager_Server>
```

The address and port have to be set to point to the computer where the Simulation Manager Server is running.

### *4.1.7.7 Starting the Program*

#### *4.1.7.7.1 TL-Editor (MSE)*

The IM must be running on the computer defined in the IM section of the configuration file (see 4.1.7.6), before the TL-Editor (MSE) can be started. The program is invoked by double clicking on the tleditor.bat file. It creates a connection to the IM. If this fails the user will get a message and the program shuts down.

#### *4.1.7.7.2 TL-Editor (MS)*

The Simulation Manager Server must be running on the computer defined in the  "Simulation Manager Server" section of the configuration file (see 4.1.7.6) before the TL-Editor (MS) can be started. The program is invoked by double clicking on the tleditor.bat file. It creates a connection to the SM-Server. If this fails the user will get a message and the program shuts down.

Alternatively the Administration GUI can be used for starting the TL-Editor (see chapter 4.4).

### *4.1.7.8 Short Overview over the TL-Editors*

Both applications have a very simlilar look and feel, since the underlaying technique is quite the same. But some differences exists.

Both applications have a module to edit classes and attributes. The MS TL-Editor has an additional module to create application templates.

#### *4.1.7.8.1 The Attribute Editor Component (MSE & MS)*

The attribute editor displays the attributes of an instance or object with name, type and predefined value for classes and name, type and current value for instances. The user sees this information in the form of a table.

Changing the type of a defined attribute is not possible. Deleting of an attribute means to delete this recursively at all instances and subclasses. Changes of attributes are send immediately to the Information Manager or to the SM-Server.

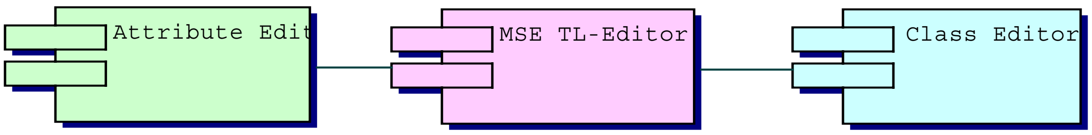


**Figure 4.14: Module structure of the TL-Editor (MSE)**

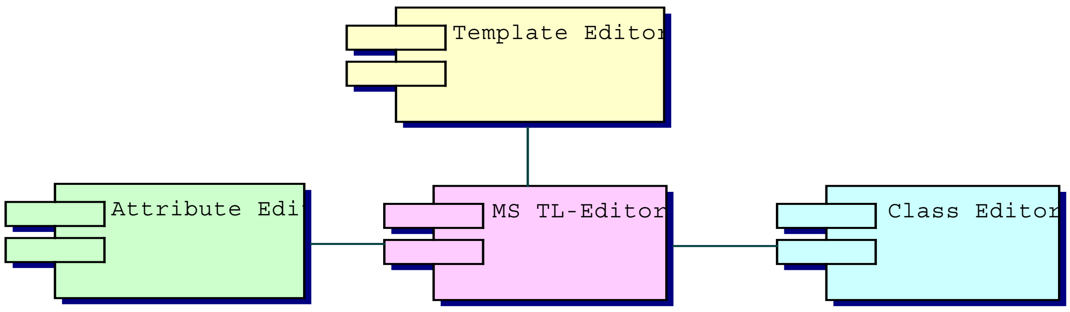


**Figure 4.15: Module structure of the TL-Editor (MS)**

The features of the attribute editor are:

- creating attributes (only for classes),
- changing value of attributes for instances,
- changing the default value of attributes for classes,
- deleting attributes (only for classes) and
- renaming attributes.

#### *4.1.7.8.2 The Class Editor Component (MSE & MS)*

The task of the class editor is to visualize the structure of a part of the classes contained in the Mission data model, especially the templates. With little modification this module (MSE TL-Editor) can display all classes stored in the Information Manager.

The class structure is loaded on demand in the background. But not the whole class structure is loaded since this would take for a big model with for example 1000 classes and 10000 instance a lot of time. For this, only the classes under the node which is selected are loaded (with a deep of 2 levels).

Changes of classes are send immediately to the Information Manager or the SM-Server.

The features of the class editor are:

- creating classes and instances,
- changing the inheritance of classes and instances,
- renaming instances,
- deleting classes and instances and
- drag and drop to move classes.

#### *4.1.7.8.3 TL-Editor (MSE)*

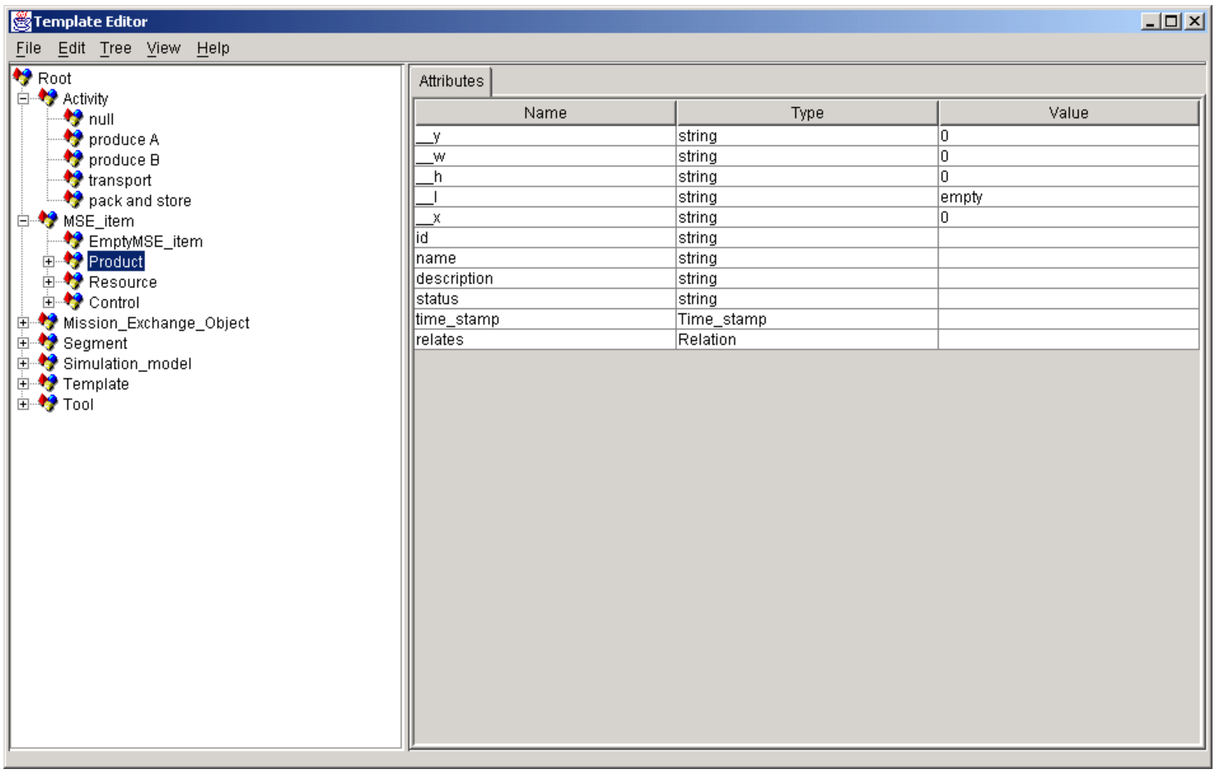
This TL-Editor allows to edit the classes and objects stored inside the Information Manager. This means that the user has the possibilities to edit the class structure, create and manage instances and attributes.

In interaction with the Manufacturing Process Design Agent (MPDA) or the Supply Chain Manager (SCA) the TL-Editor can be used as a source for creating instances.

The Application is divided into two parts, an attribute editor and a class editor.

An additional features of the class editor is “drag and drop” (from the TL-Editor to the SCA or MPDA).

There are no additional features of the attribute editor.



- Root
  - Activity
    - null
    - produce A
    - produce B
    - transport
    - pack and store
  - MSE_item
    - EmptyMSE_item
    - Product
    - Resource
    - Control
  - Mission_Exchange_Object
  - Segment
  - Simulation_model
  - Template
  - Tool

| Name | Type | Value |
|---|---|---|
| __y | string | 0 |
| __w | string | 0 |
| __h | string | 0 |
| __l | string | empty |
| __x | string | 0 |
| id | string | |
| name | string | |
| description | string | |
| status | string | |
| time_stamp | Time_stamp | |
| relates | Relation | |

**Figure 4.16 TL-Editor (MSE)**

The handling of the application is very easy. The application screen is split into two parts on the left side the class editor and on the other side the attribute editor. To manage classes and instances the popup menu which appears after a click with the right mouse button on a tree node can be used.

Also a popup menu for attributes exists. It appears after a click with the right mouse button into the attribute table. Attribute values can be changed by double clicking into the value column.

#### *4.1.7.8.4 TL-Editor (MS)*

This program allows the user to do anything necessary to create templates. But not only templates can be created. With the build in class and attribute editor it is possible to create classes and instances and to add attributes.

Like the TL-Editor (MSE) the window is split into two parts, with the class structure on the left side and on the right side the attributes of the classes and instances.

Additional features of the class editor:

- Drag and drop (from the TL-Editor to the SMPM or SMBBM) and
- creating of application templates.

Additional features of the attribute editor are:

- Changing the default value of predefined attributes.

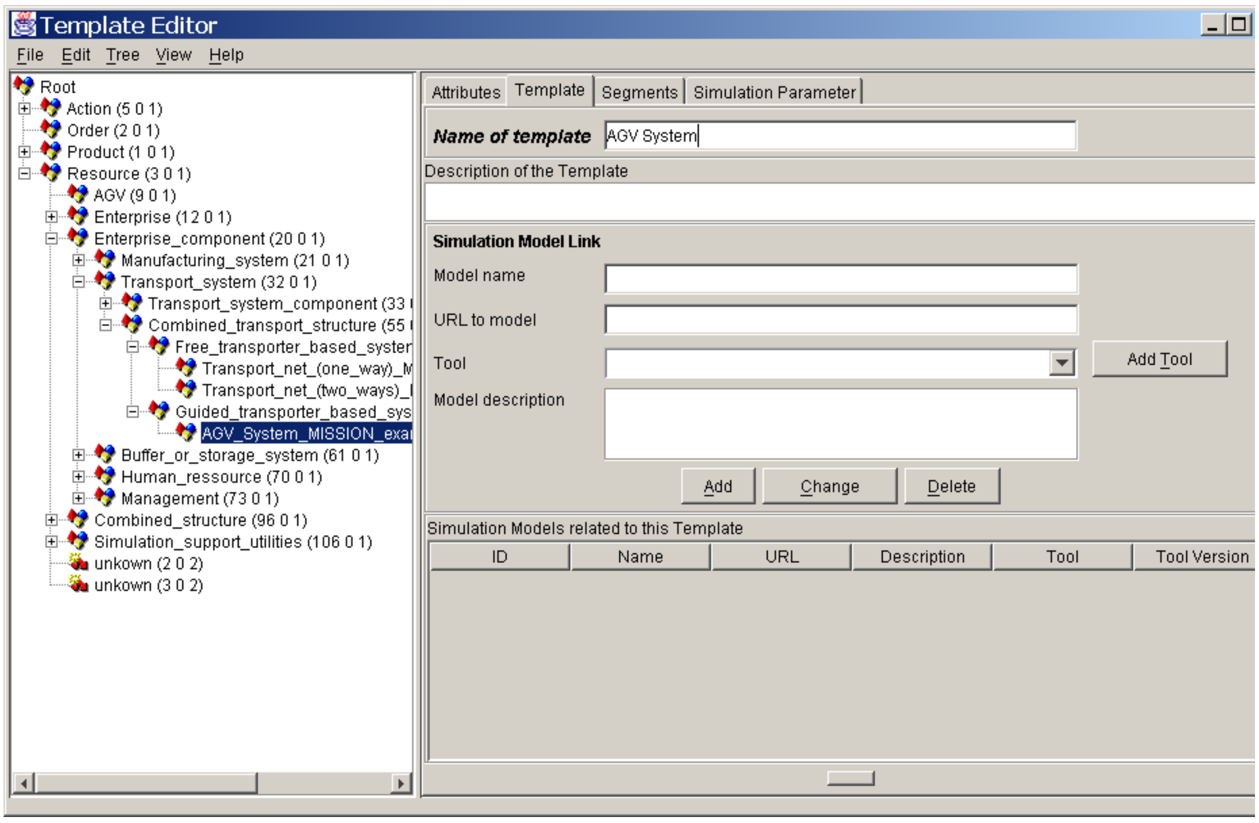

Template Editor
File Edit Tree View Help
Root
Action (5 0 1)
Order (2 0 1)
Product (1 0 1)
Resource (3 0 1)
AGV (9 0 1)
Enterprise (12 0 1)
Enterprise_component (20 0 1)
Manufacturing_system (21 0 1)
Transport_system (32 0 1)
Transport_system_component (33
Combined_transport_structure (55
Free_transporter_based_system
Transport_net_(one_way)_M
Transport_net_(two_ways)_
Guided_transporter_based_sys
AGV_System_MISSION_exa
Buffer_or_storage_system (61 0 1)
Human_ressource (70 0 1)
Management (73 0 1)
Combined_structure (96 0 1)
Simulation_support_utilities (106 0 1)
unkown (2 0 2)
unkown (3 0 2)
Attributes
Template
Segments
Simulation Parameter
Name of template
AGV System
Description of the Template
Simulation Model Link
Model name
URL to model
Tool
Model description
Add Tool
Add
Change
Delete
Simulation Models related to this Template
ID
Name
URL
Description
Tool
Tool Version

Figure 4.17: TL-Editor with template currently edited

#### *4.1.7.8.5 The Template Editor Component (only TL-Editor MS)*

This is the main module of the TL-Editor. It allows the user to create and manage templates. This component consists of several panels to add simulation models to the template, add tools for the simulation models, to link exchange objects with the template, to define input and output segments for the building block and to define parameters for the template.

The features of the TL-Editor component are:

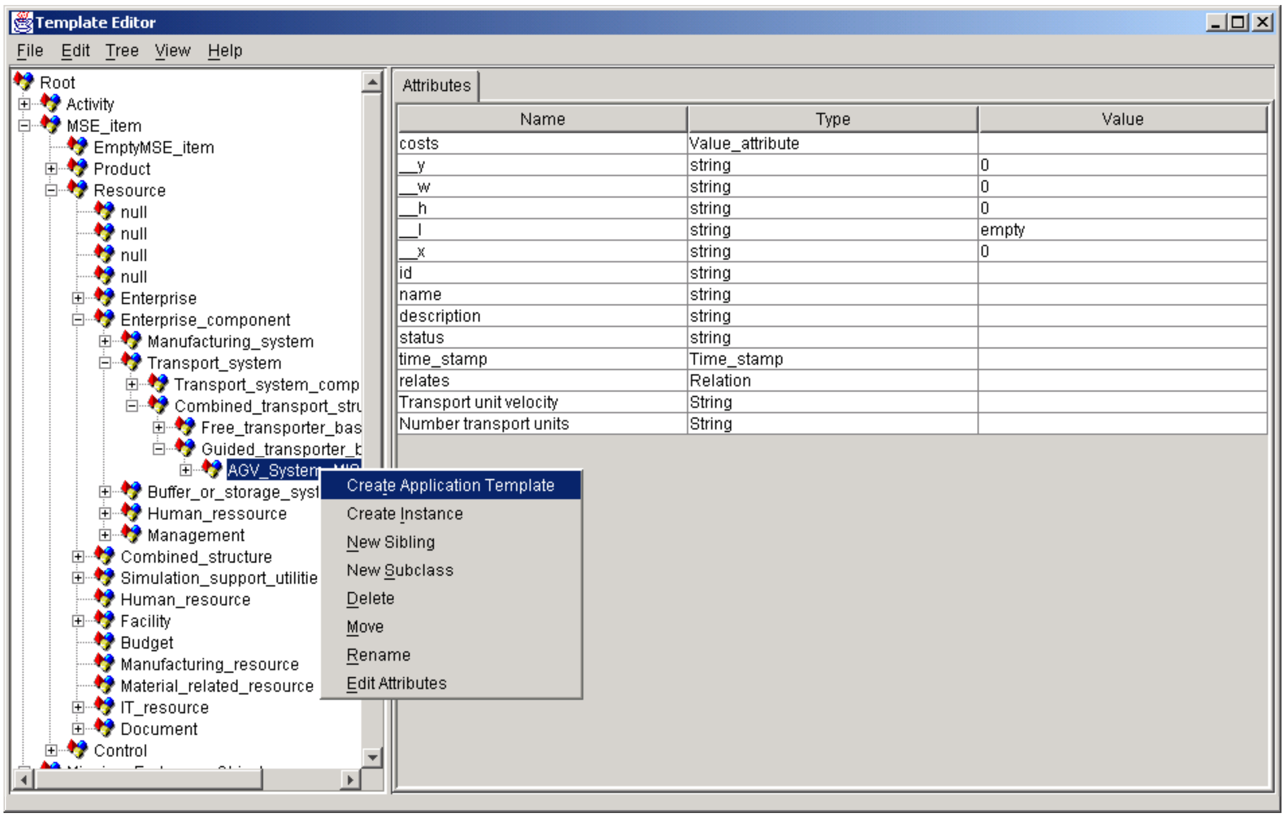
- Creating and adding simulation models to the template,
- creating tools for defined simulation models,
- linking templates with there exchange objects,
- defining segments for the template
- setting parameters for the simulation model.

## 4.1.8 Creating a Template Step by Step

### *4.1.8.1 Prerequisites*

Here follows an example for the creation of a template within the TL-Editor (MS). The Template which will be created is an AGV System.

We suppose that the system is in an executable state. This means that the Simulation Manager Server is running and the TL-Editor (MS) is started.



| Name | Type | Value |
|---|---|---|
| costs | Value_attribute | |
| __y | string | 0 |
| __w | string | 0 |
| __h | string | 0 |
| __l | string | empty |
| __x | string | 0 |
| id | string | |
| name | string | |
| description | string | |
| status | string | |
| time_stamp | Time_stamp | |
| relates | Relation | |
| Transport unit velocity | String | |
| Number transport units | String | |

**Figure 4.18: Selecting the parent class of the template**

### *4.1.8.2 Defining the Template*

The first step is to choose the parent class for the application template. In our case this is the class Guided_transporter_based_system. After a click with the right mouse button on this node a popup menu will appear. With the menu point “Create Application Template” the template will be created.

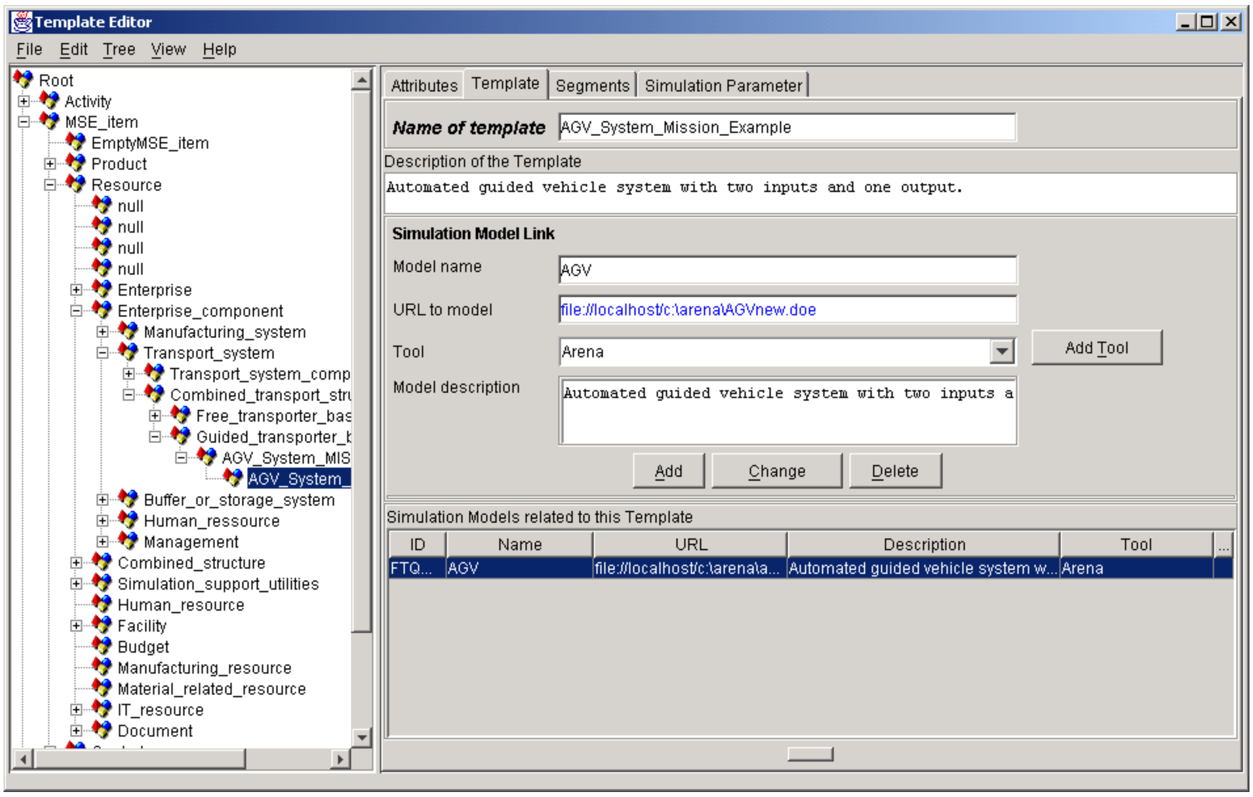
After selecting the fresh created node three tabs (Template, Segment and Simulation Parameter) are added to the “Attributes” tab. Each tab offers the possibility to add the necessary information for the usage of the template.

The first tab is the “Template” tab. The user can enter here the name and a description for the template. Since a template is related to one or several simulations these links have to be added. For each line the name of the model, the URL to the model, the used software tool and a description of the model can be entered (see Figure 4.19).

A table shows the already added links. If a tool is unknown in the system than it is possible to add that tool by using the “Add tool” button.

In the appearing dialog the name, version, manufacturer and a description of the tool can be entered (see Figure 4.20).

In this example a simulation tool called “Arena” with a version number of “4” and the manufacturer “Rockwell” is added.



**Name of template** AGV_System_Mission_Example

Description of the Template

Automated guided vehicle system with two inputs and one output.

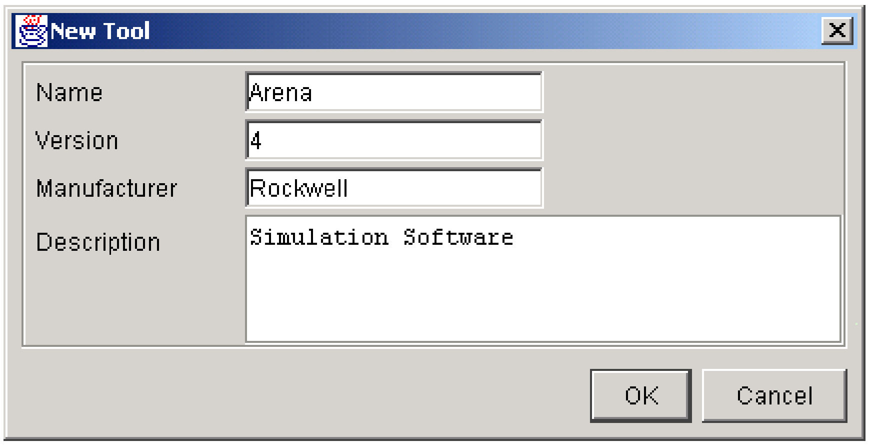
**Simulation Model Link**

| | |
|---|---|
| Model name | AGV |
| URL to model | file://localhost/c:\arena\AGVnew.doe |
| Tool | Arena |
| Model description | Automated guided vehicle system with two inputs a |

Simulation Models related to this Template

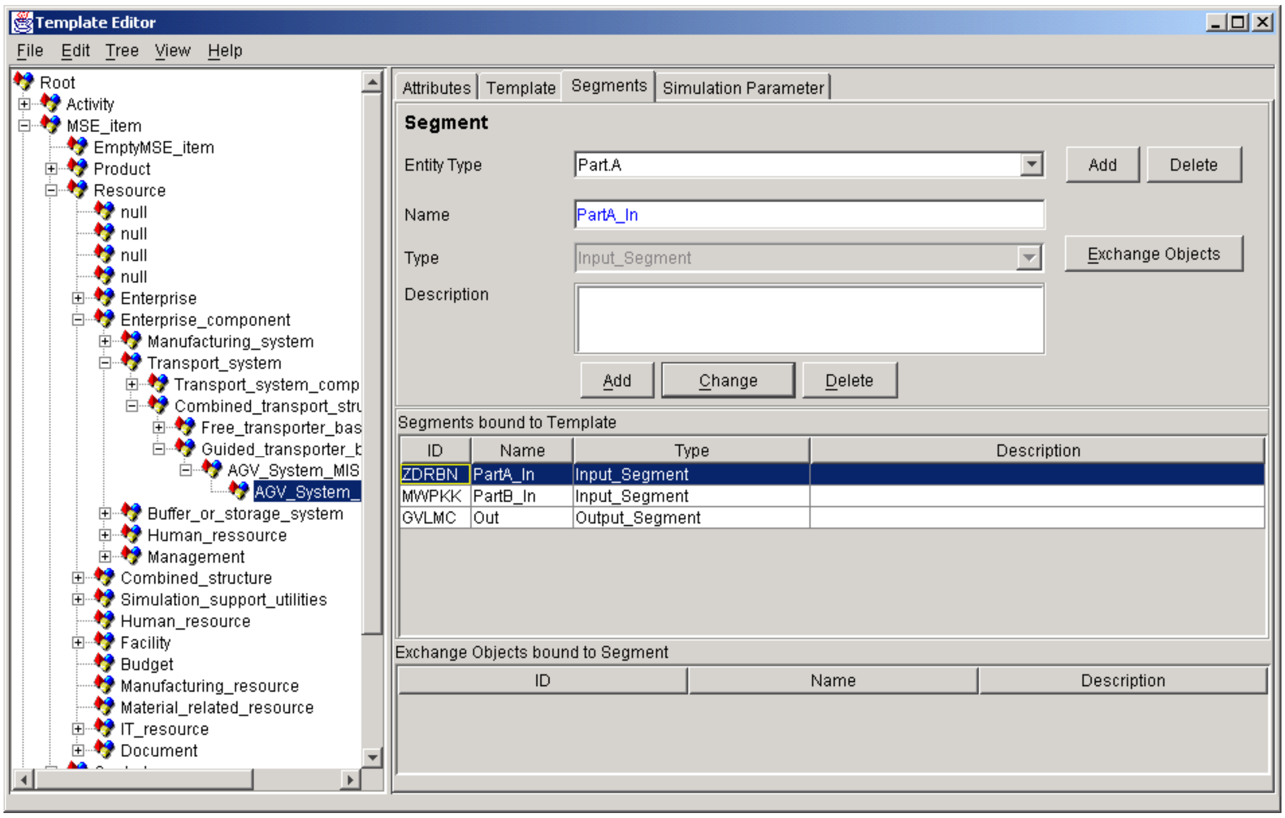
| ID | Name | URL | Description | Tool |
|---|---|---|---|---|
| FTQ... | AGV | file://localhost/c:\arena\a... | Automated guided vehicle system w... | Arena |

**Figure 4.19: The "Template" tab.**



| | |
|---|---|
| Name | Arena |
| Version | 4 |
| Manufacturer | Rockwell |
| Description | Simulation Software |

**Figure 4.20: New tool.**



Template Editor

File Edit Tree View Help

Root
- Activity
- MSE_item
  - EmptyMSE_item
  - Product
  - Resource
    - null
    - null
    - null
    - null
    - Enterprise
    - Enterprise_component
      - Manufacturing_system
      - Transport_system
        - Transport_system_comp
        - Combined_transport_stru
          - Free_transporter_bas
          - Guided_transporter_b
            - AGV_System_MIS
              - AGV_System_
      - Buffer_or_storage_system
      - Human_ressource
      - Management
    - Combined_structure
    - Simulation_support_utilities
    - Human_resource
    - Facility
    - Budget
    - Manufacturing_resource
    - Material_related_resource
    - IT_resource
    - Document

Attributes | Template | Segments | Simulation Parameter

**Segment**

Entity Type: PartA — Add — Delete

Name: PartA_In

Type: Input_Segment — Exchange Objects

Description:

Add — Change — Delete

Segments bound to Template

| ID | Name | Type | Description |
|---|---|---|---|
| ZDRBN | PartA_In | Input_Segment | |
| MWPKK | PartB_In | Input_Segment | |
| GVLMC | Out | Output_Segment | |

Exchange Objects bound to Segment

| ID | Name | Description |
|---|---|---|

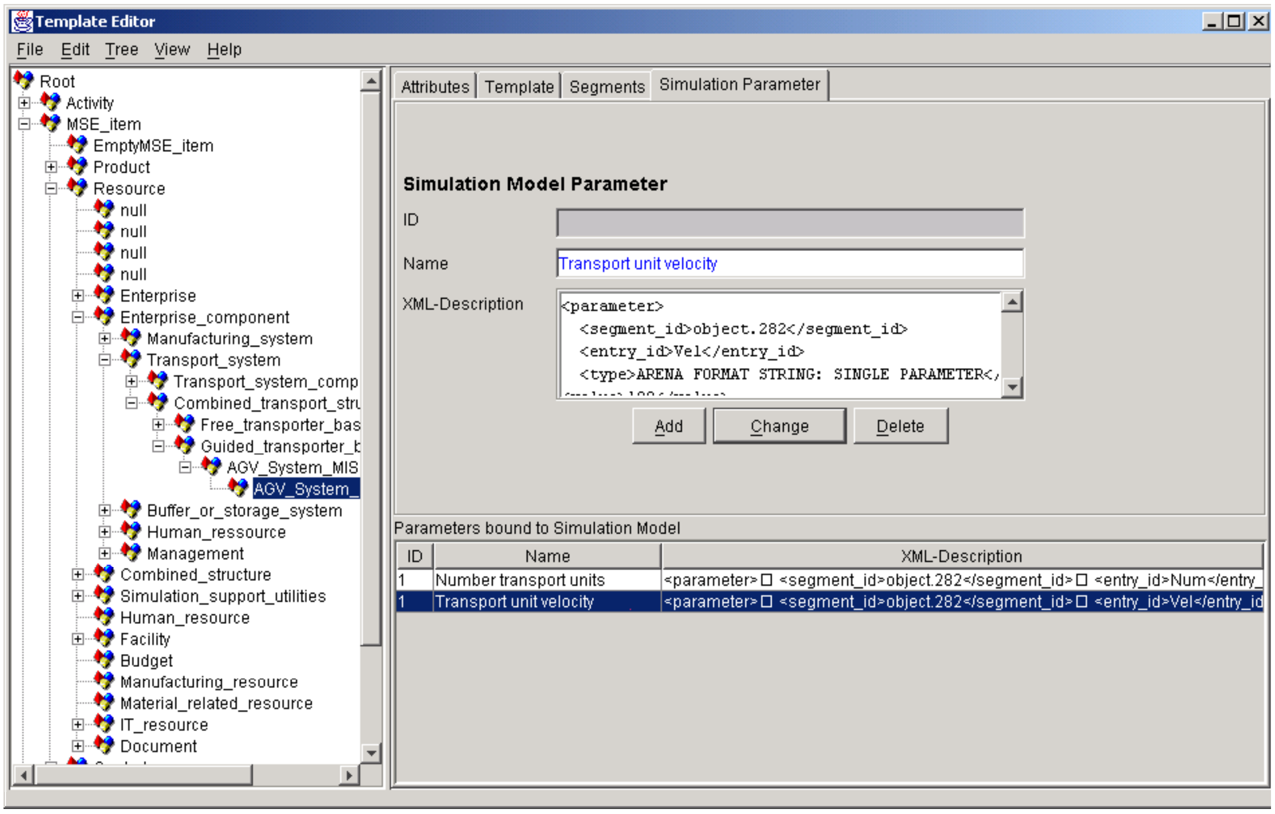
**Figure 4.21: The "Segments" tab.**

At the “Segments” tab (see Figure 4.21) the input and output ports (segments) are added to the template.

For each segment the entity type of the segment, the name, the type (input or output) and a description can be entered. In our example two input segments (PartA_In and PartB_In) and one output segment (Out) is added.

In the last step at the “Simulation Parameter” tab (see Figure 4.22), the attributes of the template are mapped to the simulation model parameters.

A model parameter consists of a name which has a direct relation the attribute of the template and a XML description for the generation of the Federation Configuration File. A table shows all entered Parameters.



**Simulation Model Parameter**

ID

Name: Transport unit velocity

XML-Description:

```
<parameter>
  <segment_id>object.282</segment_id>
  <entry_id>Vel</entry_id>
  <type>ARENA FORMAT STRING: SINGLE PARAMETER</
```

Parameters bound to Simulation Model

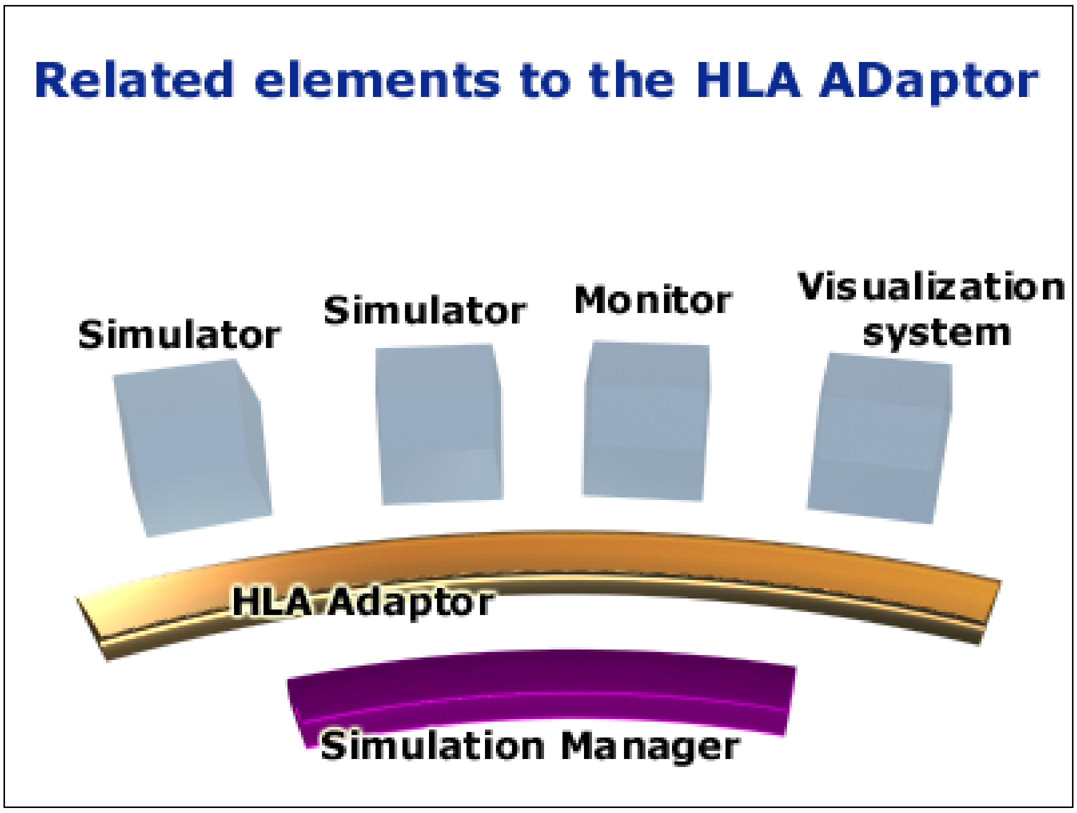
| ID | Name | XML-Description |
|---|---|---|
| 1 | Number transport units | <parameter>□ <segment_id>object.282</segment_id>□ <entry_id>Num</entry_ |
| 1 | Transport unit velocity | <parameter>□ <segment_id>object.282</segment_id>□ <entry_id>Vel</entry_id |

**Figure 4.22: The "Simulation Parameter" tab.**

After all information is entered the template can be used.

## 4.2 HLA Adapter

This important piece of software belongs to the MMP Kernel. It is involved in the MS process as the responsible to supply some basic functionalities to the distributed simulation. The HLA Adaptor is used by the simulators, as well as monitoring tools and/or visualization systems, to work together. It forms, with the Information Manager, the low level part of the MMP, that is, the MISSION user will never see it. The MISSION user will use the software that is connected to the Adaptor (for example the simulator) but there is no direct interaction between the user and the Adaptor. Instead of that, is the MISSION developer who will use the Adaptor functionalities in order to create interfaces to the higher level software.



To know about the HLA Adaptor is necessary to take a look first to the HLA, what it is, and why does it exist.

### 4.2.1 High Level Architecture (HLA)

#### *4.2.1.1 Introduction*

Simulation has played a key role in several different fields. Its relatively low cost has made it a very suitable tool for fields such as Electronics, Education, Manufacturing and many others. Included in this range is, of course, Defence. Simulating different environments has allowed military people to predict the possible consequences of their decisions over the battlefield, test new weapons and so on.

The Defence Modeling and Simulation Office (DMSO), the U.S. military branch exclusively dedicated to simulation, has played a key role in the development of new simulation techniques, applying them, in a first step, to their own simulations. The work of the DMSO in this field was such that, during the early 90s, the number of different military simulations available was countless. Wherever simulations could be applied, new simulation environments were developed to suit this new requirements.

And this was the main problem of the issue. These countless simulations were completely independent of each other. Even very closely related simulation programs such as a F-14 fighter simulation and a B-52 bomber simulation were impossible to interconnect. If there was a need to simulate a F-14 trying to shoot down a B-52, a completely new simulation would be developed, without the chance to just connect the existing ones. Model reusability was very close to zero.

Realising that this situation was completely incongruous, the DMSO started developing a new way to design simulations, a set of behaviour rules that every simulation should follow, so that they could be interconnected with little effort. The High Level Architecture (HLA) was thus born to allow simulations to work with each other, building a greater simulation. Additionally, HLA allows the small simulations to be run on different computers in a network, so that big simulations can be run over a number of not so powerful computers.

The HLA is a general purpose architecture for simulation reuse and interoperability. It was to support reuse and interoperability across the large numbers of different types of simulations developed and maintained by the DoD. The HLA Baseline Definition was completed on August 21, 1996. It was approved by the Under Secretary of Defense for Acquisition and Technology (USD(A&T)) as the standard technical architecture for all DoD simulations on September 10, 1996. The HLA was adopted as the Facility for Distributed Simulation Systems 1.0 by the Object Management Group (OMG) in November 1998. The HLA was approved as an open standard through the Institute of Electrical and Electronic Engineers (IEEE) - IEEE Standard 1516 - in September 2000. The HLA MOA was signed and approved in Nov. 2000.

#### *4.2.1.2 Objects and Interactions*

If several simulations are being run simultaneously, it’s obvious that results from one simulation will affect the behaviour of others. For instance, during the simulation of the flight of a new airplane, two federates could be the fuselage and the air around it. The air movement is influenced by the shape of the fuselage, and this movement may produce different pressures on different parts of the fuselage. The fuselage simulation must be able to get this data and simulate its influence, maybe even microscopically altering the shape of the fuselage. If this is the case, the air simulation must take into account these changes, and so on.

Data exchange is achieved within HLA using entities. These entities are like “packages” that contain attributes and values for these attributes, and can be divided in two groups:

- *Objects*: These objects are entities that endure. This is, if a federate creates a given object, another federate can later ask for that object and get its current value.
- *Interactions*: Interactions are entities that disappear just after being created. These entities “stay alive” just enough time for all interested federates to be able to get information from them, and then disappear. Federates that have an interest in an interaction that happened in the past are unable to retrieve this data.

Both objects and interactions are based on an object-oriented structure. That is, there may be several different classes of objects (and interactions), which share (inherit) some common characteristics. For example, there could be a class “Vehicle” with some defined attributes, such as “speed”. We could also have two different classes “Bus” and “Truck” which inherit from “Vehicle”. This means that both “Bus” and “Truck” will have a “speed” attribute, as was defined by “Vehicle”, but they could also add other new attributes, such as “passengers” for “Bus” or “cargo” for “Truck”. The class structure ends always in all classes inheriting, directly or indirectly, from a common class (whose name is ObjectRoot or InteractionRoot). This class structure is defined within a text file called Federation Execution Data (FED) file, although this file only defines inheritance relationships and attributes each new class adds, without reference to attribute types, default values and so on.

There are some inheritance rules that must be satisfied:

- An object class cannot inherit from an interaction class or viceversa. There are separate object and interactions class trees. This is completely logical, taking into account the great differences between them.
- The HLA supports only single inheritance. That is, any object or interaction may only have one parent class. If we had a “Tank” object, it could not inherit from both “Vehicle” and “Military resource”. This, in itself, is a limitation. But multiple inheritance presents heavy problems that are, sometimes, too severe to be compensated by the ability to inherit from several classes.

There are additional limitations. Whenever a function requires a class as a parameter (e.g. when we want to create a new object of the given class), the class name cannot be directly used. Classes, as well as attributes and objects, are associated to numeric identifiers, known as “handles”. To obtain a handle, a different function must be called, in order to let us know which one is associated to each class name. Further, the class name must be fully qualified. That is, it is not possible to just use “Bus” as a class name, but all the inheritance tree must be provided, separating the different classes with a dot. Assuming that “Vehicle” inherited directly from “ObjectRoot”, then the fully qualified name would be “ObjectRoot.Vehicle.Bus”

#### *4.2.1.3 Attribute Updates*

From the definition of objects and interactions, it can be seen that the only entities that can be updated are objects. Interactions get an update as soon as they have been created, and are deleted just afterwards. To update an object, there must be a way to identify which object is to be updated. Objects are identified in two ways: a name and a handle (similar to objects classes). Although for humans using the name would be easier, most of the functions that work with object instances use the handle instead, so that federates must call a given function to associate the object name to the handle (as happened with object classes). Also, object attributes are also referenced by handle instead of by name.

With respect to object updates, the HLA defines two additional concepts:

- *Publishing*[1]: For a federate to be able to update an object, it must “publish” the given attributes of the class. Also, a federate must publish at least one attribute of the class to be able to “register” (create) instances of that class.
- *Subscribing*[1]: A federate that subscribes to an attribute of a class is informed of changes to that attribute on any instance of the class. That is, if a federate subscribes to the attribute “speed” of class “Vehicle”, it will be informed whenever any car, be it “car1”, “car2” or “car3” brakes or accelerates. Again, any federate that subscribes to at least one attribute of a class will be informed whenever an object of that class is created.

These two concepts are very closely related, as federates that publish an attribute of a class will be informed whether or not there are any federates that subscribe to attributes of that class.

There is an additional requirement to be met by a federate wishing to update an object. Apart from publishing the class attributes it wants to update, it must “own” these attributes of the instance. By default, a federate owns all attributes of the objects it has registered. But if it wants to update an attribute of an object it has not created, it must negotiate the ownership acquisition. There are several ways to do this, resulting in quite complex ownership management. Not only seizing ownership is complicated, but also divesting ownership, as one of the most important rules of HLA is that all attributes must be owned at every moment by one and only one federate.

This requirement for all attributes to be owned at every moment comes from one of the rules of the HLA: the RTI (this is, the related software) does not store objects within it. If a federate wants to read the values of an object, the RTI will not directly provide them. It will ask the federate in charge of each attribute and these federates will inform of the current value.

#### *4.2.1.4 Deleting Objects*

There is a moment in every simulation when an object is no longer needed. For instance, if a car producing factory is being simulated, car objects would not be needed once they are sent to the customers. At this point, these objects should be deleted. But, which federate is responsible for this deletion?

To avoid this ambiguity, an additional attribute is defined for all objects. This attribute is included in ObjectRoot (in fact, it’s the only attribute this class defines), and is called privilegeToDelete[2]. If a federate wants to delete an object, it must own this attribute. This brings another problems. How can this federate be sure that all federates which own attributes have ended their updates? This issue must be solved by the federation designer.

[1] Subscribing and publishing is also applied to interactions. Only federates that publish the given class of interaction can send them, and only subscribing federates can receive them.

[2] privilegeToDelete is the attribute name used by the DMSO RTI. Other Runtime Infrastructures, such as the pitch RTI use an attribute called privilegeToDeleteObject. This causes simulations prepared for one of them to fail when running on the other one.

### *4.2.1.5 Time Management*

Up to this point, the way to exchange data between federates has been described. But, just as important as data exchange is time management. Events must happen in the correct succession, or things may go completely wrong. This is most important in distributed environments, where computers may have different capacities, there may be a network latency, and so on. For instance, let us assume a complex and useful federation that allows us to simulate football matches with a 100% accuracy.

Now, the final match is getting closer, and we want to bet a lot of money in that game. We start the simulation and let it go for 90 minutes. Let's suppose that 10 seconds before the match ends, the score is 1-0, and the visitor team is attacking. The "player federate" shoots and the "referee federate" decides that it's time to end the game just before the ball gets to goal, thus ending the match with a 1-0. It's very important that the "end game" event is processed before the "goal" event. Otherwise, we would see a tie of 1-1, and we would bet our money to a tie in the real game, instead of the victory of the home team.

This example, although very simple and far from manufacturing simulations, lets us see the importance of order in the events of a simulation.

The High Level Architecture solves this problem using two additional concepts:

- *Time regulating federates*: These federates regulate the passage of time. For the "simulation clock" to advance, all time regulating federates must be ready to advance it.
- *Time constrained federates*: These federates have their passage of time completely regulated. This is, they cannot run faster than the federation, although the federation can run ahead of them.

These two choices are completely independent of each other, so that there are four possibilities:

- *Neither time regulating nor time constrained federates*: For these federates, the "federation clock" means nothing. They cannot affect it, and they are not affected by it. They are completely independent. Still, such federates may have problems if they run too fast and a slower federate generates an event in the former federates' past.
- *Only time regulating federates*: These kind of federates only regulate the passage of time. Still, they must make sure that other non-regulating federates are not too far behind the federation's time.
- *Only time constrained federates*: In contrast to the previous ones, these federates do not affect time, but are affected by it. For example, they could be visualisation systems, which should not run faster than the simulation (they would show incorrect results, as simulators have not reached that point) but it's not important if the simulation runs ahead of them (as they can continue showing results after the simulation has ended).
- *Time regulating and time constrained federates*: These federates are perfectly synchronised with the federation. Their internal clock is, with a very small difference, the federation clock.

Each one of these configurations has its advantages and disadvantages, and are proper for certain federate behaviours.

### *4.2.1.6 The Ambassadors*

For the above mechanisms to work properly, there must be a way for the federate to communicate with the RTI, and viceversa. This communication is achieved via two interfaces known as the ambassadors: the RTI ambassador and the Federate ambassador.

The RTI ambassador is the way for the federate to access the RTI. It contains functions to perform all of the tasks defined in the documentation of DMSO, such as registering objects, updating values, sending interactions, regulate time, and so on. This ambassador is provided by the RTI developer.

The Federate ambassador, on the other hand, is the interface for the RTI to inform the federate of changes as for example, if an object has been updated or deleted, if time advances have been granted, ownership management... This ambassador is not developed by default, and each federate must implement its own, depending on its behaviour.

Usually, when programming in C++ or Java, these two ambassadors are two separate objects, the RTI ambassador is already defined, and the other one is undefined or empty (that is, the Federate ambassador receives calls from the RTI, but ignores them). One of the most important tasks when designing a federate resides in the design of the corresponding Federate ambassador.

#### *4.2.1.7 Implementation Issues*

Although, as it has been seen during the last sections, HLA is a very useful architecture, which allows us to create distributed simulation environments, it has some drawbacks, mainly related to ease of programming :

- *The RTI does not store data*: This is one of the most important rules of HLA. Created objects are always maintained by federates, so the software developers must take care of this issue. Some of the functions that are to be developed within the Federate ambassador are related to object requests.
- *Classes and attributes are referenced by handle*: The federate has to maintain a table with the relationships of name to handle for classes and attributes, or keep calling the conversion function each time, which can be very time consuming. Besides, the HLA specification states that class and attribute handles may change each time the federation is executed.
- *The Federate ambassador has to be implemented*: It is mandatory that the Federate ambassador be implemented for the federate to work. The main problem is that it has some low-level functionality that the user may find quite hard not only to implement, but even to understand.
- *The FED file only defines the class inheritance tree and attribute names for the classes*: If the FED file is directly edited (it's a plain text file), it can be easily seen that it only defines which child classes inherit from each class, and the name of the new attributes for each child class. Other data, such as default values or attribute types is missing.

#### *4.2.1.8 Glossary*

The HLA defines several important concepts that will be used throughout this document:

- *Federate*: A federate represents one small simulation that is grouped within the more complex one.
- *Federation*: This represents the group of several small simulations working together.
- *Federation execution*: This is one execution of one complete federation (ie. one run of the whole simulation).
- *Runtime Infrastructure (RTI)*: A general purpose software package that is necessary for running a federation. A free version can be obtained from the Software Development Center (SDC) of DMSO.
- *Federation Object Model (FOM)*: This is the object model describing the data exchanged between federates of the same federation.
- *RTI ambassador*: An interface that receives commands from the federates to be passed to the RTI. It is provided by the developer of the RTI.
- *Federate ambassador*: An interface that is used by the RTI to notify the federates that something has happened. Each federate must provide its implementation of this ambassador, as each federate will have individual behaviour in response to that notification.

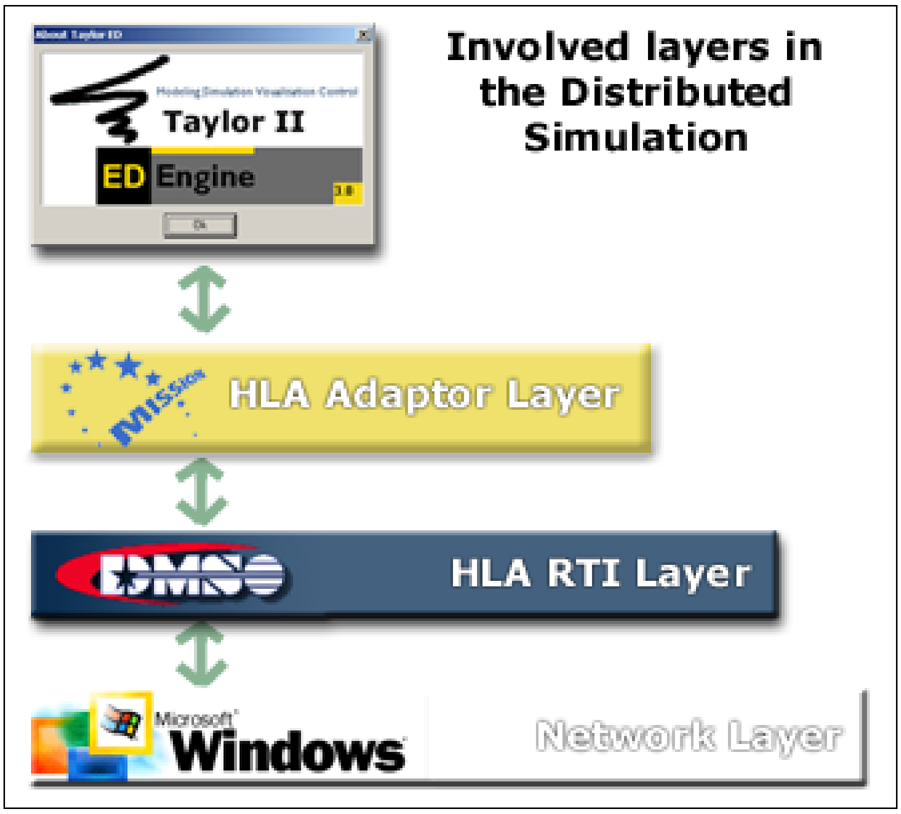
### 4.2.2 Description of the HLA Adaptor

#### *4.2.2.1 Introduction*

To address these HLA implementation issues, an adaptative layer has been developed. This layer, which is called the HLA adaptor, is located between the federate and the HLA, allowing federates to be programmed much more easily. This way, most of the difficulty to program the HLA is already performed by the HLA adaptor, without the user knowing (nor needing to know) what's really going on.

As there are Runtime Infrastructures available for a wide range of computers (Solaris, Windows, Linux...), the HLA adaptor has been developed in Java, in order to allow fast and direct reusability, without having to change source code or recompile.

The idea of the HLA adaptor is to provide the user with an easier way to program the HLA. In the process, new concepts have been introduced, while existing ones have been removed or updated.



#### *4.2.2.2 Objects and Messages*

Within HLA, the two types of entities that existed were objects and interactions. Objects are maintained within the HLA adaptor, although with some important changes that will later be seen. Inheritance within the object classes has also been maintained, whilst the concept of the handles has been completely hidden from the user. Classes, object instances and attributes are always referenced by name. It is the HLA adaptor which converts them to the handles required by the HLA.

Further, the FED file is now complemented with an XML file, whose name is Federate Configuration (FC) file, which allows each federate to dynamically discover the class inheritance structure, attribute types, default values and other configuration data that may be required.

Interactions have been removed and replaced by messages. This messages are, in themselves, interactions with the following attributes:

- *Message ID*: An identification number used to keep track of all the messages sent.

- *Message Type*: This attribute separates the different messages by subject. This way, messages with different purposes can be processed separately.
- *Time*: Identifies the federation time when this message is to be sent. If the receiving federate has not started time control or the time is negative, the message will be sent automatically.
- *Recipient*: The federate to whom this message is being sent.
- *Data*: This field depends on the Message Type field.
- *Source*: Identifies the origin of this message, so that responses (should they be necessary) can be correctly sent

#### 4.2.2.3 Attribute Updates

The concept of publishing that existed within HLA has been eliminated. Every federate is able to update object attributes at will. Still, subscription has been maintained, so that federates only receive attribute update messages they want to be kept informed about.

The adaptor stores the situation of all objects that have been created within HLA. Whenever any federate wants to know the state of one object, it's not the federate, but the adaptor that answers its call. This greatly simplifies the federates, as they don't have to cope with this problem. Besides, it represents a major change in ownership, which has also been updated.

Ownership of an attribute within HLA meant two things: that the federate could update the attribute, and that it was responsible for maintaining its value if other federate asked for it. This was the main reason why attributes had to be owned by one federate at every moment.

With the HLA adaptor, federates are no longer responsible for attribute maintenance. So, there's no need to avoid unowned attributes. Unowned attributes cannot be updated, but they are maintained by the adaptor. When a federate wants to stop owning an attribute, its adaptor accepts the call and if the federate tries to update the attribute, the adaptor prevents it from doing so. But, in fact, the adaptor does not release ownership of the attribute until another federate asks for it. This situation has been represented by a new ownership state: the "not owned" state. This means that the attribute is not owned (it cannot be updated), but the adaptor is still responsible for its maintenance (it still owns the attribute within HLA). Note that this new state is strictly internal to the adaptor. There is no way for the federate to know its "not owned" attributes, as they are seen as "unowned" for it.

The different possibilities are shown in the following figure. It represents the three possible states and how the federate and the HLA see the attribute.

This new state allows an easier attribute ownership divestiture and seizing. Any federate can stop owning an attribute at any time, without the need to find another federate willing to take the responsibility. And any federate can try to seize ownership of an attribute at any time, although it will succeed only if the attribute is not owned by any federate (that is, all federates have it "unowned", apart from one, who has it as "not owned"). Its adaptor will then take care of the negotiation required to transfer ownership.

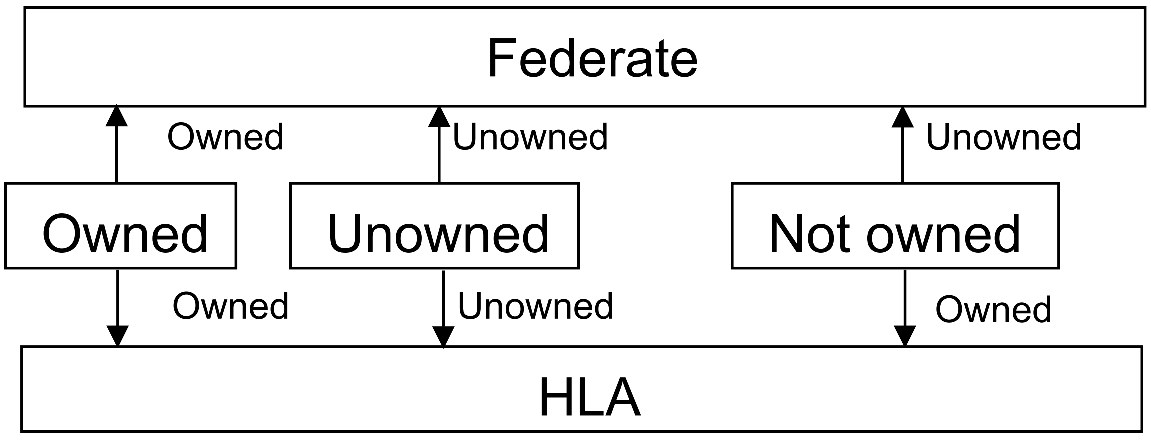


**Figure 4.23:** Attribute ownership - Different possibilities.

Another possibility has been added, which is a "forced ownership seizing". This is used when one federate needs to own an attribute, and cannot wait for that attribute to become "not owned". This case corresponds to one of the methods HLA had to transfer ownership. Its use is highly discouraged, because if the other federate owned the attribute, that means it wants to update it. It could be used, for example, when a federate owns an attribute and won't be able to release it (e.g. a hung federate, as long as the adaptor still works).

#### *4.2.2.4 Deleting Objects*

Object deletion has also changed. The privilegeToDelete (or privilegeToDeleteObject) method has been discarded. Now, for a federate to be able to delete an object, it must own all of its attributes before calling the delete function.

In fact, the only adaptor (not federate) which is able to delete an object is the one that created it. The "privilegeToDelete" attribute (removed from federate's view, but still in the HLA) is always owned by the object creator. When another federate tries to delete the object (we assume that it already owns every attribute of that object), what happens is that a system message is broadcast to every adaptor to delete the object. Only one of them recognises that the deletion is its responsibility, and deletes the object.

This way, it's impossible that an object be deleted when there are federates updating it. Still, the federate itself should make sure that no federates are interested in reading it too.

#### *4.2.2.5 Time Management*

For the HLA adaptor, "time regulation" and "time constraining" have been grouped in the "time control". This reduces the HLA four possibilities of time management to only three:

- *Not time controlling*: These federates are not interested in time control. Still, this represents a disadvantage, because they will receive messages and object updates as soon as they are sent, and not when they are supposed to be received (which depends on the federation clock). This is the default state for all federates, and some of them will remain in this state (for example, the federates connected to the MSE Runtime Infrastructure).
- *Time controlling*: In contrast, these federates are controlled by, and control, the federation clock. Messages and object updates are received when they are to be received (as stated in their respective calls).
- *Visualization mode*: These federates are not interested in controlling time, but they still want to receive events when they are scheduled to happen, as opposed to "not time controlling" federates.

### 4.2.3 The *A*mbassadors

In the HLA there were two ambassadors (RTI ambassador and Federate ambassador). In this approach, there are two objects that can be associated to this ambassadors, which are the adaptor in itself (associated to the RTI ambassador) and the callback object and the event queue (associated to the Federate ambassador).

The change of the RTI ambassador to the adaptor is quite clear: instead of using the RTI functions directly, the interface that is used to access the HLA is the adaptor, and its functions.

However the other change is more fundamental. Within HLA, each action had a different function related to it within the Federate ambassador. Now, in the HLA adaptor, the main actions (or events, are they known) are represented by objects, which are stored in an events queue. These objects can be of three types: deletion event, creation event and update event objects. In object oriented environments, these three classes inherit from an event class, allowing them to be more easily retrieved and identified. These messages include an identifying code (which is used to know which class the object belongs to), an identifying string to ease event visualization and several event-specific data, such as event time, related object, etc[1]

[1] For more information on the different message types and formats, refer to D13

Further, whenever a new event is added to the queue, a callback is sent to the defined callback object. This object only has one function, which is to inform the federate that a new event has been added. Its implementation in the federate is not mandatory as the federate could just keep checking the event queue. This is another difference from HLA, because in HLA, an empty or default Federate Ambassador prevented the federate from working.

#### 4.2.3.1 Addressing HLA Implementation Issues

With the described structure, the HLA implementation issues disadvantages have been addressed as follows:

- *The adaptor stores data*: Now, there's no need for federates to store all objects created within HLA. At any time, they can ask their corresponding adaptors for the updated value.
- *Classes and attributes are referenced by name*: This was one of the main problems of HLA. Now, with the adaptor discovering all name-to-handle relationships, and storing them within internal tables, there's no need for handles to be used in this cases. Object handles are still maintained, so that object instances can be referenced by either object name or RTI handle.
- *A mechanism similar to the federate ambassador still has to be implemented*: As it was explained in chapter 4.2.5, the federate ambassador has been replaced by a callback object and an events queue. Although there's still a need for federate specific functions to be implemented, there's more freedom in their implementation.
- *The FC file provides additional information*: The Federate Configuration file includes all of the information contained in the FED file, as well as other data which is used to reduce the effort in implementing interfaces to simulation models. Now, simulation models can be easily parametrized and reused, as well as connected to each other.

**Comparison Table between the HLA RTI and the HLA Adaptor**

| | **HLA RTI** | **HLA Adaptor** |
|---|---|---|
| Ease of Programming | *LOW* | *HIGH* |
| Multiplatorm software | *YES* | *YES* |
| Aditional configuration files | *NO* | *YES* |
| Time management modes | *4* | *3* |
| Data storing | *NO* | *YES* |
| Object referencing | *BY HANDLE* | *BY NAME* |

## 4.3 Simulation Tool Interfaces

### 4.3.1 Federate Configuration File

#### *4.3.1.1 Definition of FCF*

Federation configuration file consists of two main parts. The first part describes the technical information like the “host name” and the “port” address. This part is required by the HLA adapter. The second part (content of the tag “simulation”) is related to the content of the simulation scenario. The usage of the second part of the FC file will be described here. In the following, the term “FC file” will be used instead of the them “second part of the FC file”.

The FC file is used to pre-run configuration of the interface and the simulation model. The FC file is fed and evaluated after the connection to the HLA adapter is established and before the simulation starts. The following tags are carried out and the following configuration steps are performed:

```
<simulation>
        <<< root >>>
        <<< administration data >>>
        <<< object class definition >>
        <<< persistent object definition>>>
        <<< model linkage >>>
        <<< description of entities >>

        <<< model element parameter >>
</simulation>
```

**Figure 4.24: Structure of the second part of FC file**

#### *4.3.1.2 Start Element "Root"*

The FCF includes a class structure of the exchange object classes in relation to the class structure within the FED file. The root "tag" indicates the highest level of the class structure. The syntax of this tag is described below.

The tag has an identifier to indicate the id of the class. This id is used to refer to this tag as a parent class.

```
<root id=  />
```

**Figure 4.25 The tag "root"**

```
<admin_data
    model_id="Buffer_A"
    lookahead="0"
    simulator="ARENA"
    model_path="D:\anna\user\Arena\Models\Model10new.doe"
    simulation_time="2000"
/>
```

**Figure 4.26: Example of the tag "admin_data"**

### *4.3.1.3 Administration Data*

Each simulation model within a simulation scenario needs an unique identification. So, each FC File needs some administration data e.g.

- The attribute “model_id” is stored in the interface. During the simulation run, the “location” attribute of each entity, that will be registered within the HLA bus, will be set to this value.
- The lookahead within the HLA bus is set to the value of the attribute “lookahead”
- A simulation tool is started corresponding to the value of the attribute “simulator”
- A model with the path “model_path” is opened
- In the model, the length of simulation run is set to the value of the attribute “simulation_time”. If the attribute is not provided, the simulation length is set to infinite.

The model identification can be used within the FCF to indicate the ownership of an object, the subscription of attributes and to define the linkage between the different simulation models.

The administration data is identified by the tag "admin_data" and the parameters "model_id" and "lookahead".

### *4.3.1.4 Object Class Definition*

#### *4.3.1.4.1 Description*

The FC file includes a description of classes and attributes in relation to the FED file. The tag <exchange_class> is used to describe a class of objects which are exchangeable between different

```
<exchange_class id="part_a" name="part_a">
        <parent><exchange_class_reference ref="root"/></parent>
        <attribute>
            <name>location</name>
            <type>STRING</type>
        </attribute>
        <attribute>
            <name>segment_ID</name>
            <type>STRING</type>
        </attribute>
</exchange_class>
```

**Figure 4.27: Example of the tag "exchange_class"**

simulation models. Attribute types and values can be described here. Only the subscribed classes are described within the FC file related to one simulation model. The following structure illustrates the description of one class:

The parameter "id" of the tag <exchange_class> is used to refer to this tag as a parent class. Additional the parameter "name" indicates the name of this class. This name must be described as class name within the FED file.

The tag <parent> includes a reference of the parent class. Unfortunately at the moment the notations of XLink and XPointer for XML are still in the standardisation process. So they are not yet available. Instead of them a tag is used to indicate such reference. The tag <exchange_class_reference> has an special parameter what refers to the parameter "id" of a class.

The tag <scope> allows a informal description of the class and their attributes.

The tag <attribute> is used to describe a set of attributes. The names (<name>) of these attributes must be described within the FED file. Additional to the FED file the value type (<type>) and a default value (<value>) can be described for the attribute.

All the class attributes with name (tag "name"), type (tag "type") and optionally default value (tag "value"). Attributes "location" and "segment_ID" have a technical purpose and are required for each exchange class; for this attributes, no default value can be provided. The class information is stored within the interface.

#### *4.3.1.4.2 Example*

The example shows the configuration of the subscription of attributes of the class "part_a". The attribute "color" has a default value because the value may not be set by any federate. Only the tags "name" and "type" are required. The other tags of the attribute description are optional.

```
<root id= "root" />
<exchange_class  id=part_a, name="Part.a">
    <parent> <exchange_class_reference ref= root />        </parent>
    <attribute>
              <name> location               </name>
              <type>  STRING                </type>
    </attribute>
    <attribute>
              <name> status                 </name>
              <type>  STRING                </type>
    </attribute>
    <attribute>
              <name> cycle                  </name>
              <type>  STRING                </type>
    </attribute>
    <attribute>
              <name> color                  </name>
              <type>  STRING                </type>
              <value> white                 </value>
    </attribute>
</exchange_class>
```

**Figure 4.28: Example of configuration of subscription attributes**

### *4.3.1.5 Persistent Object Definition*

#### *4.3.1.5.1 Description*

Persistent Objects are static during one execution of an simulation scenario. The owner of the persistent object has to create the object within the RTI.

If the generation of objects at starting time is required  (maybe static objects), then the exchange object definition can be used. The tag <segment_id> describes a segment of the simulation model. The tag <entry_id> describes a data field for the value of the attribute (<attribute>) described below. The attribute is inherited from the parent class so the type information can be fond there.

```
<persistent_object  id=, name=, owner=/>
        <parent>        <exchange_class_reference ref= />   </parent>
        <segment_id>                                        </segment_id>
        <entry_id>                                          </entry_id>

        <attribute>
                <name>                                      </name>
                <scope>                                     </scope>
                <value>                                     </value>
        </attribute>
</persistent_object >
```

**Figure 4.29: Example of the tag "persistent object definition"**

#### *4.3.1.5.2 Example*

A warehouse has a finite capacity. If the maximal capacity is reached then the warehouse port is closed.

```
Segment of FCF for Warehouse:
<persistent_object  id=01, name=Warehouse_port, owner=Warehouse/>
    <parent> <exchange_class_reference ref= port/>                    </parent>
    <scope> "gives the status of the warehouse port (open, close)"    </scope>
    <segment_id>      P                                               </segment_id>
    <entry_id>        STATUS                                          </entry_id>
    <attribute>
            <name> status                           </name>
            <value> open                            </value>
    </attribute>
</ persistent_object>
```

**Figure 4.30: Warehouse capacity**

### *4.3.1.6 Model Linkage*

#### *4.3.1.6.1 Description*

Each simulation model has a number of input and output segments. It is necessary that these segments have an unique id. It is possible to combine the simulation model id with the segment id. So it is only necessary that the segment id is unique within one simulation model. It is not necessary that it is unique within the whole simulation scenario.

```
<model_linkage segment_id1="Out" simulation_model_id2="Storage" segment_id2="In"/>
```

**Figure 4.31: Example of the tag "model_linkage"**

The description of the physical connection of two simulation models requires the information which segment of the first model is connected to which segment of the second model. The id of the first model is known because the FCF was created for this model. So only the segment id is necessary (parameter: segment_id1). For the second model both is necessary the model id (parameter: simulation_model_id2) and the segment id (parameter: segment_id2).

Each link is stored within the interface with all indicated attributes

#### *4.3.1.6.2 Example "Simple MISSION Factory"*

The FCF-Example of the linkage based on the Simple MISSION Factory (Figure 4.32).

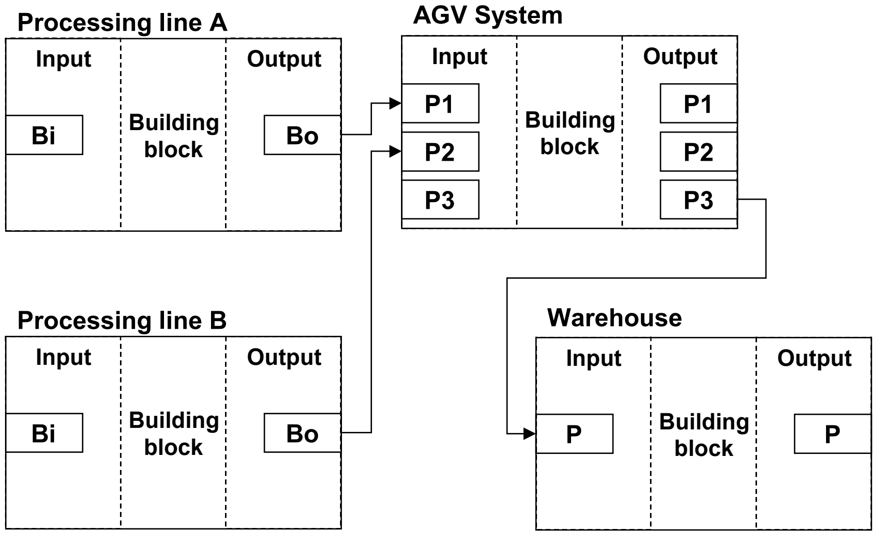


**Figure 4.32: Example for model linkage I**

FCF for PART A PROCESSING LINE:

```
<simulation>
    <<< root >>>
    <admin_data model_id= "PART A PROCESSING LINE", lookahead=10/>
    <<< object class definition >>>
    <<< object definition >>>
    <model_linkage segment_id1= Bo,
              simulation_model_id2= "AGV System",
              segment_id2= P1 /> ...
</simulation>
```

FCF for Warehouse:

```
<simulation>
    <<< root >>>
    <admin_data model_id= " Warehouse",  lookahead=1/>
    <<< object class definition >>>
    <<< object definition >>> ...
</simulation>
```

FCF for PART B PROCESSING LINE:

```
<simulation>
    <<< root >>>
    <admin_data model_id= "PART A PROCESSING LINE", lookahead=10/>
    <<< object class definition >>>
    <<< object definition >>>
    <model_linkage segment_id1= Bo,
              simulation_model_id2= "AGV System",
              segment_id2= P2 /> ...
</simulation>
```

FCF for AGV System:

```
<simulation>
    <<< root >>>
    <admin_data model_id= "AGV System", lookahead=10/>
    <<< object class definition >>>
    <<< object definition >>>
    <model_linkage segment_id1= P3,
              simulation_model_id2= "Warehouse",
              segment_id2= P /> ...
</simulation>
```

### *4.3.1.6.3 Example "Two Successors"*

An other example of linkage shall demonstrate one “two successors” Figure 4.33).

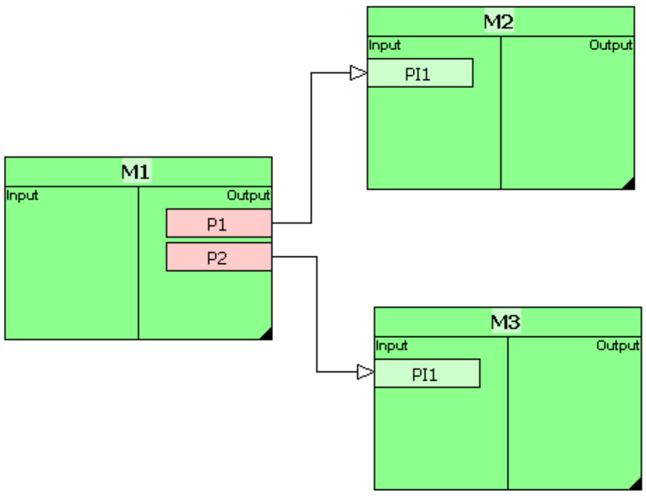


**Figure 4.33: Example for model linkage II**

FCF for M1:

```
<simulation>
    <<< root >>>
    <admin_data model_id= "M1", lookahead=10/>
    <<< object class definition >>>
    <<< object definition >>>
    <model_linkage segment_id1= P1,
              simulation_model_id2= "M2",
              segment_id2= PI1 />
    <model_linkage segment_id1= P2,
              simulation_model_id2= "M3",
              segment_id2= PI1 /> ...
</simulation>
```

FCF for M2:

```
<simulation>
    <<< root >>>
    <admin_data model_id= "M2", lookahead=15/>
    <<< object class definition >>>
    <<< object definition >>> ...
</simulation>
```

FCF for M3:

```
<simulation>
    <<< root >>>
    <admin_data model_id= "M3", lookahead=17/>
    <<< object class definition >>>
    <<< object definition >>> ...
</simulation>
```

### 4.3.1.7 Description of Entities

#### 4.3.1.7.1 Description

One possibility of a template is the parameterisation of the entities. For example we have two processing lines. The two lines have the same simulation model. Only the processing time and the entity types are different. The description of such an entity requires the parent class and a relation to an internal representation of the entity within the simulation model (related_id). This can be the id of a creation segment or the id of an object template.

```
<entity type="part_a">
          <related_id>Part.A</related_id>

</entity>
```

**Figure 4.34: Example of the tag "entity_type"**

For each stored class with the HLA name which matches to the value of the attribute "type", the model specific name contained in the tag "related_id" is provided. In this way, the mapping of the model names to the HLA names can be carried out.

#### *4.3.1.7.2 Example*

Segment of FCF for PART A PROCESSING LINE:

```
<entity type= "part.a">
            <related_id>        Part.A                          </related_id>
</entity>
```

Segment of FCF for PART B PROCESSING LINE:

```
<entity type= "part.b">
            <related_id>        Part.B                          </related_id>
</entity>
```

### *4.3.1.8 Model Element Parameter*

#### *4.3.1.8.1 Description*

Similar to the description of entities it is also necessary to describe model parameters (<parameter>), e.g. processing times, capacities, etc. In the simulation model, the parameter with the name “entry_id” of the module with the id “segment_id”.

```
<parameter>
        <segment_id>object.282</segment_id>
        <entry_id>Vel</entry_id>
        <type>ARENA FORMAT STRING: SINGLE PARAMETER</type>
        <value>20</value>
</parameter>
```

**Figure 4.35: Example of the tag "parameter"**

#### *4.3.1.8.2 Example*

Segment of FCF for PART A PROCESSING LINE:

```
<parameter>
    <segment_id>    "Working station"                       </segment_id>
    <entry_id>              DELAY                           </entry_id>
    <type>   "ARENA FORMAT STRING"                          </type>
    <value>  "HoursToBaseTime(Triangular(.5,1,1.5)),,VA:NEXT(16$)"</value>
</parameter>
```

Segment of FCF for PART B PROCESSING LINE:

```
<parameter>
    <segment_id>    "Working station"                               </segment_id>
    <entry_id>      DELAY                                           </entry_id>
    <type>          "ARENA FORMAT STRING"                           </type>
    <value>         "HoursToBaseTime(Triangular(.2,1,1.2)),,VA:NEXT(16$)"   </value>
</parameter>
```

### 4.3.2 Arena Interface

#### *4.3.2.1 Introduction*

Arena is a simulator developed by Systems Modelling (in April 2000 acquired by Rockwell Software). It allows direct connectivity to several tools and technologies. For the implementation of the MMP connection, a bi-directional interface to an intermediate program is used. This intermediate program is responsible for the communication with the adapter, accesses the Arena simulator using ActiveX, and receives interactions from the Arena side via a user defined DLL through a DCOM server.

#### *4.3.2.2 Description of HLA Arena Templates*

Six additional flow modules are defined for the HLA RTI connection and grouped within the "HLA" Template Panel. The modules are representing the necessary segments (input, output) defined within the Template Library approach.

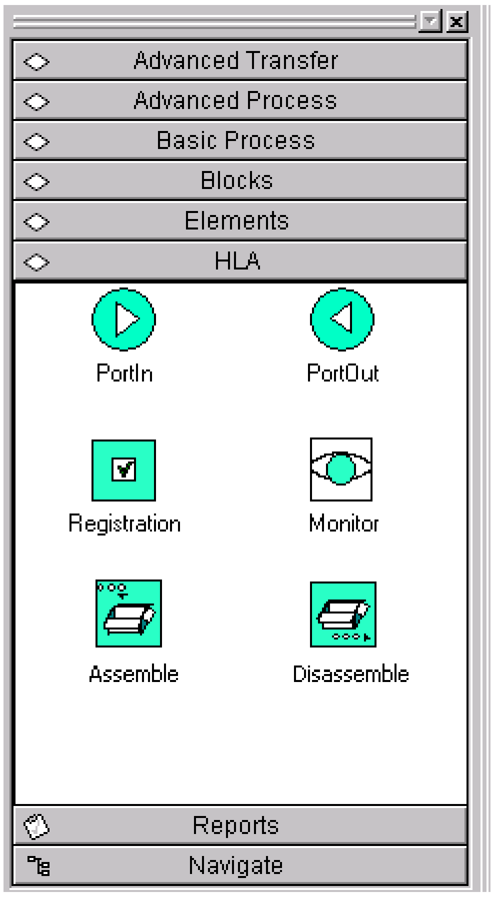


Figure 4.36: HLA Template panel

- **PortIn**
  Entities of other federates which are registered within the RTI can be received via a PortIn module from another Arena simulation model. At this time an Arena entity is created for the RTI object and (if possible) the RTI object is owned by the Arena federate. Then, the Arena entity crosses the simulation model.
- **PortOut**
  If the Arena entity reaches the PortOut module then the RTI object will be updated and the Arena entity will be destroyed. The ownership of the RTI object is given up. All entities which are registered at the RTI have to leave the model via a PortOut module. Otherwise the objects will be lost for the RTI.

- **Registration**
  An entity which is created within an Arena simulation model can be registered at the RTI by using the registration module.
- **Monitor**
  If an entity which is registered within the RTI reaches the Monitor module, all the available published attributes of the corresponding RTI object will be updated within the bus.
- **Assemble**
  An entity that is registered within the RTI and represents a container will be assembled here with the representative for the group of entities. In the bus, the container and its content will be marked as belonging together. Then, the resulting assembled entity cross the simulation model.
- **Disassemble**
  If an entity that is registered within the RTI and contains another objects (attribute ”carry” contains the list of the contained object references) reaches this block, then the contained objects will be extracted (only the topmost level). Then, the container and its content will be send into the model through two different exit points.

It is possible to have several PortIn, PortOut, Registration and Monitor modules within a single simulation model.

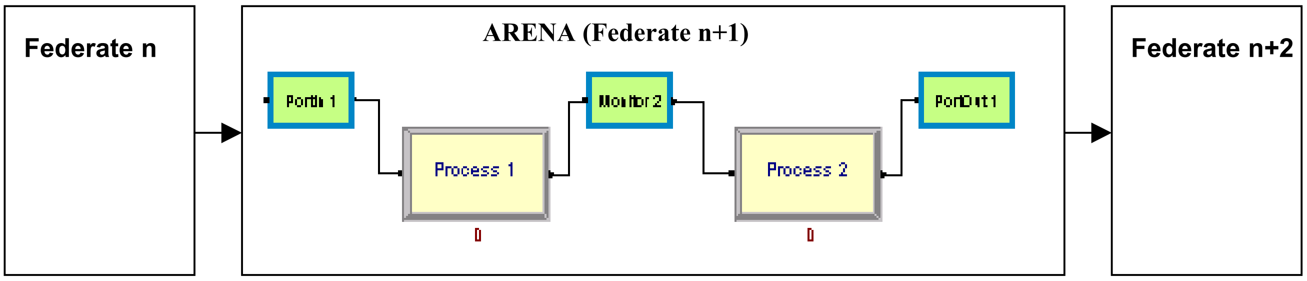


**Figure 4.37: Use of PortIn, Monitor and PortOut modules**

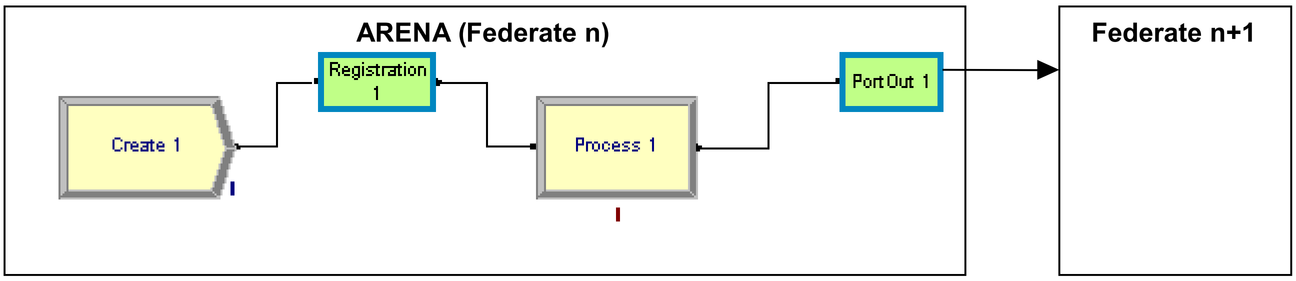


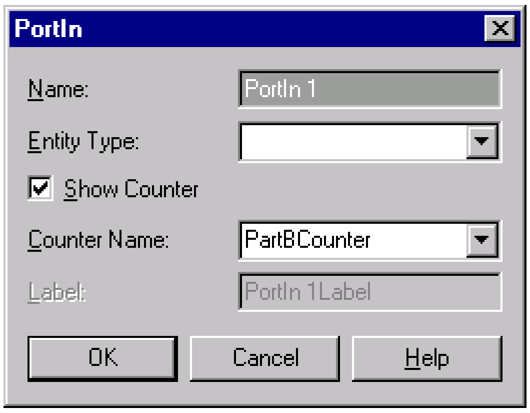
**Figure 4.38: Use of the Registration module**

Each of these modules has predefined parameters to identify the relations between them and the whole federation.

The parameters of the PortIn module (Figure 4.39) are:

- **Name**
  The value of the parameter "Name" is required. It can be any string but it has to be unique within the simulation model.
- **Entity Type**
  The "Entity Type" parameter allows the description of the entity/RTI object type which has to be caught from the RTI for this module. The entity type must be predefined within the entity data module (entity table).

- **Show Counter**
  This check box switches the displaying of a animation variable which contains amount of incoming entities on the model view.
- **Counter Name**
  This parameter is only available if the “Show Counter” is checked. A predefined counter (i.e. from the Template “Advanced Process”\”Statistic”) must be supplied here.
- **Label**
  This parameter is only for technical reasons and can not be changed by the end user.

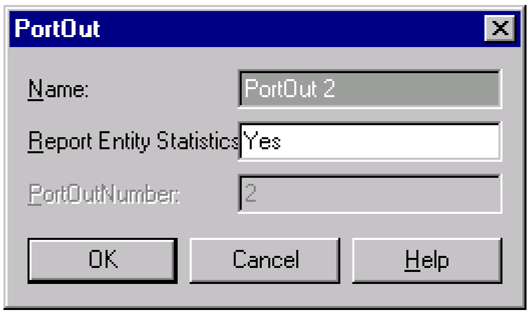


| | |
|---|---|
| Name: | PortIn 1 |
| Entity Type: | |
| ☑ Show Counter | |
| Counter Name: | PartBCounter |
| Label: | PortIn 1Label |

**Figure 4.39: Parameters of a PortIn module**

The parameters of the PortOut module (Figure 4.40) are:

- **Name**
  The value of the parameter "Name" is required. It can be any string but it has to be unique within the simulation model.
- **Report Entity Statistics**
  This parameter indicates if statistics are required.
- **PortOutNumber**
  This parameter is only for technical reasons and can not be changed by the end user.



| | |
|---|---|
| Name: | PortOut 2 |
| Report Entity Statistics | Yes |
| PortOutNumber: | 2 |

**Figure 4.40: Parameters of a PortOut module**

The parameters of the Registration module (Figure 4.41) are:

- **Name**
  The value of the parameter "Name" is required. It can be any string but it has to be unique within the simulation model.
- **Entity Type**
  The parameter "Entity Type" allows the description of the entity/RTI object type which will be registered at the RTI if an entity pass this module. The entity type must be predefined within the entity data module (entity table).

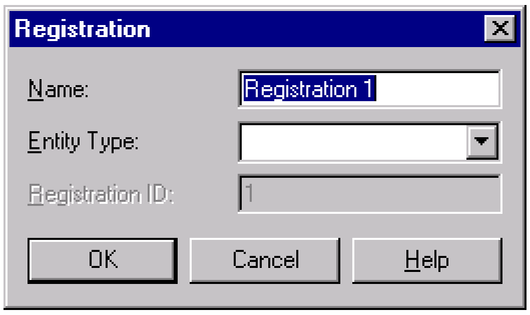


**Figure 4.41: Parameters of a Registration module**

- **Registration ID**
  This parameter is only for technical reasons and can not be changed by the end user.

The parameters of the Monitor module (Figure 4.42) are:

- **Name**
  The value of the parameter "Name" is required. It can be any string but it has to be unique within the simulation model.

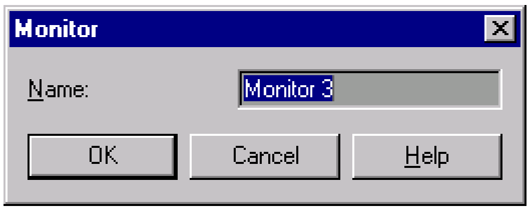


**Figure 4.42: Parameters of a Monitor module**

The parameters of the Assemble module (Figure 4.43) are:

- **Name**
  The value of the parameter "Name" is required. It can be any string but it has to be unique within the simulation model.
- **Assemble Type**
  The manner of the assembling must be supplied here. The user can select between “carry” and “parts”.
- **Entity Type Container**
  The type of container entity which must be known within the RTI. An container entity that passes this block must be registered within the RTI.

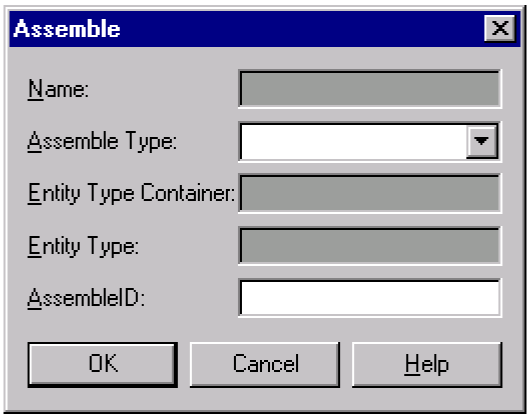


**Figure 4.43 : Parameters of a Assemble module**

- **Entity Type**
  The parameter "Entity Type" allows the description of the entity type which must be a representative for an entity group in Arena (i.e. constructed in a Batch module). The parts of such an entity must be entities that represents the registered RTI objects. The entity type must be predefined within the entity data module (entity table).
- **Assemble ID**
  This parameter is only for technical reasons and can not be changed by the end user.

The parameters of the Disassemble module (Figure 4.44) are:

- **Name**
  The value of the parameter "Name" is required. It can be any string but it has to be unique within the simulation model.
- **Entity Type**
  The parameter "Entity Type" allows the description which type of entity will be extracted from an incoming entity that must be a container object known within the RTI.
- **DisassembleID**
  This parameter is only for technical reasons and can not be changed by the end user.

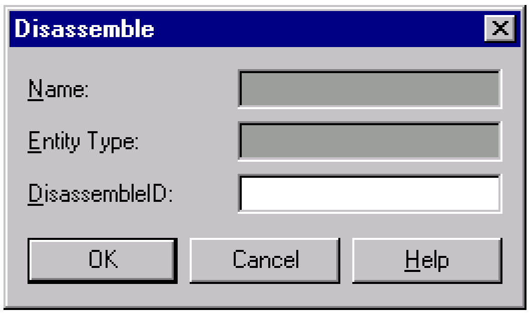
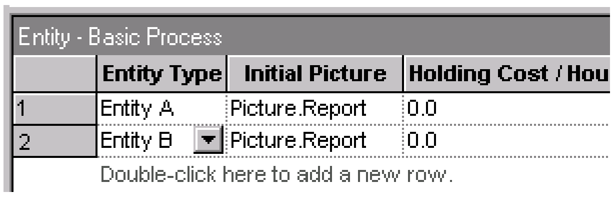


**Figure 4.44: Parameters of a Disassemble module**

### 4.3.2.3 *Involved Standard Modules of Arena*

Following information are additionally required in the standard modules:

- In the Entity data module (template “Basic Process”), entity types must be supplied which matches the simulation entity types in the FC file (tag “entity type”, subtag “related_id”).



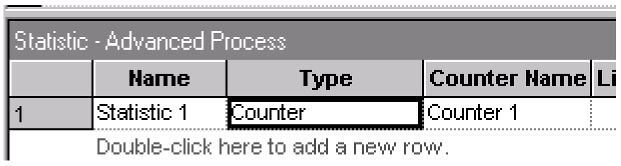
Entity - Basic Process

| | Entity Type | Initial Picture | Holding Cost / Hou |
|---|---|---|---|
| 1 | Entity A | Picture.Report | 0.0 |
| 2 | Entity B | Picture.Report | 0.0 |

Double-click here to add a new row.

**Figure 4.45: Entity data module**

- In the Statistic data module (template “Advanced Process”), one counter per PortIn module must be defined if incoming entity counting in this module is wished. The corresponding counter name must be entered into the parameter “Counter Name” of such one PortIn module.



Statistic - Advanced Process

| | Name | Type | Counter Name | Li |
|---|---|---|---|---|
| 1 | Statistic 1 | Counter | Counter 1 | |

Double-click here to add a new row.

**Figure 4.46: Statistic data module**

- In the Attributes flow module (Template "Elements"), all the HLA attributes must be added with exactly the same names as in the FC file (in the subtag "name" of the tag "attribute" in "exchange_class").

**Figure 4.47: Attributes flow module**

### 4.3.3 Taylor ED Interface – HLA Adapter

#### *4.3.3.1 Introduction to Taylor ED*

The simulation Taylor Program Enterprise Dynamics of F&H Simulations is classified among the discrete events driven simulator. This is a very "strong" program mainly focused on the industrial simulation, although it also accepts projects of general purpose.

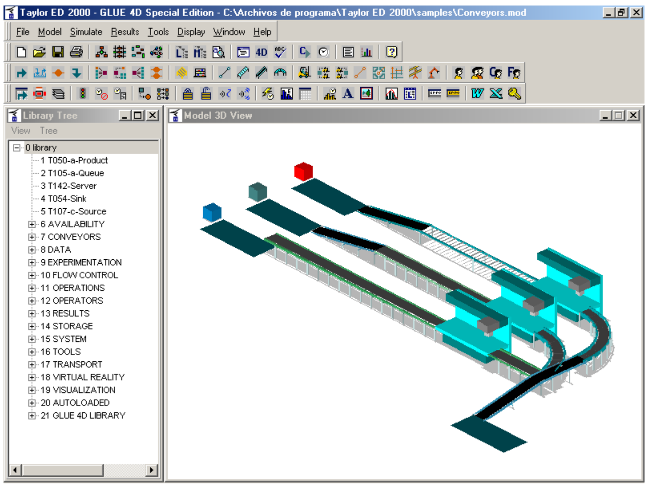
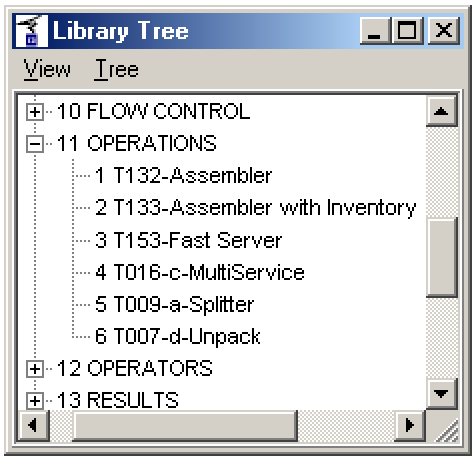


**Figure 4.48: Taylor user interface**

The Taylor ED models are built with atoms, which are the basic units provided by the program. Therefore, a model is a kind of puzzle made by the same or different type of pieces. F&H Simulations provides to the ED Taylor user some assembly atoms, packer atoms, conveyor atoms, warehouse atoms, so as to complete the wild functional variety of the product manufacturing.

Taylor ED includes a programming language so as the user could create those atoms required for a determined which hasn't been taken into account, for example, some kind of specific machine that doesn't fit with any of the standard series. This language is called 4D Script and is the basis of the interface between the HLA and Taylor ED.

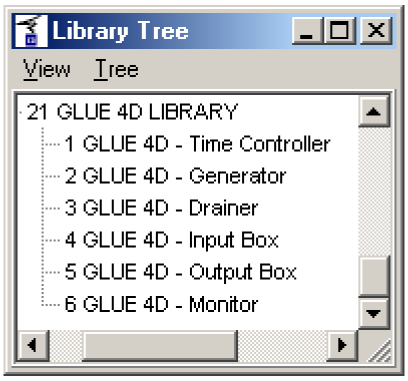
### *4.3.3.2 Communication with the HLA Adapter*



10 FLOW CONTROL
11 OPERATIONS
1 T132-Assembler
2 T133-Assembler with Inventory
3 T153-Fast Server
4 T016-c-MultiService
5 T009-a-Splitter
6 T007-d-Unpack
12 OPERATORS
13 RESULTS

**Figure 4.49: Taylor atom set**

Taylor ED manages the communication with other external programs in different ways. The one selected by the MISSION team is the communication through DDL libraries for the adapter HLA and the creation of a group of special atoms for the interface part of the user.



21 GLUE 4D LIBRARY
1 GLUE 4D - Time Controller
2 GLUE 4D - Generator
3 GLUE 4D - Drainer
4 GLUE 4D - Input Box
5 GLUE 4D - Output Box
6 GLUE 4D - Monitor

**Figure 4.50: MISSION HLA adapter Atom set**

The DDL library is written in a C language and is function is a dialogue of message with the adapter HLA. These messages are referred to exchange object's manipulation among the simulators.

The interface with the user has been developed in such a way that integrates softly with the atoms provided in ED Taylor. To reach this the following group of atoms has been created; all of them programmed with the language of Taylor ED, 4D Script.

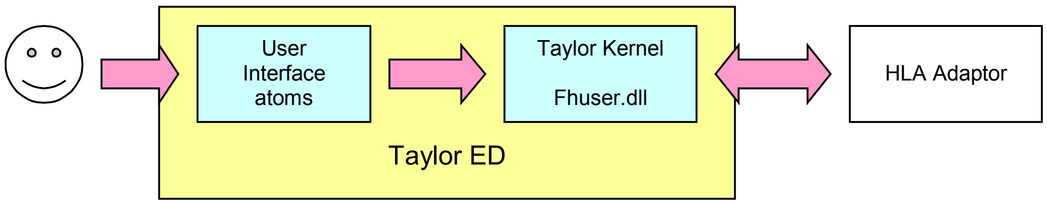


**Figure 4.51: Communication Interface**

#### 4.3.3.3 The Atom Set in Detail

***Time Controller***

*Purpose*:

It synchronises the simulation according to the time indicated by the HLA Adapter. It also realises some of the secondary functions, such as registration of characteristics of the exchange objects and the internal distribution of messages among the rest of the atoms.

*Parameters*:

Atom name. Internal Taylor name that identifies each atom (not the atom type).

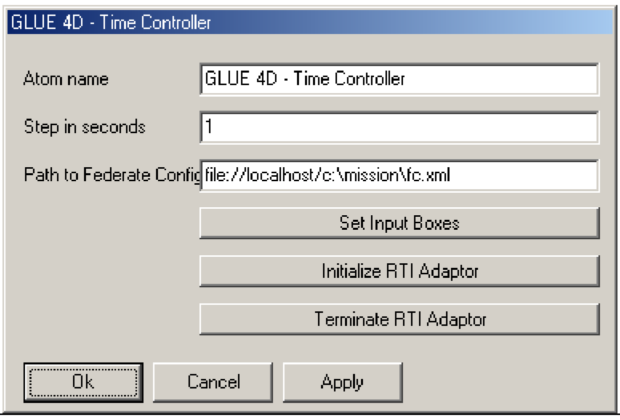
Step in seconds. Number of simulation seconds to be passed between synchronisation steps. The higher this number the less the accuracy in the distributed simulation and the less the time spent in communication between simulators.

Path to Federate Config file. Pointer to the XML file containing the Federate information.

*Functionality*

The Time Controller is the only atom that must exist in every simulation model and must be unique for each model. It does:

- Initialise the HLA RTI adapter: the model enters the distributed scenario as a new federate. Called function in fhuser.dll: IN.
- Terminate de HLA RTI adapter: the model exits the distributed scenario. Called function in fhuser.dll: TE.
- Registers the exchange object attributes. Called function in fhuser.dll: SA.
- Starts and checks the time control to synchronise with the distributed scenario. Called functions in fhuser.dll: ST and TA.
- Reads all the events from the HLA RTI adapter and filters the input events to the input boxes consigned in the internal table (see the screens). Called functions in fhuser.dll: WA and NW.


GLUE 4D - Time Controller

| | |
|---|---|
| Atom name | GLUE 4D - Time Controller |
| Step in seconds | 1 |
| Path to Federate Config | file://localhost/c:\mission\fc.xml |

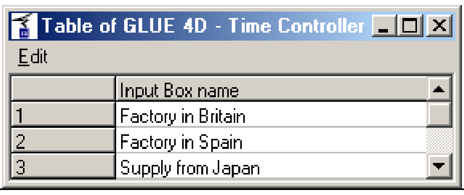


Table of GLUE 4D - Time Controller

| | Input Box name |
|---|---|
| 1 | Factory in Britain |
| 2 | Factory in Spain |
| 3 | Supply from Japan |

| GLUE 4D | *Generator* |
|---|---|

*Purpose:*

It creates exchanged objects in the stage of simulation. When an exchanged object is created, it is notified to all the simulators, though simulator keeps the property of the object which has created it.

*Parameters*:

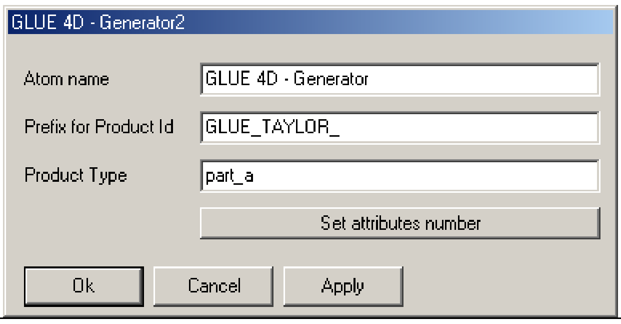
Atom name. Internal Taylor name that indentifies each atom (not the atom type).

Prefix for Product Id. Every entity created in the distributed scenario must have a unique id that is automatically generated by this atom. To ease the understanding of what an entity is, this atom lets the user to specify here an alphanumeric word which will be added to the id. For example, if we are producing automobiles we can put the word 'car_' so an entity could be 'car_27'.

Product type. The product type must be one of the declared types in the Federate Config file fc.xml.

*Functionality*

The Generator creates a new entity using the function CO in the fhuser.dll. This implies that a new object exists in the distributed scenario but it still haven't any data contained in it. So the Generator also put the attributes entered by the user throw the internal table using the function UO.


GLUE 4D - Generator2

| | |
|---|---|
| Atom name | GLUE 4D - Generator |
| Prefix for Product Id | GLUE_TAYLOR_ |
| Product Type | part_a |

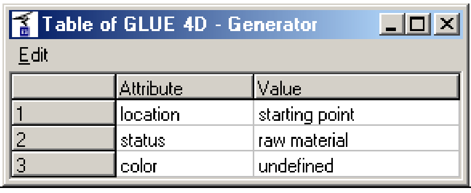


Table of GLUE 4D - Generator

| | Attribute | Value |
|---|---|---|
| 1 | location | starting point |
| 2 | status | raw material |
| 3 | color | undefined |

| GLUE 4D | ***Input Box*** |
|---|---|

*Purpose*:

In a simulator, it admits the entrance of exchanged objects coming from other simulator. This operation implies that the source simulator losses the property on the object, this property will be undertaken by the target simulator. Obviously, in the source simulator there will be an Output *Box.*

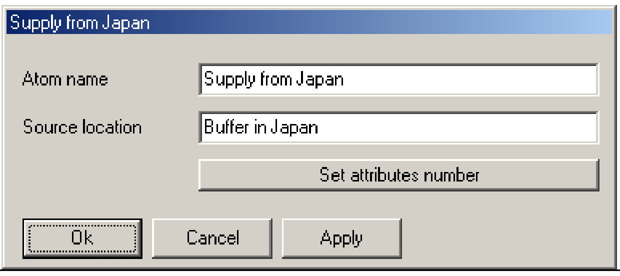
*Parameters*:

Atom name. Internal Taylor name that indentifies each atom (not the atom type).

Source location. This value must be the same as that one put in the location attribute in the Output Box connected to this Input Box. In fact the connection is made throw the concordance of these values.

*Functionality*

This atom works in narrow contact with the Time Controller. When the location of an entity is changed, every Time Controller in the scenario repeats the event to their respective Input boxes. The Input Boxes check if the new location is that which are waiting for, and if affirmative then take the property of the entity. Called function in fhuser.dll: SO.


Supply from Japan

| | |
|---|---|
| Atom name | Supply from Japan |
| Source location | Buffer in Japan |

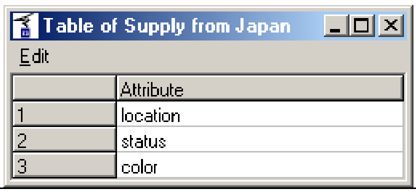
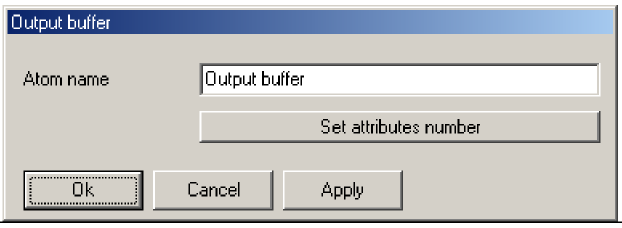
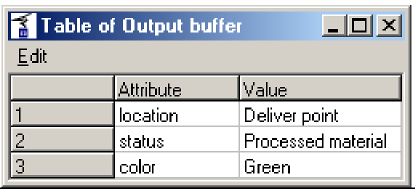


Table of Supply from Japan

| | Attribute |
|---|---|
| 1 | location |
| 2 | status |
| 3 | color |

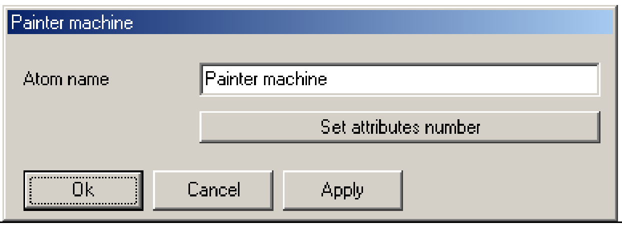
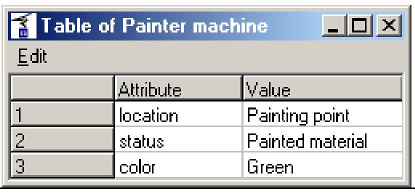
| GLUE 4D | ***Output Box*** |
|---|---|
| *Purpose:*<br>It prepares an exchanged object for the output of the simulation in which is placed towards another one. In this operation current simulator losses its property on the object and leaves it free so as another simulator could take it. | |
| *Parameters*:<br>Atom name. Internal Taylor name that identifies each atom (not the atom type). | |
| *Functionality*<br>In an abstract way you can think in an Output Box as a 'virtual place' where entities wait until an Input Box takes them. Every time an entity reaches this atom, its location is updated to that specified in the internal table. In some place of the distributed scenario must be an Input Box which is waiting for entities in that location who will take the control of them. Called functions in fhuser.dll: UO and RO. | |
|   | |

Output buffer

Atom name: Output buffer

Table of Output buffer

| | Attribute | Value |
|---|---|---|
| 1 | location | Deliver point |
| 2 | status | Processed material |
| 3 | color | Green |

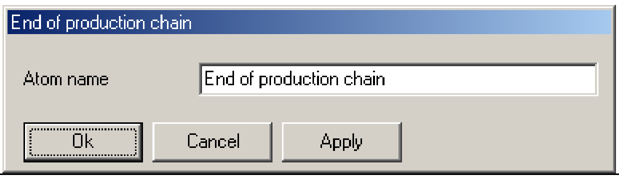
| GLUE 4D | ***Monitor*** |
|---|---|
| *Purpose:*<br>It allows the user to read and write the values in the attributes of the exchanged objects. Changes will be notified to all the simulators. | |
| *Parameters*:<br>Atom name. Internal Taylor name that identifies each atom (not the atom type). | |
| *Functionality*<br>When an entity reaches this atom it suffers a modification in its attributes. The new values must be put in the internal table. The difference between doing this internally (with Taylor proprietary atoms) or with the Monitor is that changes in the Monitor are notified to the all distributed scenario. Take note that although you can change the location attribute of an entity the current model keeps the property of it. Called function in fhuser.dll: UO. | |
|   | |

Painter machine

Atom name: Painter machine

Table of Painter machine

| | Attribute | Value |
|---|---|---|
| 1 | location | Painting point |
| 2 | status | Painted material |
| 3 | color | Green |

| GLUE 4D | ***Drainer*** |
|---|---|
| *Purpose*:<br>It eliminates exchanged objects from the simulation stage. It is the opposite of the *Generator.* | |
| *Parameters*:<br>Atom name. Internal Taylor name that identifies each atom (not the atom type). | |
| *Functionality*<br>This atom is extremely easy to understand. It simply eliminates from the distributed scenario all the entities that reaches it. It doesn't need any additional data. Called function in fhuser.dll: DO. | |
|  | |

End of production chain

Atom name: End of production chain

### 4.3.3.4 Example of Taylor User Interface

With this group of atoms it is possible to make all the necessary combinations for the distributed simulation manufacturing. The Simple MISSION Factory is a small but complete simulation scenario divided in four pieces. Following this it is the first piece: Processing Line A built in Taylor.

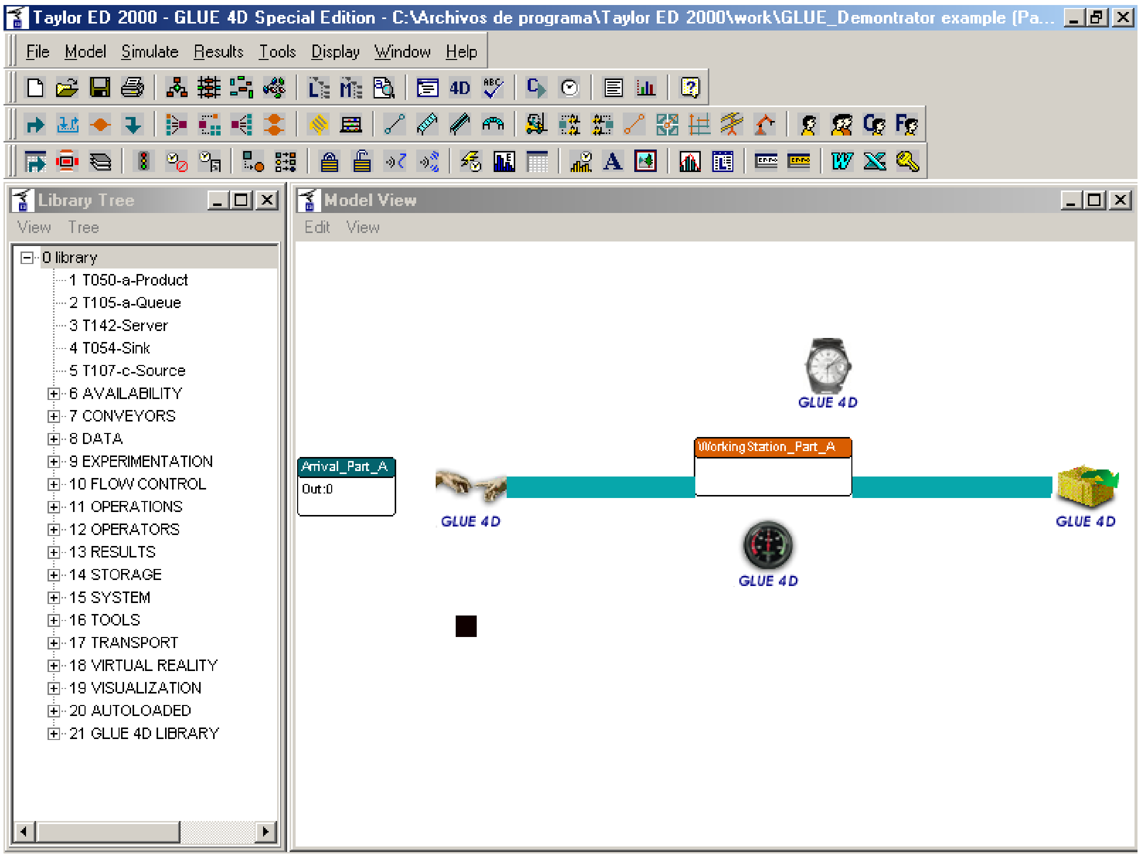


Figure 4.52: Simulation Model (Note the special atoms integrated with the built-in atoms)

See the Time Controller atom for the synchronisation of the rest of the simulation stage, the Generator that create new entities, the monitor that change entities attributes and the Output Box which is the exit point to the AGV part.

### *4.3.3.5 The FHUSER.DLL File in Detail*

The connection between the Taylor Simulator and the Adapter is based in a **D**ynamic **L**ink **L**ibrary*,* a library of executable functions or data that can be used by a Windows Application. Typically, a DLL provides one or more particular functions and a program accesses the functions by creating either a static or dynamic link to the DLL. A static link remains constant during program execution while a dynamic link is created by the program as needed.

For this application we have used the dynamic link so the connection between the Adapter and Taylor is only established when an event occurs.

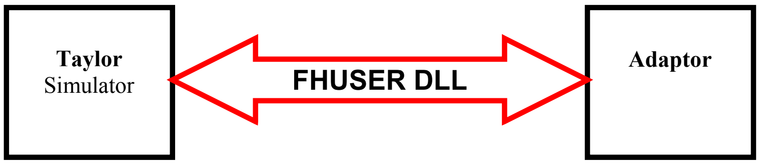


**Figure 4.53: Connection between Taylor ED and the Adapter**

For communication with external applications Taylor uses a special DLL written in C called **FHUSER** provided with the package. The functions needed for that purpose are added to this DLL. We have developed a set of commands that are deployed when ever a simulation event occurs. This commands are:

| **The FHUSER function set** |
|---|
| **1. IN<XML_file>**<br>• *Abstract*<br>This command calls to the function MissionShellInitialize of the C shell provided for the HLA Adapter, which must be the first function called in order to create the Java Virtual Machine, The HLA Adapter and the Federation.<br>• *Arguments*<br>-XML_file: FC file to configure the federate<br>• *Return values*<br>-OK : if the initialization is successful<br>-ERR_SHELL: otherwise |
| **2. TE**<br>• *Abstract*<br>This command calls to the function MissionShellTerminate, which must be the last function called.<br>• *Arguments*<br>None<br>• *Return values*<br>-END : The Adapter is ended |

**3. ST**

- *Abstract*

This command calls to the function MissionShellStartTimeControl. This function starts time control or changes lookahead if time control was already started.

- *Arguments*

  None

- *Return values*

  -OK : if successful

  -ERR_TIME: otherwise

**4. TA<Time>**

- *Abstract*

This command calls to the function MissionShellAdvanceTimeTo. This function advances the time x steps, where x represents <Time> argument.

- *Arguments*

  -Time: new time where the simulation wants to advance to.

- *Return values*

  - TA_OK : if successful

  -ERR_SHELL: otherwise

**5. WA**

- *Abstract*

This command calls to the function MissionShellAllWarningsReceived. This function is used to know if there is any warning pending.

- *Arguments*

  None

- *Return values*

  -EVENT : if there is a warning

  -NO_EVENT: if there is not a pending warning

  -ERR_EV: otherwise

**6. NW**

- *Abstract*

This command calls to the function MissionShellGetNextWarning. This function is used to get the warnings stored in the Adapter.

- *Arguments*

  None

- *Return values*

  -Warning : if successful  it returns the pending warning

  -NO_EVENT: if there is not any pending warning

  -ERR_NW: otherwise

**7. SA<Class>,<Attribute>**

- *Abstract*

This command calls to the function MissionShellSubscribeAttribute. This function is used to subscribe to an attribute.

- *Arguments*

  -Class: The Class name of the attribute

  -Attribute: The name of the attribute to subscribe to.

- *Return values*

  -SA_OK: if successful

  -ERR_SA: otherwise

**8. CO<Id>,<Class>**

- *Abstract*

This command calls to the function MissionShellCreateObject. This function is used to create an object  that belongs to  class <Class> and called  <Id>.

- *Arguments*

  -Class: The Class name of the new object

  -Id: The name of the object to be created.

- *Return values*

  -OBJ_CR: if successful

  -ERR_CO: otherwise

**9. RO<Id>**

- *Abstract*

This command calls to the function MissionShellReleaseObject. This function is used to release an object ownership.

- *Arguments*

  -Id: Id of the object to release

- *Return values*

  -RO_OK: if successful

  -ERR_RO: otherwise

**10. GO<Id>**

- *Abstract*

This command calls to the function MissionShellGetObject. This function is used to read an object from the bus.

- *Arguments*

  -Id: The name of the object to read.

- *Return values*

  -GO_OK: if successful

  -ERR_GO: otherwise

**11. SO<Id>**

- *Abstract*

This command calls to the function MissionShellSeizeObject. This function is used to seize all the attributes of the object called <Id>.

- *Arguments*

  -Id: The name of the object to be seized.

- *Return values*

  -SO_OK: if successful

  -ERR_SO: otherwise

**12. DO<Id>**

- *Abstract*

This command calls to the function MissionShellDeleteObject. This function is used to delete the object called <Id> from the bus.

- *Arguments*

  -Id: The name of the object to delete

- *Return values*

  -DO_OK: if successful

  -ERR_DO_OK: otherwise

## 4.4 Simulation Manager

### 4.4.1 Introduction

This chapter describes the functionality of the Simulation Manager Process Modeller (SMPM) and the Simulation Manager Building Block Modeller (SMBBM). These two components are described with each other, because they are very similar.

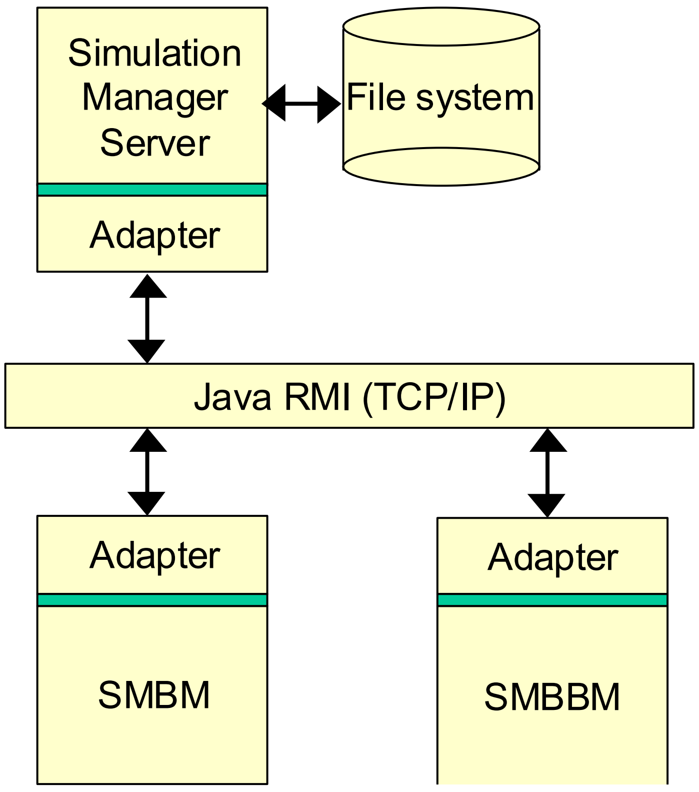


**Figure 4.54: Communication between SMPM / SMBBM and the Simulation Manager Server**

To work with these two tools a special tool called Administration GUI is included.

Since the server has no graphical user interface, the Adminstration GUI has the task to allow the user to load and save models or lunch applications. Without this tool a working with the SMPM and SMBBM is not possible.

### 4.4.2 Technical Issues

The SMPM and the SMBBM are connected with the Simulation Manager Server (SM-Server) via the Remote Method Invocation (RMI) method of Java. RMI has the advantage of security, multi-threading and object orientation. Especially because of the object oriented aspects a faster and more reliable implementation is possible.

## 4.4.3 Requirements

### *4.4.3.1 Hardware*

The minimum Hardware requirement for a practicable usage of the SMPM / SMBBM is a AMD or Pentium System with at least 233 MHz and 64 MB RAM. For running the Simulation Manager Server on the same computer 128 MB RAM are recommended.

### *4.4.3.2 Software*

For the applications it is needed to have a Java Runtime Environment 1.2 or greater installed. If you do not have a JRE, than you can downloaded one without charge from Sun at http://java.sun.com/j2se/1.3/ or from IBM at http://www-106.ibm.com/developerworks/java/jdk/?dwzone=java for Windows 9x, NT, 2000 and several *nix Versions.

The Simulation Manager Server requires Windows to work, but it runs on Windows 9x, NT and 2000.

## 4.4.4 Installation

The distribution consists of one archive (simmanager.zip) for the Simulation Manager which contains the following files:

| **Filename** | **Directory** | **Description** |
|---|---|---|
| smpm.jar | ./lib | Archive containing the SMPM program files. |
| smpm.bat | ./bin | Batch file for starting the SMPM. |
| smpm-config.xml | ./config | A XML configuration file containing some necessary information for the execution. |
| smpm -config.dtd | ./config | The document type definition for the configuration file. |
| smbbm.jar | ./lib | Archive containing the SMBBM program files. |
| smbbm.bat | ./bin | Batch file for starting the SMBBM. |
| smbbm -config.xml | ./config | A XML configuration file containing some necessary information for the execution. |
| smbbm -config.dtd | ./config | The document type definition for the configuration file. |
| tl.jar | ./lib | Archive containing the TL-Editor program files. |
| simserver.jar | ./lib | Archive containing the Java bindings for the Simulation Manager Server. |
| server.bat | ./bin | Batch file for starting the Simulation Manager Server. |
| admin.bat | ./bin | Batch file for starting the Administration Tool. |
| umcsjnii.dll | ./lib | DLL needed by the adapter. |
| umcServer.dll | ./lib | The Simulation Manager Server. |
| ….jar | ./lib | Several additional libraries which are needed by the applications. |

**Table 4.5: Content of the simmanager.zip file**

For the installation it is only necessary to unzip the zip-file with his directory structure files into any directory.

### 4.4.5 Configuring the Programs

The configuration file for the SMPM / SMBBM is a simple XML file (smpm-config.xml, smbbm-config.xml) which has to be edited by the user before the programs can be executed. The configuration file has the following format:

```
<Simulation_Manager_Server>
      <Address>
            [address]
      </Address>
      <Port>
            [port]
      </Port>
</Simulation_Manager_Server>
```

where

- address is the TCP/IP address or a name which can be resolved by a name server where the Simulation Manager Server is running at this time and
- port is the port where the Simulation Manager Server is running at this time.

A valid configuration file is for example:

```
<Simulation_Manager_Server>
      <Address>
            um183.ipk.fhg.de
      </Address>
      <Port>
            5500
      </Port>
</Simulation_Manager_Server>
```

The address and port have to be set to point to the computer where the Simulation Manager Server is running.

### 4.4.6 Starting the Program

The Simulation Manager Server must be running on the computer defined in the Simulation Manager Server section of the configuration file (see 4.4.5), before the SMPM / SMBBM can be started. The Server is invoked by double clicking on the server.bat.

If the connection to the Simulation Manager Server fails then the user will get an error message and the program shuts down. Remember that if a firewall is used, then the Simulation Manager needs the rights to pass the firewall.

After this step the Administration GUI can be started by using the admin.bat file.

### 4.4.7 The Administration GUI

The Administration GUI allows the user to manage systems (create new, load or store systems) and to start the SMPM, the SMBBM and the TL-Editor. If every information is entered a FC-File can be generated with this tool.

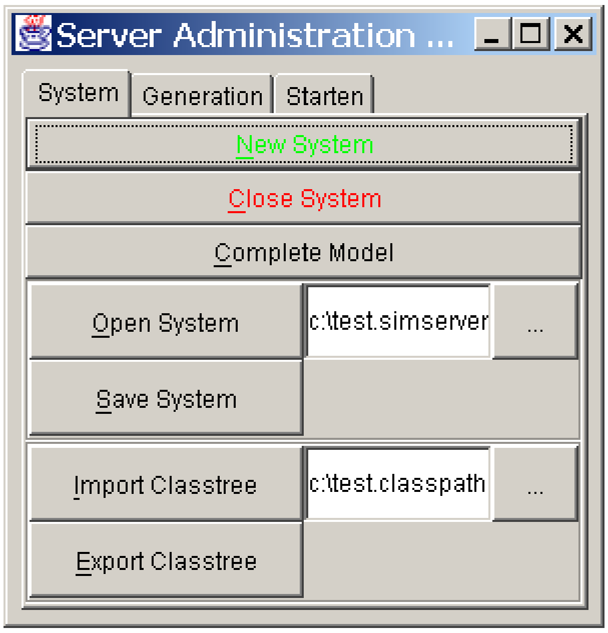


**Figure 4.55: The Administration GUI, system management**

| | |
|---|---|
| **New System**: | Creates a new empty model. |
| **Close System:** | Closes the currently edited system. |
| **Complete Model:** | Adds some missing classes to the system. |
| **Open System**: | Opens a existing model from a file. |
| **Save System**: | Saves the currently edited model to a file. |
| **Import Classtree**: | Imports a the class structure of the library. |
| **Export Classtree**: | Exports the class structure of the library to a file. |

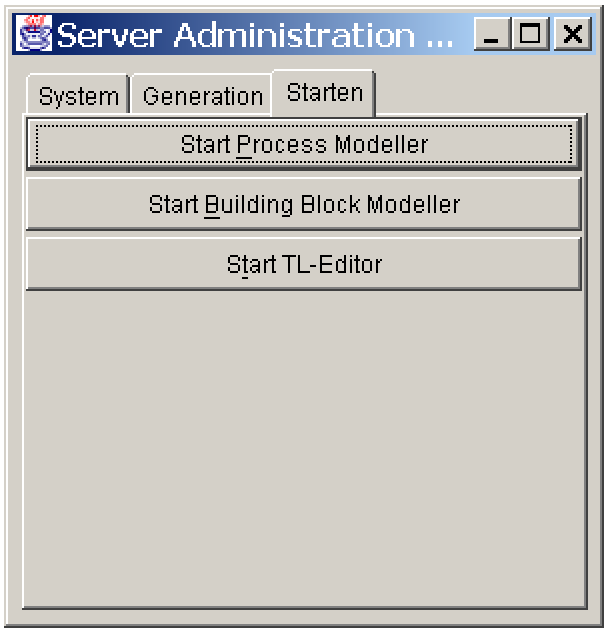


**Figure 4.56: The Administration GUI, starting of tools**

| | |
|---|---|
| **Start Process Modeller**: | Starts the SMPM. |
| **Start Building Block Modeller**: | Starts the SMBBM. |
| **Start TL-Editor**: | Starts the TL-Editor. |

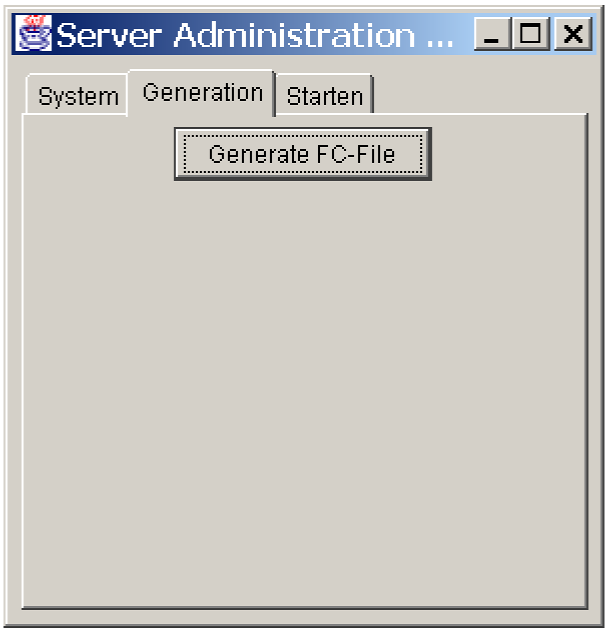


**Figure 4.57: The Administration GUI, Generating register**

**Generate FC-File**: Generates the FC-File for the currenty edited system.

### 4.4.8 SMPM and SMBBM

The two tools are from the look and feel very similar.

The SMPM and SMBBM have the following features:

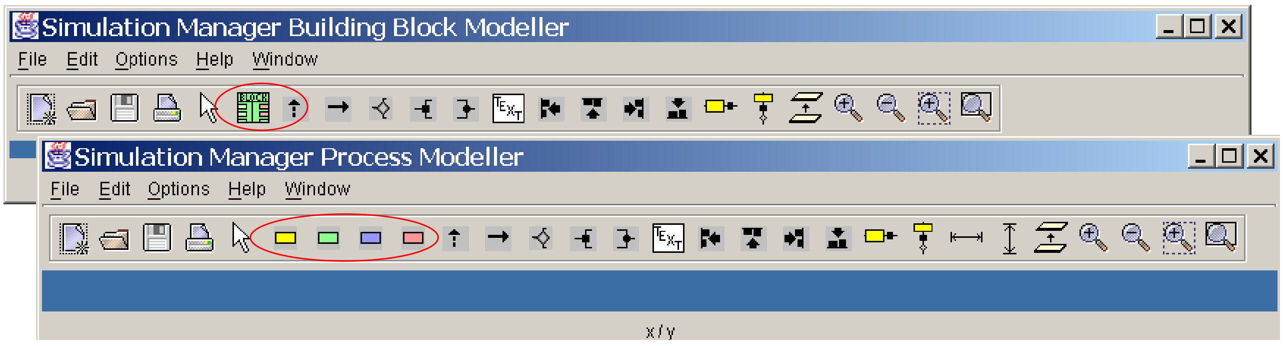

Simulation Manager Building Block Modeller
File Edit Options Help Window
BLOCK
Simulation Manager Process Modeller
File Edit Options Help Window
x / y

**Figure 4.58: SMPM and SMBBM appearance**

- opening models
- saving models during editing
- printing models
- inserting elements
- editing elements
- connecting elements
- aligning elements
- inserting text comments
- zooming.

### 4.4.8.1 SMPM Toolbar

The SMPM toolbar has the following appearance:

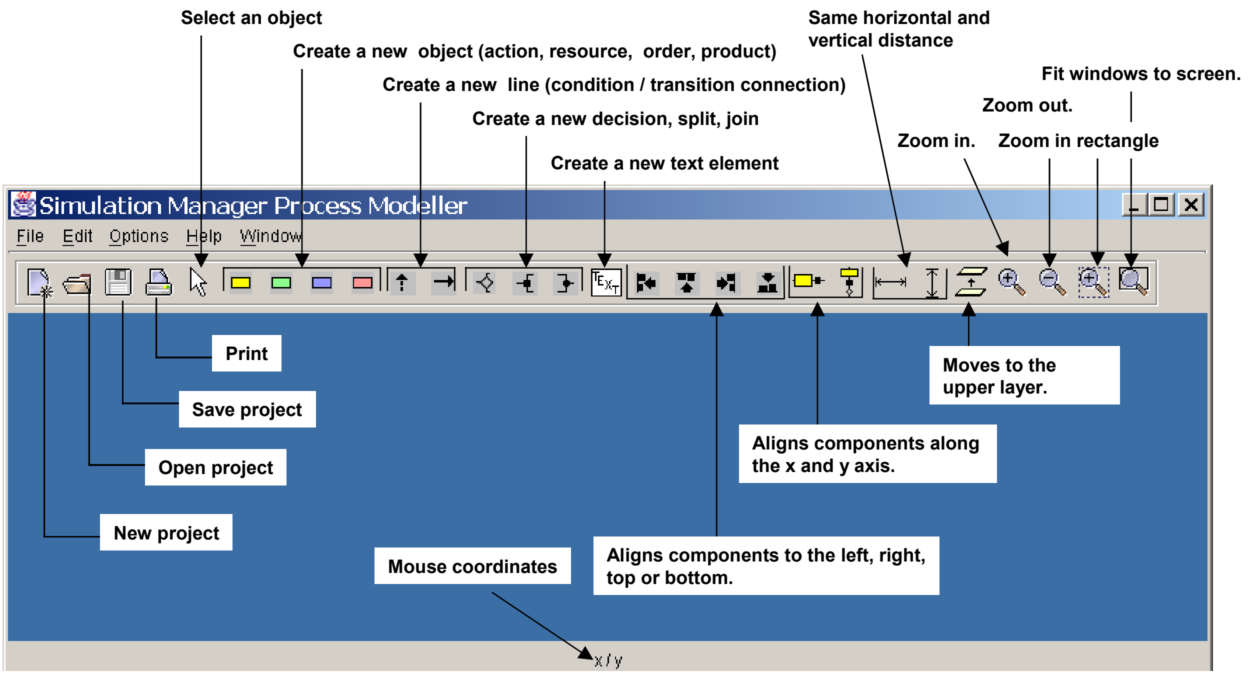

Select an object
Create a new object (action, resource, order, product)
Create a new line (condition / transition connection)
Create a new decision, split, join
Create a new text element
Same horizontal and vertical distance
Fit windows to screen.
Zoom out.
Zoom in.
Zoom in rectangle
Simulation Manager Process Modeller
File Edit Options Help Window
Print
Save project
Open project
New project
Mouse coordinates
Aligns components to the left, right, top or bottom.
Aligns components along the x and y axis.
Moves to the upper layer.
x / y

**Figure 4.59: SMPM toolbar**

### 4.4.8.2 SMBBM Toolbar

The SMBBM toolbar looks like this:

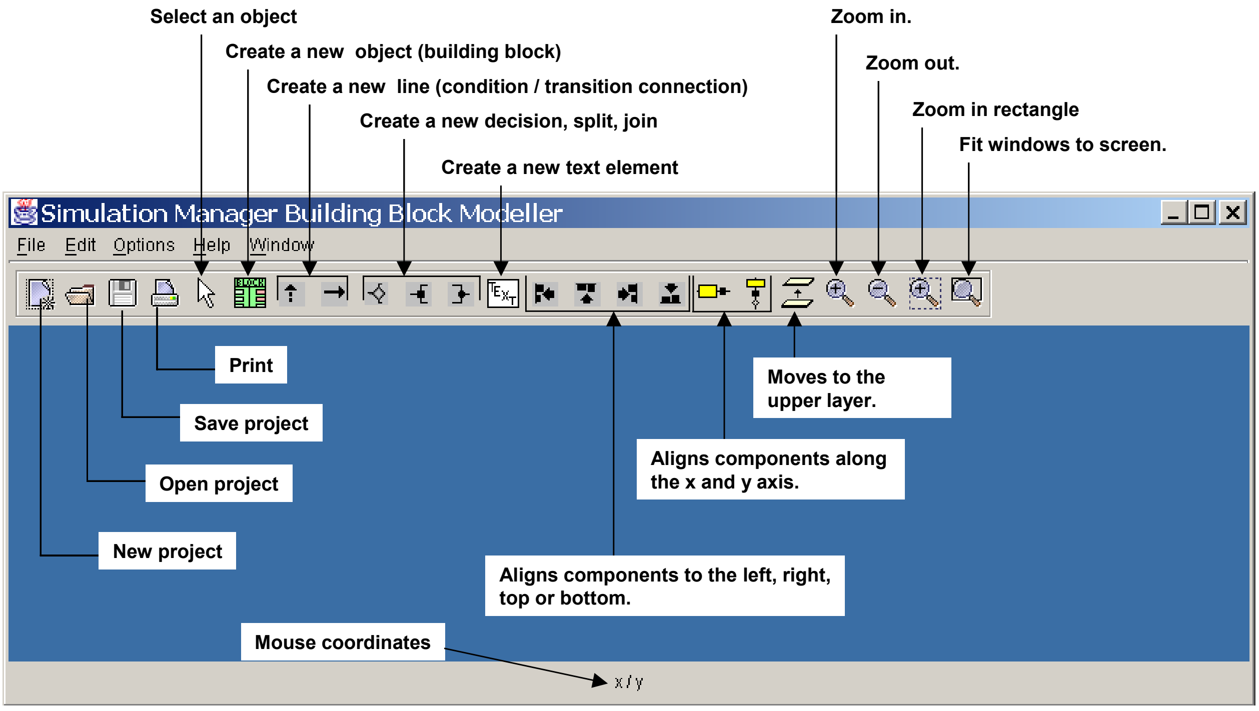

Select an object
Create a new object (building block)
Create a new line (condition / transition connection)
Create a new decision, split, join
Create a new text element
Zoom in.
Zoom out.
Zoom in rectangle
Fit windows to screen.
Simulation Manager Building Block Modeller
File Edit Options Help Window
Print
Save project
Open project
New project
Moves to the upper layer.
Aligns components along the x and y axis.
Aligns components to the left, right, top or bottom.
Mouse coordinates
x / y

**Figure 4.60: SMBBM toolbar**

### 4.4.8.3 File Menu

The file menu gives the following possibilities:

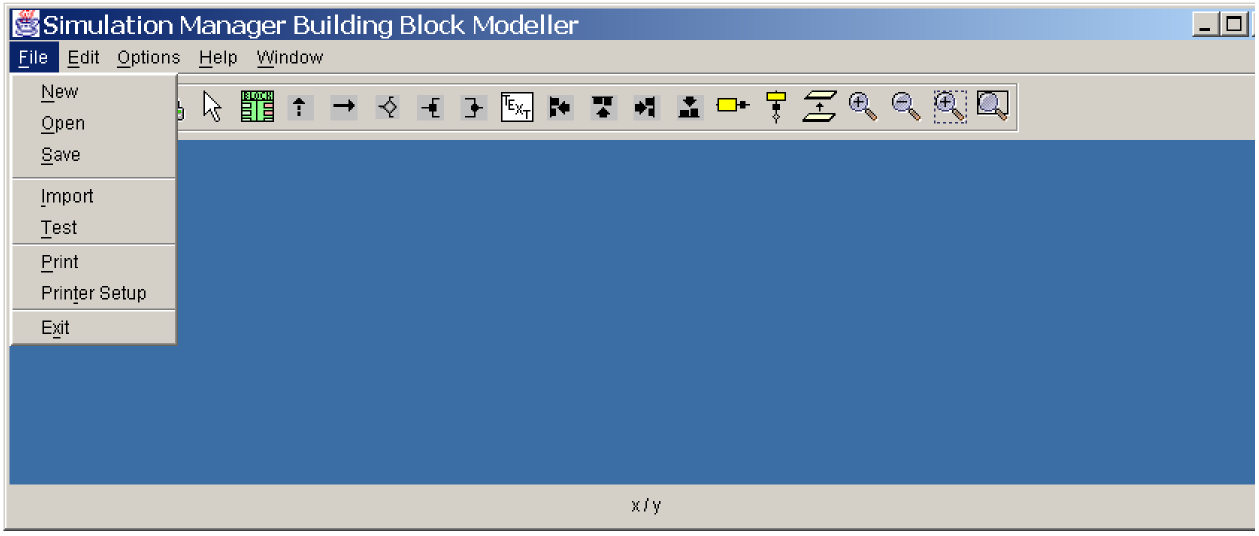

Simulation Manager Building Block Modeller
File Edit Options Help Window
New
Open
Save
Import
Test
Print
Printer Setup
Exit
x / y

**Figure 4.61: The file menu**

Functionalities:

| | |
|---|---|
| **New**: | Create a new view. |
| **Open**: | Open an existing view. |
| **Save**: | Save the view. |
| **Import**: | Import a model from another source. |
| **Print**: | Print the model. |
| **Printer Setup**: | Edit the printer preferences. |
| **Exit**: | Close the SMPM / SMBBM. |

#### *4.4.8.4 Options Menu*

The "**Options Menu**" allows the user to change the program preferences. Especially to change the language of the program. For example from English to German or to Spain. This can be done with little effort.

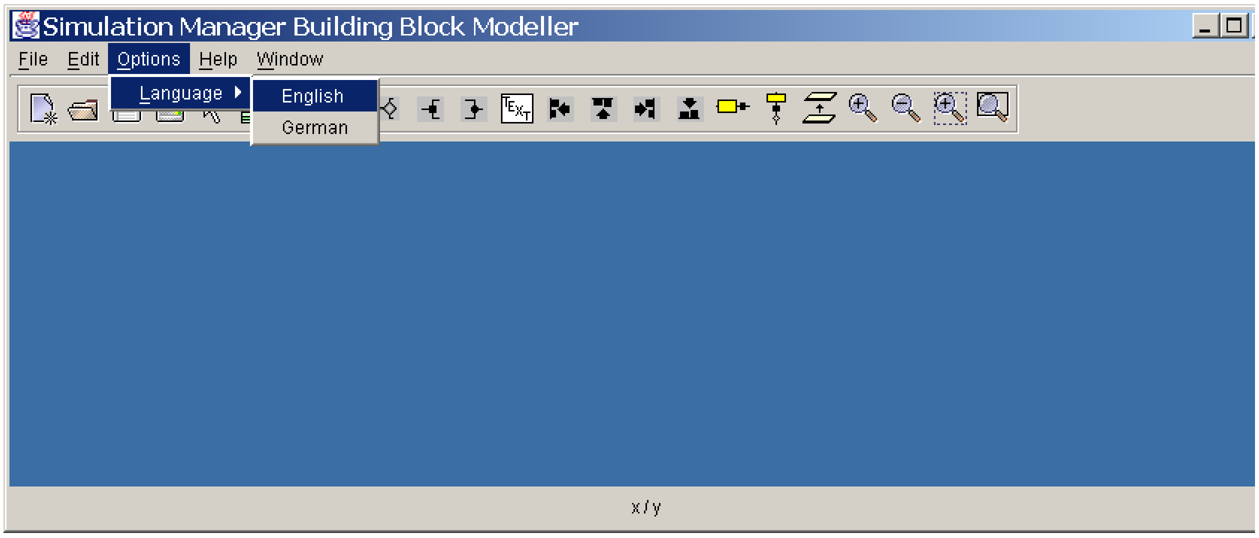


**Figure 4.62: The options menu**

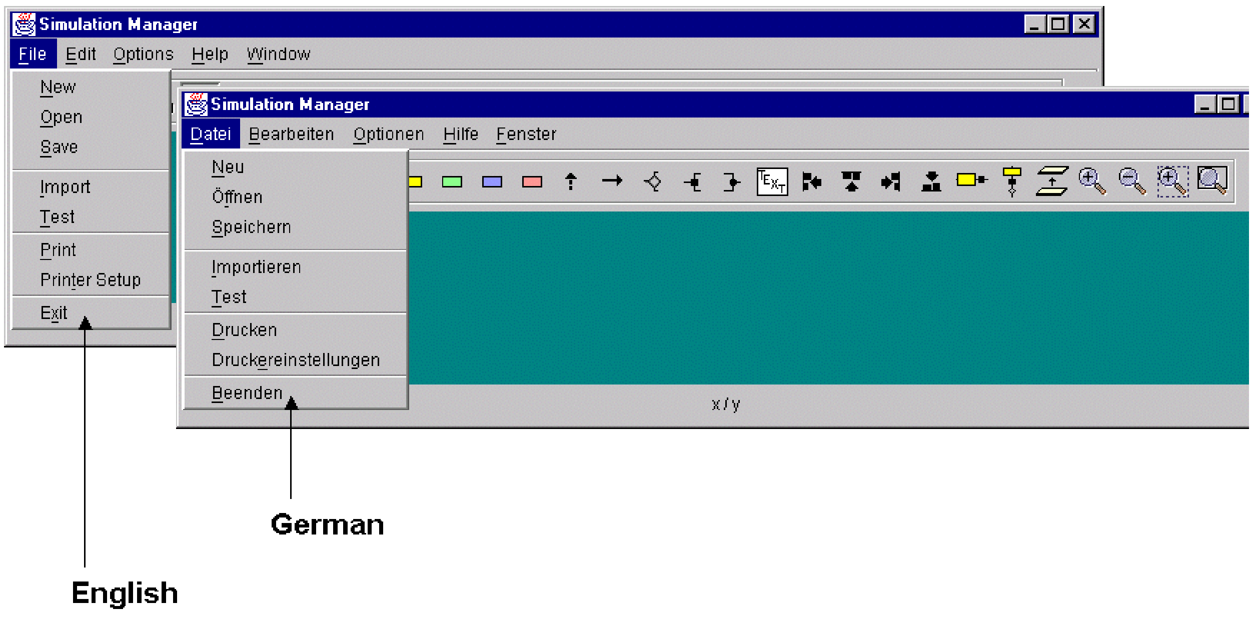


**Figure 4.63: View of different languages**

### *4.4.8.5 Window Menu*

The “**Window menu**” allows the user to switch from one window to another.

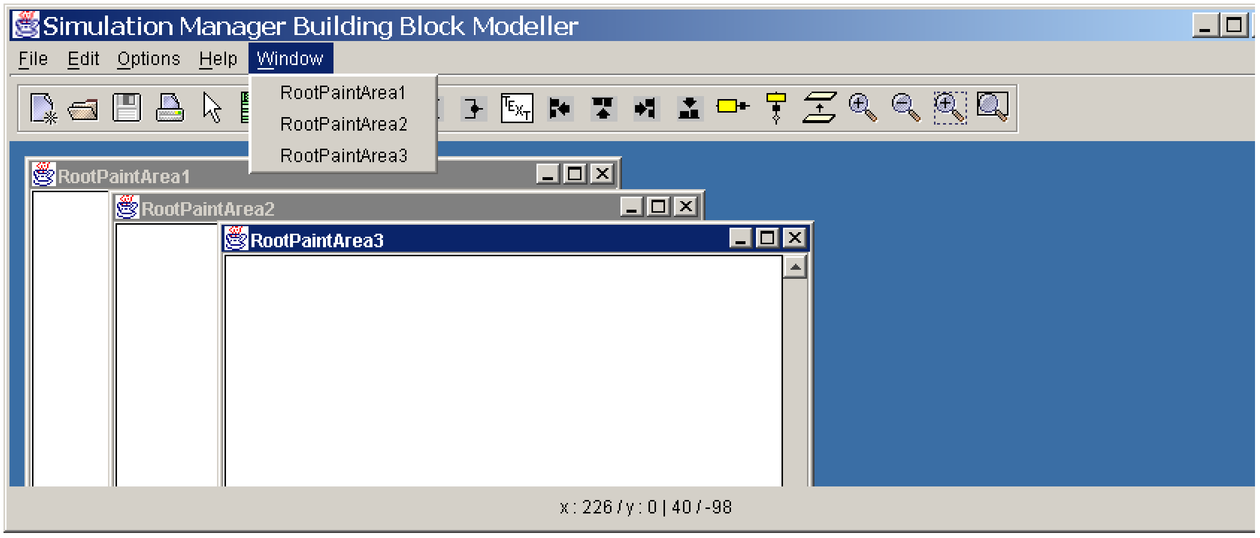

Simulation Manager Building Block Modeller
File Edit Options Help Window
RootPaintArea1
RootPaintArea2
RootPaintArea3
x : 226 / y : 0 | 40 / -98

**Figure 4.64: The windows menu**

### *4.4.8.6 Creating a Model*

For the creation of a model the Administration GUI needs to be run and a system has to be loaded or created at first.

The element to place into the model has to be selected from the toolbar with the left mouse button and after a click with the left mouse button inside the draw area it will be appear.

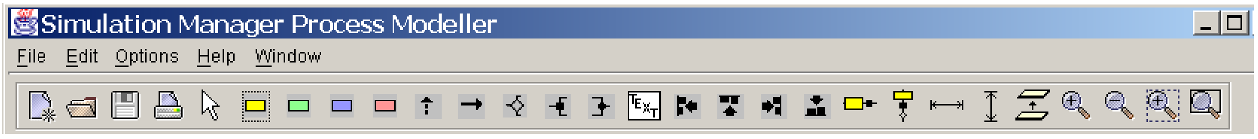
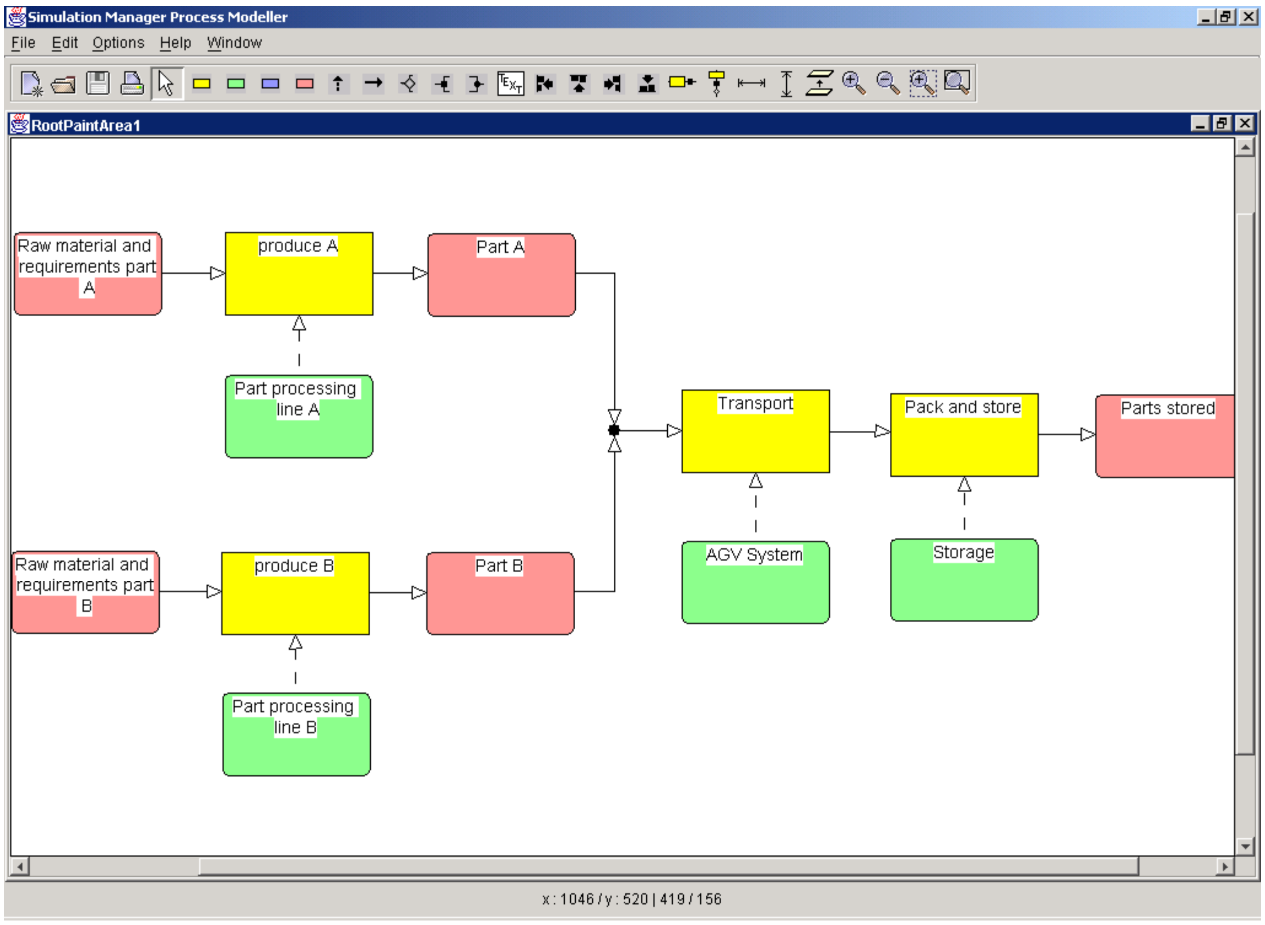


**Figure 4.65: Selected element at the toolbar**

Connecting two elements is done by selecting a connector from the toolbar and pressing the left mouse button at the source element and dragging the mouse to the target element. After unhanding the mouse button the connection is created.

The connection of building blocks in the SMBBM is done by the same way.

### 4.4.9 Example for a Model Inside the SMPM



F

**Figure 4.67: Example for an SMPM model (in this case: Simple Mission Factory)**

### 4.4.10 Example for a Model Inside the SMBBM

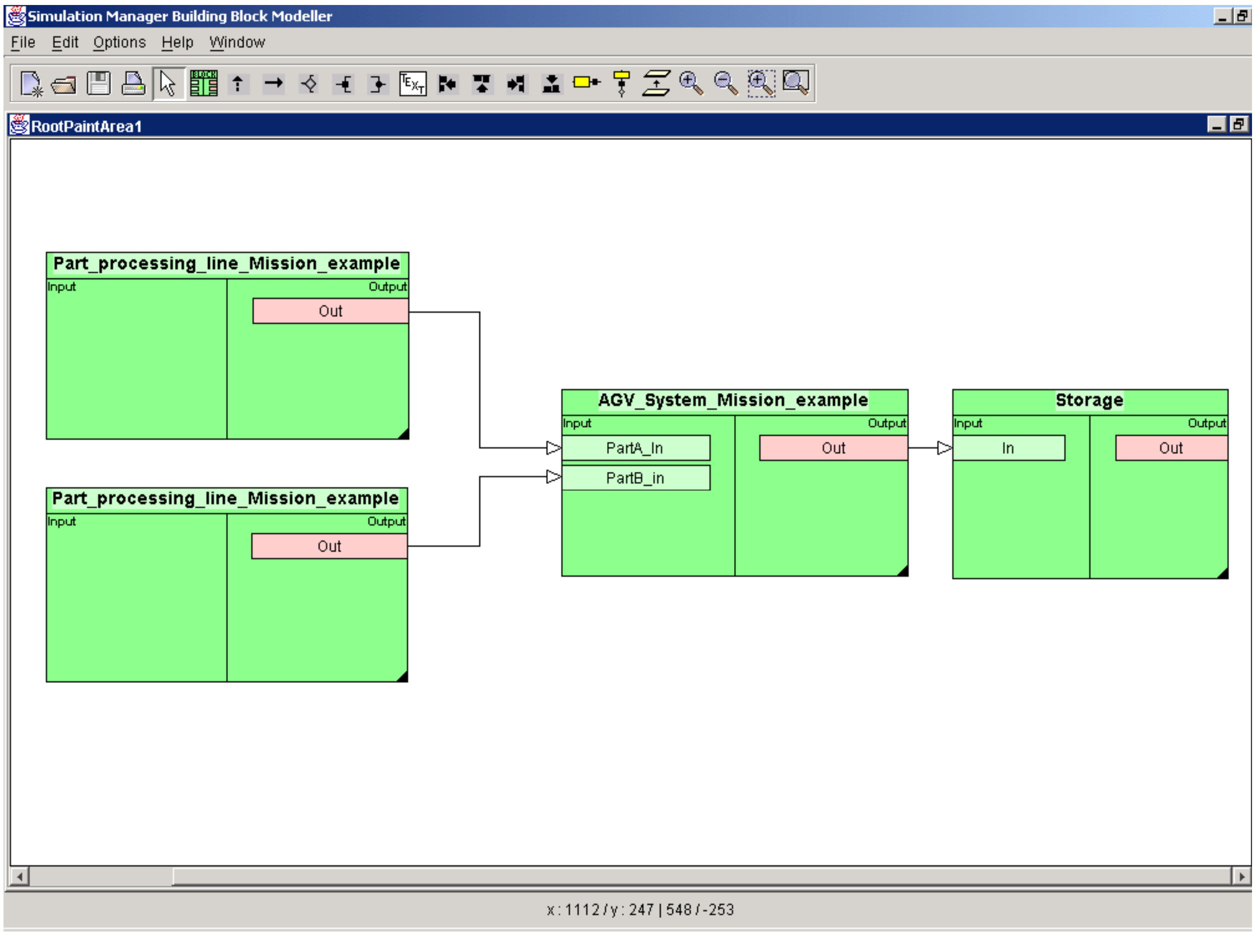

Simulation Manager Building Block Modeller
File Edit Options Help Window
RootPaintArea1
Part_processing_line_Mission_example
Input
Output
Out
Part_processing_line_Mission_example
Input
Output
Out
AGV_System_Mission_example
Input
Output
PartA_In
PartB_in
Out
Storage
Input
Output
In
Out
x : 1112 / y : 247 | 548 / -253

**Figure 4.68: Example for a SMBBM model (Simple Mission Factory)**

### 4.4.10.1 *Changing Attributes*

It is possible to change some attributes of an element (especially the name). For doing this the user has to click with the right mouse button on an element to choose the “**Set Attributes**” menu point. A dialog to set attributes of the element will come on the screen. The following dialogs displays this sequence.

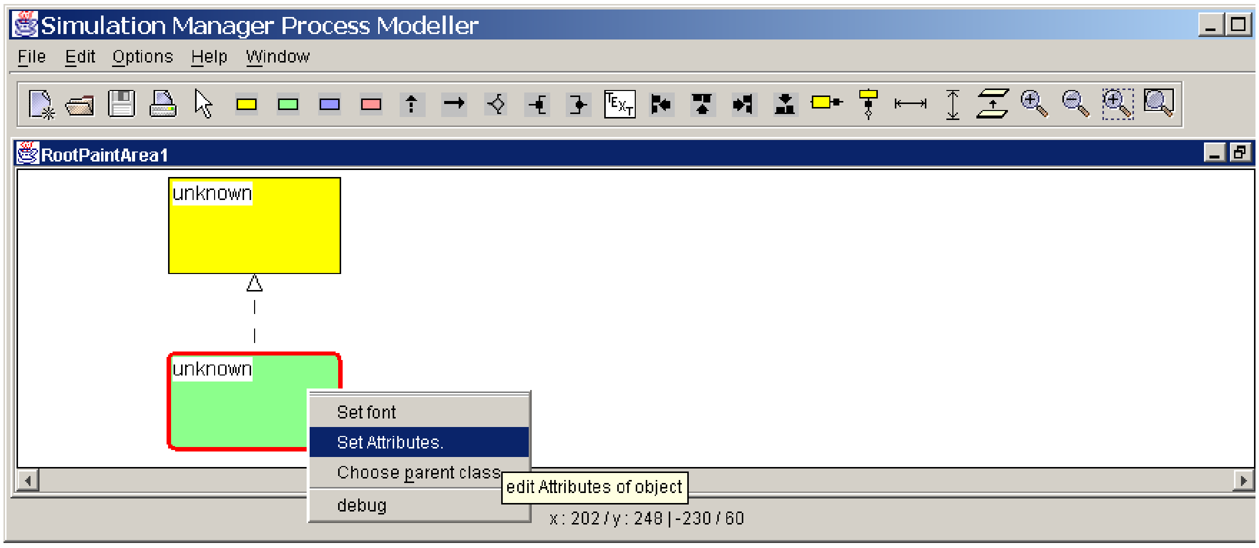


**Figure 4.69: Changing the attributes of an element**

To edit attribute values the user can click inside the value column of the table and set the attribute values.
There is also the possibility to add new attributes to the parent class of this object and to delete attribute from the parent class. With the renaming button attributes can be renamed.
Changing attributes in the SMBBM is a quite similar process.

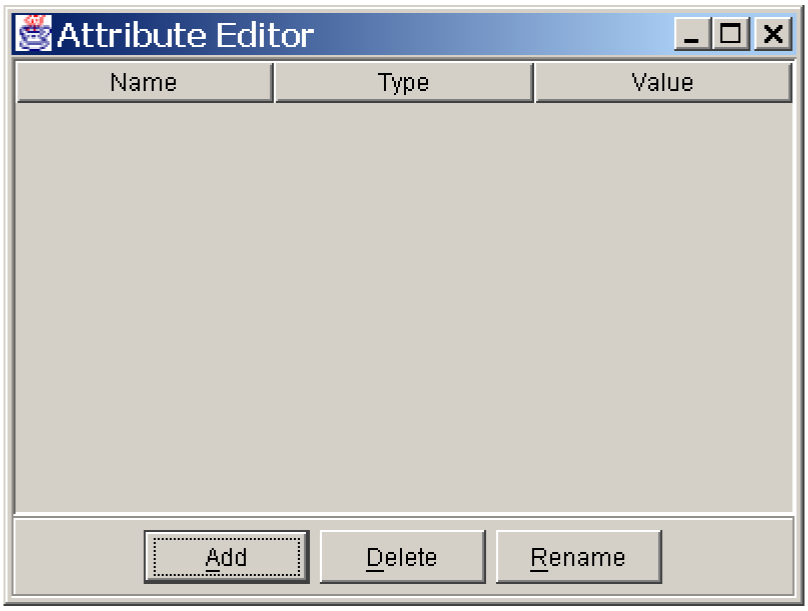


**Figure 4.70: Dialog for setting the values of the attributes.**

#### *4.4.10.2 Interaction with the Template Editor*

Another possibility to create elements is, to drag them from the Template Editor into the SMPM or SMBBM. After this they will appear and they can be used. Please take a look at the description of the Template Editor for more details.

#### *4.4.10.3 Saving a Model*

The model needs not to be saved since every change is transmitted automatically to the Simulation Manager Server and stored there.

## 4.5 Visualization

### 4.5.1 Introduction

The company VRA is a small independent architecture company with 9 employees. The core business of VRA consists of three customers segments.

**Customers segment 1: private house and small office buildings**

The requirements for this clients are normal architecture tasks like planning and production engineering. In this segment the competitors are a big number of architecture offices, which means that there is extreme competition. As an outstanding point VRA offers the 3D planning methods with the advantage of the visualization technology which includes photorealistic, textured and lightened environments. In this customers segment the visualization skill is a unique selling point and used for acquisition of customers. Simulation aspects are not relevant in this segment.

Customer samples are municipalities, builders companies and private family houses

**Customers segment 2: smaller and middle industrial companies**

The requirements for this clients are also the full architecture services but also visualization and simulation of 3D models and logistic processes. In the area of small and medium industrial companies principal competitors are big architecture offices and engineering companies. The common planning processes in these companies are mostly 2D. In this field VRA differs from their competitors with 3D planning and the ability to visualize all movements by simulation, connected via the MMP. In the visualization competitors could be found at graphic companies, which often make 3D graphs and graphics only for final presentations of projects. With the consequently developed 3D planning method of VRA, which is based on optimised 3D objects and textures, VRA can provide real time visualization for presentation and planning decisions very easily during the whole developing process. VRA does not provide logistics simulation themselves, therefore VRA works with the present network partners like the Frauenhofer IPK as subcontractors, with the necessary know how for simulation processes. The advantage for the customers exists in the all in one company VRA which allows a more efficient and easier working process.

Customers samples are the Logistikzone Tirol, Multivac and the Plansee AG.

**Customers segment 3: Global players**

The requirement for this clients is especially VRA's know how in 3D-planning combined with visualization and simulation of logistic processes for the development of their internal future evolution of factories and distribution systems. In this segment the competitors are special university institutes which work in the area of VR visualization and simulation as well as internal departments.

The sales opportunities of VRA clearly are the acquisition together with Frauenhofer or Universities as well as the cooperation with the internal special departments of global players and companies like vr-com which provide the necessary software for the presentation on power walls and Cave's.

As VRA will not be able to acquire industrial planning projects in the typical size global players will request, VRA need to integrate into a consultancy network where they can offer their unique skills and tools at an adequate price.

Customers samples are the Robert Bosch GmbH, Daimler Chrysler AG and Boeing Europe.

VRA became involved into the MISSION project out of the general company strategy. VRA started from the beginning exclusively with 3D planning and CAD methods in the core business of industrial and architecture plants and engineering with high end computing for animation, real time rendering and virtual processing on graphics workstations. (UNIX, - NT- and _MacOS- compatible).

Out of this strategy VRA got in contact with the Fraunhofer Institute IAO in Stuttgart, which introduced VRA to the Fraunhofer Institute IPK in Berlin.

The decisive project for starting planning virtual factories came from the Plansee AG company in the year 1997. In this project VRA made a 3D planning of a virtual factory with the production line, which was animated with a process simulation based on inventor files. This model was also adapted for the Cave in

Stuttgart (FHG/IAO). Therefore the Fraunhofer Institute IAO first was interested on VRA and introduced VRA to the Fraunhofer Institute IPK.

One of VRA's roles in the MISSION project was the experience in building 3D-models of quality from wireframe up to photorealistic textured and lightened environments and the needs of industrial customers beyond 3D-visualisation, for integrating movements and processes of production and logistics into the computer models. The experience of VRA is based on planned and built projects by the employees of VRA for example Robert Bosch GmbH, Plansee AG, Multivac and Logistikzone Tirol, which brings a lot of know how into the project.

VRA also brought the special know-how in exchanging databases between different computer platforms and network integration by using their own created interfaces for data transfer between different software applications. Samples are

- 3D-CAD geometry interfaces (MacOS>Unix, NT>Unix) and vice versa
- 3D-CAD materials interfaces (MacOS>Unix, NT>Unix) and vice versa
- WEB Integration of 3D models (Inventor files and VRML 2.0)
- CAVE Integration of 3D models (Inventor files and FHS files; virtual design 2)
- Integration of 3D models into industrial environments by the use of digital video and textures

With this know how and the input of all MISSION partners VRA was able to develop the MISSION compliant D-MVE software to support the 3D visualization model with necessary dynamic information of the production scenario as a basic software solution.

The target of VRA is to work with the D-MVE software on project development, selling the D-MVE will help to carry the cost for this maintenance, even if selling software is not a company strategy.

### 4.5.2 Tasks in MISSION

**Choosing compatable software for modelling 3D objects and animating them:**

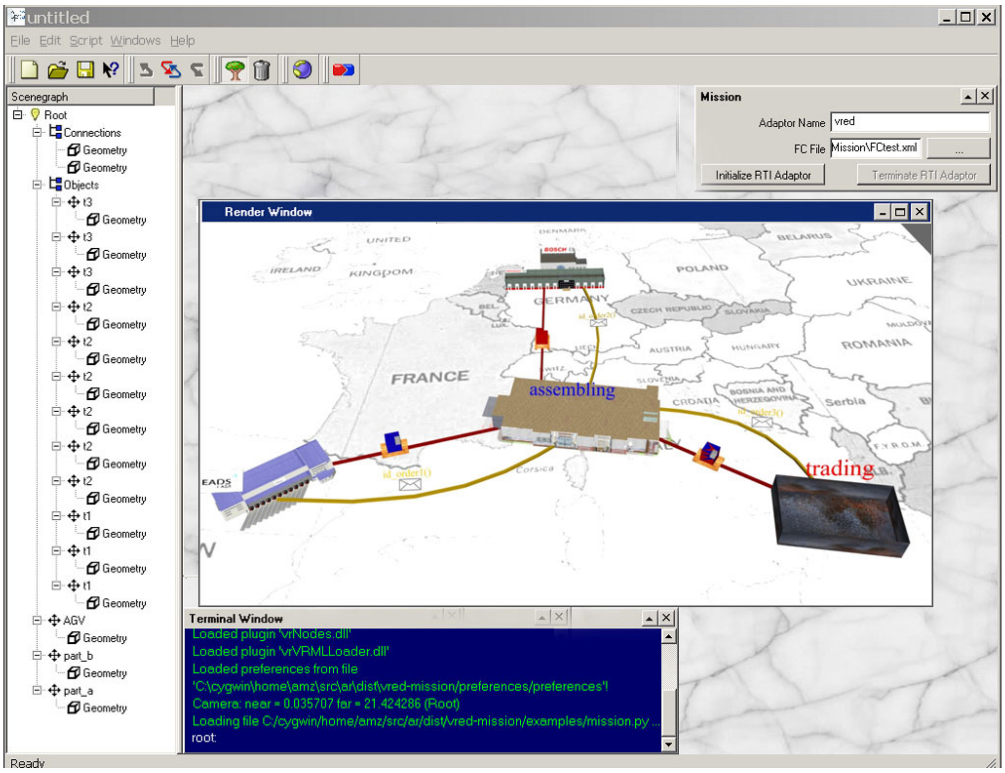
The basic modelling work was done with *Graphisoft's ArchiCAD 6.5* which was build from the ground up for architecture. It introduces the power of the integrated 3D model and object technology to the architectural design process. So it is possible to directly export geometry into the internet compatible VRML file format.

This generated objects are afterwards imported into *Cosmo Worlds 2.0*, a very simple but powerful VRML (= Virtual Reality Markable Language) and IV (= Open Inventor) editor. Unfortunately Cosmo Software, the company which produced this program doesn't exist anymore and consequently can't be developed further on. However this fact directly doesn't influence the MISSIN Project because we always worked with the same version of CW.

Cosmo Worlds is used to bring some kind of live into the 3D scenario by implementing animations. Also textures, coloured bitmap images are applied onto a geometry. This means that you control the diffuse colour of a surface on a pixel by pixel basis and so get a more realistic environment.

**Choosing a compatable viewer to connect to the HLA RTI, visualising the results of the simulators:**

During the past three years in the MISSION project a special viewer was developed, the so called *D-MVE* (= Dynamic MISSION Viewer Engine). With that software it is possible to receive the necessary events, generated from the different distributed simulators from the HLA Run Time Interface and then translated into the according movements and properties. Such properties could be the showing of helpful information (e.g. id_product, id_cycletime, colour, …) or the simple animation of products and orders. Fig. 1 shows the GUI (= Graphic User Interface) of D-MVE.



Mission

Adaptor Name: vred
FC File: Mission\FCtest.xml

Initialize RTI Adaptor    Terminate RTI Adaptor

Terminal Window

```
Loaded plugin 'vrNodes.dll'
Loaded plugin 'vrVRMLLoader.dll'
Loaded preferences from file
'C:\cygwin\home\amz\src\ar\dist\vred-mission/preferences/preferences'!
Camera: near = 0.035707 far = 21.424286 (Root)
Loading file C:/cygwin/home/amz/src/ar/dist/vred-mission/examples/mission.py ...
root:
```

**Fig. 4.71: GUI D-MVE**

From the technical point of view in the background of D-MVE  an OpenSG rendering engine is working. This portable scenegraph system is based on OpenGL and follows the open source principles.

The functions of the Dynamic MISSION Viewer Engine include

- Establishment of connections to the different software packages (e.g. simulators, template library, simulation manager,… and so on) by starting the HLA RTI software.
- Start of the D-MVE which is automatically configured by the FCF (= Federate Configuration file).
- Starting of the 3D model (e.g. Supply Chain scenario).

When the whole simulation is started, consequently the 3D scenario begins to move and react according to the simulation.

**Modelling: to create 3D models of objects and scenes within the simulation scenario, and apply textures**

The modelling work was done with ArchiCAD 6.5 and Cosmo Worlds 2.0, as explained above.

**Importing 3D models from other companies and configuring them to work with the MISSION software:**

One of the main goals of the visualisation system is to be able to import data from foreign companies and future customers. For this functionality it is only necessary that the delivered geometry (objects) is available in a VRML 2.0 format. Another requirement is that the 3D scene should not contain too much information which does note really contribute to improvement of the visual scene. For example, one model used by the project partner Bosch within one of the MISSION test cases, a PCB manufacturing line, consisted of 127,000 polygons. This results in unacceptable performance visualising the model on a regular Windows PC - not to mention a Laptop. Therefore, VRA had to reduce and optimise the model in order to get an efficient and moveable model, which then included 32,000 polygons, only.

**Creating realtime animations of the simulation scenario using the 3D models:**

After the 3D objects are imported into the D-MVE software the models are animated in real time. Depending of the situation it might be possible to have to define the animations within this software to ease the generation process of the single elements (it is relatively difficult to duplicate already animated VRML objects).

**Connecting the proposed viewer to the HLA RTI via the RTI adapter:**

The D-MVE program is equipped with an HLA RTI adapter which enables the software to receive the necessary information (events) from the distributed simulators via the RTI. It is only possible to receive information, but there is no feedback to the simulators.

The FCF file configures the MISSION interface (MISSION Adapter) adapter which supports the retrieval of class and attribute information as well as the visualisation system. Figure 2 shows the principle information flow.

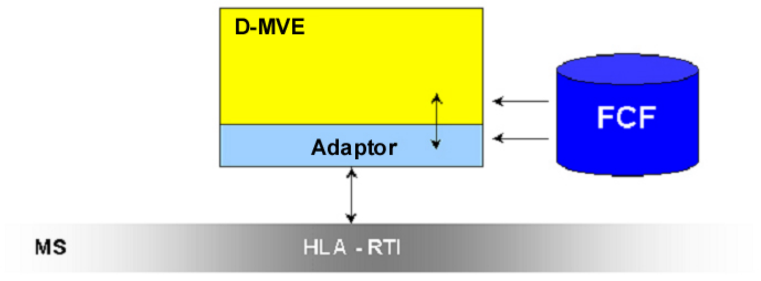


**Fig. 4.72**

Here we have got a part of a FC file example:

```
                <elements FEDfile="c:\mission\SimpleSC.FED">
                         <element name="Order">
                                 <attribute name="location"/>
                                 <attribute name="segment_ID"/>
                                 <attribute name="order_quantity"/>
                                 <attribute name="purchase_order_id"/>
                         </element>
                         <element name="Product">
                                 <attribute name="location"/>
                                 <attribute name="segment_ID"/>
                         </element>
                 </elements>
```

Using MissionShellSubscribe() all the elements are subscribed (like order, products, vehicles,… and so on). In the line `<exchange_class id="Order" name="Order">` all the objects are listed.

There is a special visualization part in the file:

```
<visualization>
        <scene file=""/>
                <simulator name="AGV" simfile="" simobj="" simparentobj=""
                simpos="" createobj="" updateobj=""/>

                <simulator name="vred" simfile="" simobj="" simparentobj=""
                simpos="" createobj="" updateobj=""/>
</visualization>
```

Scene file: is the environment wrl file (space, …). It is possible to specify an URL here.

Simulator name: is the mapping between the simulator and the visualized object.

Another important element of the D-MVE is the so called visualisation plugin. With this tool its possible to connect the visualisation system to the RTI. Underneath it is described how and which information is exchanged during the process.

<u>List of most important methods:</u>

void vrMission::initAdaptor(const char *adaptor, const char *fcfile)

→ this function parses the mentioned FCxml file and initialises the HLA adaptor over MissionshellInitialize(). In addition the objects which are specified in the XML file are registered over

MissionShellSubscribe() (e.g. orders and products); it is only possible to receive events for already registered objects.

void vrMission::terminateAdaptor(void)

→ terminates the connection to the HLA RTI

void vrMission::loop(void)

→ the loop method is regularly called (e.g. 100 times per second), depending on the duration of the scenerendering. The events are evaluated over MissionShellGetNextEvent(). At the moment there are the three types CREATED_EVENT, UPDATED_EVENT and the DELETED_EVENT.

Depending on which event is received, following methods are called:

```
void Simulator::createObj(const char *sname, const char *id, const char
*className);
```

**creates an element on the place of a simulator (e.g product)**

```
void vrSimulator::updateObjTo(const char *id, const char *name);
```

**moves an element from one simulator to the other**

```
void vrSimlulator::deleteObj(const char *id);
```

**deletes an element**

After importing the appropriate VRML file into the D-MVE an VRML defined animation is started. The Create and Delete events which are finally responsible for the number of objects are than shown as texture.

Otherwise there is  simulator class which manages the simulators including the above mentioned methods (creatObj, …), the generation of textures, number of object, the automatic placing of the simulators …

**Testing the viewer/RTI interface connection, using all the MISSION software:**

First tests with all for the visualisation system necessary software packages were conducted at VRA on 4 Silicon Graphics Windows PC's. The whole MISSION supply chain scenario was installed and executed including the BOSCH manufacturing line, the Assembly & Warehouse and the retailer. The statistical monitoring component was not tested because it was agreed at the last technical meeting in Loughborough that the collected information is shown in EXCEL or ACCESS kind. One outcome was a successful connection of the BOSCH Arena simulation to the D-MVE (elements moved from the manufacturing line to the assembly & warehouse). Unfortunately the other simulation models couldn't be tested cause of not explicable incidences. Maybe one of the downloaded files was corrupt.

**Rich Visualisation: Optimisation of the models and animation to produce a clearer more efficient final visualisation:**

At the beginning the animated and controlled elements have been simple spheres and the different simulators have been primitive cubes covered with modifiable textures, to show the information. The replacement of this simple shapes by higher sophisticated ones (Fig. 3) is the next step in the visualisation development of MISSION.

Another point is Adding of textures to get more realism into the 3D scene.

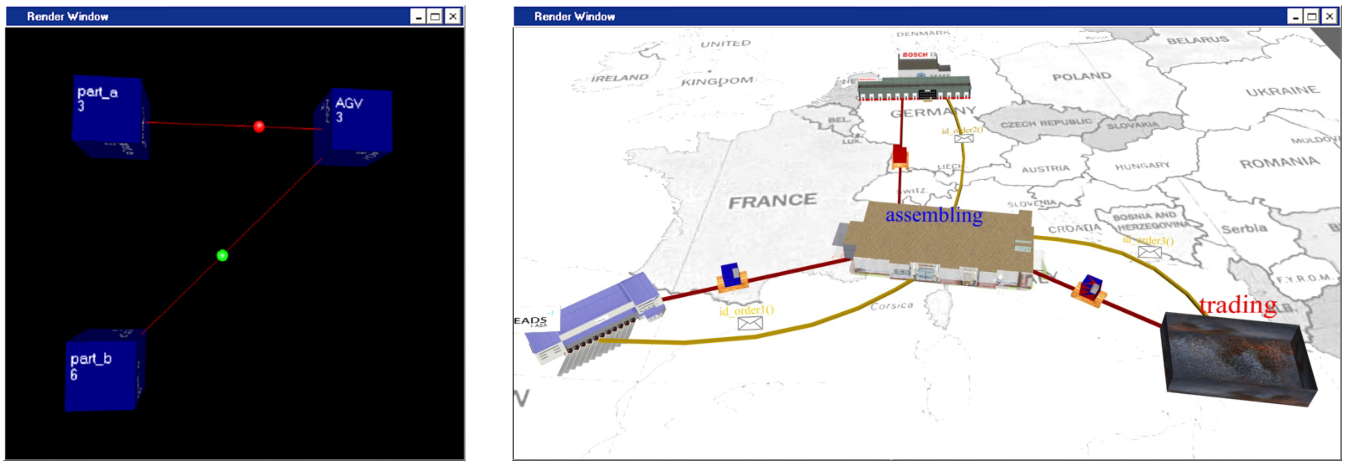


**Fig. 4.73: Comparison simple and complex scenario**

**Contributions to the Deliverable documents for CEC evaluation of progress:**

Contributions to following Deliverables had been made for the deliverables D14 and D15, which are restricted in distribution.

| | |
|---|---|
| D14<br>MMP Reference Model Libraries | • Explanation of D-MVE (Dynamic MISSION Viewer Engine)<br>• Graphic User Interface, functionalities, …<br>• Connection of the D-MVE to the HLA RTI<br>• Hardware and Software requirements |
| D15<br>MMP Reference Model Catalogue | • Description of the MISSION Supply chain model<br>• Description of the single elements (products, orders, machines, buildings, …) |

There is further information delivered to the technology implementation plan (D25), which is confidential and not explained here.

### 4.5.3 Conclusion

For the consortium, the D-MVE developed by VRA is the only chance to visualise the material and information flow between the single segments of the scene (federation). The only chance without the D-MVE is to watch traces within the monitoring components, which are quite difficult to follow.

The D-MVE gets especially important when applying the MMP to supply chain scenarios. Here the material and information flow between the elements of the chain is a very interesting point. Exactly this flow is visualised, which gives significant additional transparency to the engineers working on the supply chain.

VRA is starting a new strategic development with services which are improved through the MISSION project. Acquisition of construction building projects is now focussed on such enterprises where the MMP technology is useful. This is especially valid for automotive industry and other companies with complex production systems. One major advantage is the chance to handle architecture, production engineering, simulation and VR representation separately and then integrate them at any point of time of time where this appears useful through the MMP.

Without the MMP technology, VRA often had no chance to get in touch with interesting projects in an early phase. Now VRA provides means and methods which are needed at an early  point of time. This enables VRA to prepare the acquisition of complex construction building projects in the area mentioned.

MISSION is the opportunity for VRA to create a consistent logical way from classic CAD tools like AutoCAD and simulation tools like ARENA or Taylor into the real world of building in trade and industry. The special know-how from MISSION for VRA is to link the 3D-visulisation model with the results of the simulation scenario.

VRA profits by improving the know-how in VR technology with scope to services for globally distributed enterprises (developing international co-operation based on the MMP with view towards manufacturing systems and customers.

# 5 Demonstration Scenario

The virtual MISSION Enterprise is a global organisation with facilities distributed world-wide. A new MISSION product, a DC-Motor, is currently being designed, and the time has come to design the manufacturing system for this product. For full details see Deliverable Report D7.

This document describes the MISSION demonstration scenario, in which MISSION software and commercial software packages are integrated through the MISSION Modelling Platform (MMP) to design the MISSION Enterprise. The scenario is shown in a sequence of stages beginning with project definition and progressing through outline design of the supply chain to an eventual detailed design of manufacturing facilities and simulation of the integrated global supply chain to predict performance against project objectives.

## 5.1 Stage 1: Identification and Specification of Manufacturing System Objectives.

At this stage the main requirements from the proposed manufacturing system are identified. These include:

- Number of products to be made
- How fast products should be made
- Cost of manufacturing product
- Delivery policy (eg from stock, or within 10 days of order, or…)

Feasibility of requirements will also depend on market, technology, material supplies, etc..

### 5.1.1 Demonstration Activities in Stage 1

These requirements are set up, by the Project Agent, as part of the initial activity in creating the new project (see Figure 5.1). The Project Agent software drives a dialogue in which the project definition and objectives are entered into the database.

### 5.1.2 End-User Benefits from Stage 1

There are no direct benefits from Stage 1. However this stage sets up the project infrastructure to make possible the benefits achievable from later stages.

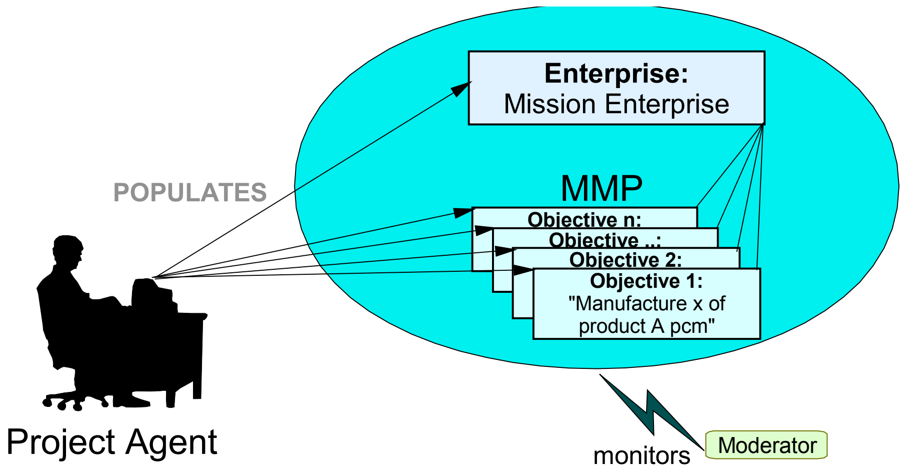


**Figure 5.1: Identification and specification of manufacturing system objectives.**

## 5.2 Stage 2: Identification of Facilities to be used in Manufacturing System.

The components of the MISSION product are assumed to be supplied by the specialist MISSION manufacturing shops in Spain (CASA), Germany (BOSCH), Japan (DENSO) and USA. It is also assumed that assembly of components will take place in a facility located in Germany (Bosch). But, in order to test the feasibility of using these facilities in the manufacturing system, the capabilities of these facilities need to be checked against the specification of the manufacturing system objectives. Strategies for the operation of both the individual parts of the manufacturing system and of the manufacturing system as a whole, also need to be decided in Stage 2.

### 5.2.1 Demonstration Activities in Stage 2

We apply the Manufacturing Process Design Agent (MPDA) to select the facilities, and the Supply Chain Agent (SCA) to determine an interfacility transport network and inventory policy. To do so, it is necessary to both specialise and instantiate templates from the Template Library, and then embed these within the database. This is illustrated in figure 5.2.

### 5.2.2 Potential Design Conflicts at Stage 2

*Conflict 1: Occurs if the MPDA selects a facility which is incompatible with the transport mechanism to serve it as selected by the SCA. (eg 1: lot quantities required are to small to justify sea freight; eg 2: rail transport selected for a facility with no rail access)*

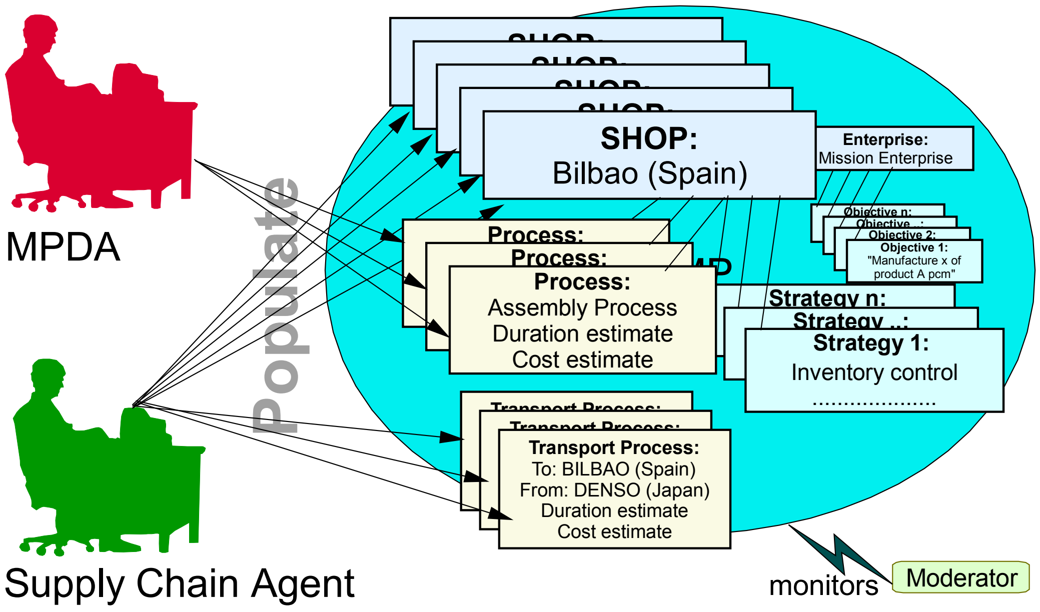


**Figure 5.2: Identification of facilities to be used in the manufacturing system.**

The MSE Moderator has been monitoring design changes since the start of the design activity. However, it has not been visible up to now, since no real conflicts have occurred. It now makes its first appearance. If a transport method is selected which is incompatible with either the lot sizes appropriate to the facility, or the availability of access to that transport method, it can signal the conflict to both agents (MPDA and SCA).

### 5.2.3 End-User Benefits from Stage 2

1. In a commercial implementation the MPDA can provide functionality to access and re-use knowledge about plant capabilities defined in previous projects. The benefit of this is knowledge capture to reduce the effort and skill required for future projects.
2. Moderation can prevent further development of an impracticable design.

## 5.3 Stage 3: Simulation of Facility Level Design

This is a high level, simple simulation based on estimates. It treats individual facilities as black boxes, with inputs and outputs. The activities within individual facilities are not simulated at this stage. It therefore uses:

- Estimates of throughput quantities and times
- Estimates of transport times
- Inventory policy.

The outputs from this simulation should enable performance prediction (in terms of throughput levels and inventory).

### 5.3.1 Demonstration Activities at Stage 3

The Simulation Agent works with the Template Library Manager to construct a distributed simulation based on the templates selected to represent facilities. The Simulation Agent then defines and executes an experimental protocol to predict performance measures consistent with project objectives. Results from simulation include predictions of performance measures comparable with project objectives. Results are retained in the Manufacturing System Design Database (MSDD) so they can be retrieved for future reference as the project develops (see Figure 5.3).

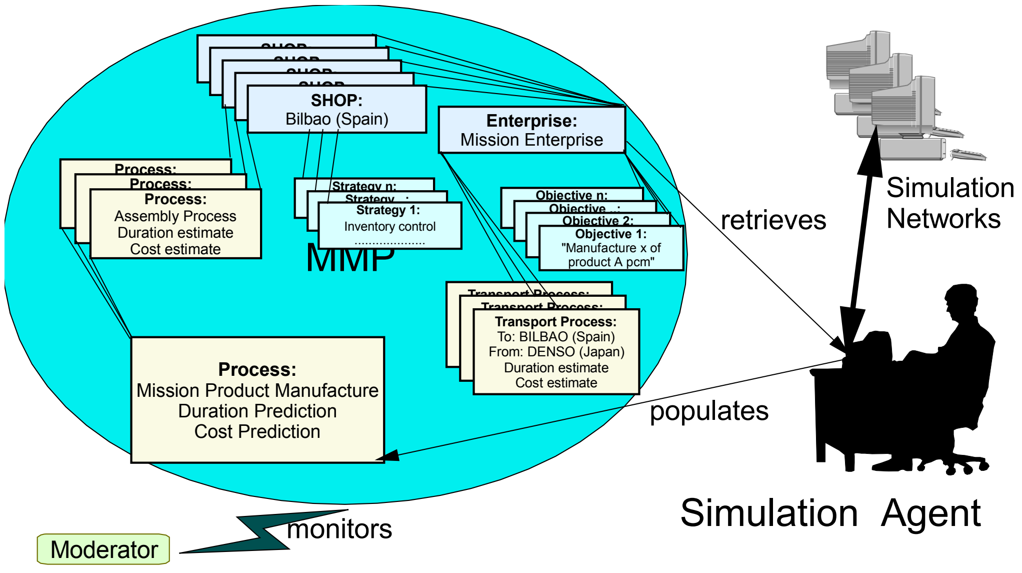


**Figure 5.3: Simulation of facility level design**

### 5.3.2 Potential Design Conflicts at Stage 3

*Conflict 2: Performance prediction does not meet specification requirements.*

The MSEM interprets the performance predictions added to the design in terms of the project objectives. In order to accomplish this the MSEM must have knowledge of how to retrieve comparable objectives from the MSDD, and MSEM knowledge can be sufficient to determine which confidence limits on which measures are critical either singly or in combination. For example it may be that the confidence limits on manufacture of a component are perfectly acceptable, and the limits on transport from the manufacturing facility to the assembly facility are acceptable, but that when the two distributions are convoluted, the joint confidence limits are not acceptable – there is a conflict between MS design performance as simulated, and MS requirements as defined for the project. Understanding the relationships between limits on measures is not trivial, even in the relatively simple scenario of the MISSION enterprise at facility level. Once a conflict is

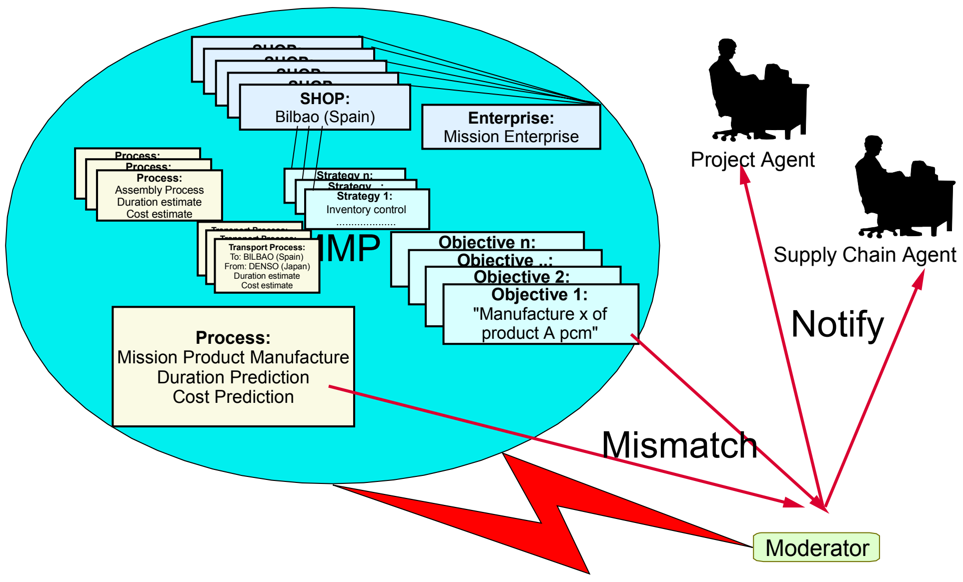


**Figure 5.4: Conflict Identification**

identified, the MSEM must have knowledge of which agents are interested in resolving the conflict, and how to inform these agents that potential conflict has arisen. This is illustrated in Figure 5.4.

### 5.3.3 End-user Benefits from Stage 3:

1. The capability to evaluate the predicted performance of the supply chain, including the supply chain control system given assumptions about the facility capabilities, before detailed design of the manufacturing systems within the facilities.
2. The capability to automatically undertake possibly complex comparisons between predicted and required performance, and to alert concerned MSE Agents where conflict arises between these.
3. The structure of the global supply chain and its interconnections remains defined in the MSDD for use in future detailing of the design – it does not need to be re-entered in a later design or model.

## 5.4 Stage 4: Individual Facilities – Design and Simulation

The individual facilities are modelled, and a detailed design of each facility is produced. At this stage, decisions are made on internal policy, for example, lot sizing. Simulations of the individual facilities are run, using best estimates of performance times for particular operations. The outputs from this simulation enable the performance of the facility to be predicted.

### 5.4.1 Demonstration Activities at Stage 4

This is similar to stages 2 and 3 but at the level of individual facility. An MPDA peculiar to the particular facility and its operating company selects, specialises and instantiates appropriate templates to represent the resources to be employed in the facility, whilst the Material Flow Agent for the facility determines control policies to be applied. There may be interaction with a CAD system through a Factory Layout Agent. This latter will probably consist of a human, a CAD system and a dialogue window to pass transport distances back to the Information Manager.

### 5.4.2 Potential Conflict at Stage 4

*Conflict 3: lot sizing and global inventory policy do not match.*

The MSE Moderator can detect conflicts in internal manufacturing policy and global policy: for example a lot size selected to be locally optimum will probably conflict with a global optimum. The global Supply Chain Agent and the facility Material Flow Agent are informed of conflict and can attempt compromise.

*Conflict 4: Performance predictions for individual facilities do not match global estimates used.*

The MSEM can identify when the performance prediction of a facility falls outside the sensitivity limits of the overall model (in terms of confidence). A similar conflict would arise if design and simulation at the facility level is done in terms of measures not relevant to the overall level, so the local is being designed to meet criteria which do not contribute to the objectives of the complete MS.

*Conflict 5: Internal conflicts can be generated within a facility MS design. For example the replacement in the MS design of one machine by another with a different footprint may impact on layout; changing the layout may impact on transport times; changing transport times may impact on facility performance, etc..)*

### 5.4.3 End-user Benefits from Stage 4

These include, but are not limited to:

1. Re-use of global design from stage 2 saving some input effort.
2. Rapid recognition of failure of design to meet project requirements.
3. Use of templates to simplify and accelerate facility simulation model construction.
4. Incorporation of existing simulators to accelerate simulation model construction.
5. Recording of simulation results for future reference.
6. etc…

## 5.5 Stage 5: Transport network and Global Model – Design and Simulation

An outline design of the complete manufacturing system, including design estimates for transport network. Each individual facility is represented as a black box, so simulations of the global model can be used to predict transport performance.

### 5.5.1 Demonstration Activity at Stage 5

This is similar to Stage 4, but concentrates on application of transport system templates.  This is a  Supply Chain Agent activity, now working in more detail on transportation planning.

### 5.5.2 Potential Conflict at Stage 5

*Conflict 6: Transport network performance confidence limits lie outside global requirements.*

### 5.5.3 End-User Benefits at Stage 5

These include, but are not limited to:

1. Re-use of global design from Stage 2 saving some input effort.
2. Rapid recognition of failure of design to meet project requirements.
3. Use of templates to simplify and accelerate transport simulation model construction.
4. Incorporation of existing simulators to accelerate simulation model construction.
5. Recording of simulation results for future reference.

## 5.6 Stage 6: Global Simulation

A detailed design of the entire manufacturing system, including performance predictions for transport network. This simulation is used to predict detailed performance of complete manufacturing system design.

### 5.6.1 Demonstration Activity at Stage 6

The SCA and MDSA are used to assemble a fully detailed design of the global manufacturing system. The Simulation agent constructs a distributed simulation model of the detail design. Running of this simulation generates the most accurate available predictions of system behaviour and performance, which are retained in the MSDD, for future reference and for comparison with requirements.

### 5.6.2 Potential Conflict at Stage 6

*Conflict 7: Any aspects of predicted detailed performance with any element of the manufacturing system specification.*

This is more of the above but with much more complex interaction and analysis of confidence limits on predictions. Also there may be many more measures of performance (some local, some global).

### 5.6.3 End-User Benefits at Stage 6

These include, but are not limited to:

1. Re-use of global design from Stage 2 and detailed facility designs from stages 4 and 5, saving some input effort.
2. Rapid recognition of failure of design to meet project requirements.
3. Use of templates to simplify and accelerate distributed simulation model construction.
4. Incorporation of existing simulators to accelerate simulation model construction.
5. Recording of simulation results for future reference.

# 6 Dissemination Activities

## 6.1 Conferences

### 6.1.1 ASIM Dedicated Conference on Simulation in Production and Logistics, 2000

Rabe, M.: Future of Simulation in Production and Logistics: Facts and Visions. In:Mertins, K.; Rabe,M. (Hrsg.): The New Simulation in Production and Logistics. 9. ASIM - Fachtagung Simulation in Produktion und Logistik, Berlin. Eigenverlag, 2000, S. 21-43.

Rabe,M.; Friedland, R.: Neutral Template Libraries for Use in Globally Distributed Enterprises. In: Mertins, K.; Rabe, M.(Hrsg.): The New Simulation in Production and Logistics. 9. ASIM- Fachtagung Simulation in Produktion und Logistik, Berlin, Eigenverlag, 2000, S. 385-394

Harding, J.A., Popplewell, K.: Simulation Modelling Agent: An Aid to Enterprise Design and Performance Evaluation. In: Mertins, K.; Rabe, M. (Hrsg.): The New Simulation in Production and Logistics. 9. ASIM - Fachtagung Simulation in Produktion und Logistik, Berlin. Eigenverlag, 2000

McLean, C., Riddick, F., Leong, S.: Architecture for Modelling and Simulation of Global Distributed Enterprises. In: Mertins, K.; Rabe, M.(Hrsg.): The New Simulation in Production and Logistics. 9. ASIM- Fachtagung Simulation in Produktion und Logistik, Berlin, Eigenverlag, 2000.

Jäkel, F.W.; Arroyo Pinedo, J.S.: Development of a Demonstrator for Modelling and Simulation of Global Distributed Enterprises. In: Mertins, K.; Rabe, M.(Hrsg.): The New Simulation in Production and Logistics. 9. ASIM-Fachtagung Simulation in Produktion und Logistik, Berlin, Eigenverlag, 2000.

### 6.1.2 Winter Simulation Conference, 2000

McLean, C.; Riddick, F.:The IMS MISSION Architecture For Distributed Manufacturing Simulation. In: Joines, Barton, Kang, Fishwick (eds.): Proceedings of the 2000 Winter Simulation Conference, pp. 1539-1548.

Mertins,K.; Rabe,M; Jäkel,F.W.: Neutral Template Libraries for Efficient Distributed Simulation within a Manufacturing System Engineering Platform. 2000 Winter Simulation Conference (WSC), Orlando (USA), December 2000 (ESS), Hamburg, September 2000, pp. 1549-1557.

Son, Y.S., Jones, A.T.; Wysk, R.W: Automatic Generation Of Simulation Models From Neutral Libraries: An Example. In: Joines, Barton, Kang, Fishwick (eds.): Proceedings of the 2000 Winter Simulation Conference, pp. 1539-1548.

### 6.1.3 EUROSIM 2001

Rabe, M.; Garcia de Gurtubai, G.; Jaekel, F.-W.:Modelling and Simulation for Globally Distributed Enterprises. In: Proceedings of the EUROSIM 2001 conference, Delft 2001.

Lee, Y.T.; Umeda, S.: Information Model of the Supply Chain Simulation. In: Proceedings of the EUROSIM 2001 conference, Delft 2000.

Nakano, M.; Kubota, F.; Yura, Y.; Arai, E.: Design Agent system for the Collaborative Engineering Process in Manufacturing Systems. In: Proceedings of the EUROSIM 2001 conference, Delft 2001.

Popplewell, K.; Harding, J.A.: A Manufacturing Systems Engineering Moderator - Identifying Conflicts in the Design of Distributed Manufacturing Systems. In: Proceedings of the EUROSIM 2001 conference, Delft 2001.

### 6.1.4 Conferences of the Society of Computer Simulation International (SCS)

Rabe, M.; Jäkel, F.-W.: Simulation for Globally Distributed Enterprises. 12th European Simulation Symposium (ESS), Hamburg 2000, pp. 1549-1557.

### 6.1.5 Further Conferences

Rabe, M.; Jäkel, F.-W.: Non Military Use of HLA within Distributed Manufacturing Scenarios. Simulation and Visualisierung 2001, Magdeburg 2001, S. 141-150.

J. Yagi; Susumu Fujii; Masaru Nakano; Hironori Hibino; Eiji Arai; Fumio Kojima; Yoshiro Fukuda; Nobuhiro Sugimura; Soichiro Isago: IMS MISSION PROJECT -MISSION Modelling Platform. In:Proceedings of 2000 Pacific Conference on Manufacturing, Detroit (Michigan, USA), September 2000

Popplewell,K.; Harding J.A.: Enabling Information Technology for Virtual Distributed Enterprises. Industrial Engineering Research '99 Conference (Conference of the Institute of Industrial Engineers), Phoenix, Arizona, USA, May 1999.

## 6.2 Exhibitions, Workshops and Fairs

### 6.2.1 Workshops

#### *6.2.1.1 HLA Workshop Magdeburg, 2000*

Markus Rabe and Frank-Walter Jaekel (IPK) presented the MISSION project and discussed with related projects. The presentation was published as:

Rabe, M.: Einsatz von HLA für die Verbindung bestehender Referenzmodelle in Produktion und Logistik. HLA-Forum am 03.03.1999, Magdeburg. ASIM Mitteilungen aus den Arbeitskreisen, Heft Nr. 64, 2000.

Michael Jenewein (VRA) presented the MISSION project and discussed with potential users at the following workshops:

Workshop “Architecture and /or Design”, 27.06.2001 Innsbruck (Austria), organized by A-Null EDV GmbH and Graphisoft.

BTV – Bauforum, 08.11.2001 Reutte (Austria), organised by the Bank for Tirol and Vorarlberg

### 6.2.2 Fairs

#### *6.2.2.1 Presentation of the MISSION Commercial Product: GLUE 4D*

In order to promote the idea of MISSION and receive significant feedback from potential users, Sisteplant has presented (and is presenting yet) the MISSION platform at the following exhibitions:

| Fairs held during year 2000 | | |
|---|---|---|
| Fair name | Dates | Location (in Spain) |
| MANAGING Certamen Internacional de Innovación en la Gestión Empresarial | May 17-19 | Bilbao |
| SIL Salón Internacional de la Logística | June 13-16 | Barcelona |
| INNOVA | October 3-5 | Murcia |
| Salón del Mantenimiento | November 28-30 | Barcelona |

| Fairs held during year 2001 | | |
|---|---|---|
| Fair name | Dates | Location (in Spain) |
| SIL Salón Internacional de la Logística | June 12-15 | Barcelona |
| Cumbre Internacional | September 16-29 | Bilbao |
| MANUTEC Salón de la Manutención y el Almacenaje | October 3-6 | Madrid |
| FIMMA, Feria Internacional de Maquinaria para la Madera | November 23-27 | Valencia |

| | | |
|---|---|---|
| MAQUITEC Feria Industrial | December | Barcelona |

VRA has presented MISSION at the following exhibitions:

| Fairs held 2000/2001 | | |
|---|---|---|
| Fair name | Dates | Location |
| Allgäutech | February 2000 | Kaufbeuren (Germany) |
| Hannover Messe Industrie (in cooperation with IPK) | March 2000 | Hannover (Germany) |
| Messe Friedrichshafen | November 2000 | Friedrichshafen (Germany) |
| Frühjahrsmesse | March/April 2001 | Innsbruck (Austria) |

##### *6.2.2.2 Exhibition at Dedicated ASIM Conference on Simulation in Production and Logistics*

VRA exhibited the MISSION and their goals and services applying MMP at the exhibition. Details are available from http://www-plt.ipk.fhg.de/asim-fachtagung/engausstellung.htm.

## 6.3 Further Publications

Rabe, M.; Jaekel, F.-W.: Einsatz von HLA für die Verbindung bestehender Referenzmodelle in Produktion und Logistik. From: ASIM Mitteilungen aus den Arbeitskreisen, Heft Nr. 64, 2000.

## 6.4 Open MISSION Interest Group

### 6.4.1 Purpose

The Open MISSION interest group (OMIG) provides the opportunity to discuss the projects vision and findings with a broader range of external companies of different kinds and sizes as well as research institutes working in related projects. The knowledge about concepts, specifications and implementations developed in MISSION gets further propagated by the OMIG.

Within the OMIG companies and interested researchers get detailed information about the project, the developed concepts and the project's process. The MISSION project received from the OMIG members additional ideas, feedback and intense contact to companies working in the field of simulation.

The membership in the OMIG is free.

### 6.4.2 Foundation

The foundation session of the OMIG took place on March 9, 2000 at the Production Technology Centre of Berlin. There have been 40 persons participating in the foundation session, which was organised in the framework of the 9th ASIM Dedicated Conference on Simulation in Production and Logistics. In addition, the foundation of the OMIG was announced with written information and through the MISSION web site.

### 6.4.3 Members

Most members of the OMIG have joined within the foundation session. However, the demonstration activities of the partners as well as the internet page of MISSION have attracted several additional companies to join the OMIG. In total, OMIG has more than 40 members now, without counting the international partners of the IMS MISSION project.

### 6.4.4 Meetings

After the foundation meeting in 2000, there was an OMIG meeting in Delft in June 2001. The next meeting is planned for March 2002 to occur in Duisburg, in the framework of the 10th ASIM Dedicated Conference on Simulation in Production and Logistics.

### 6.4.5 Further Activities

The consortium plans to keep the OMIG running. This will help to support the product development with feedback and new ideas as well as to establish new projects exploiting the benefits of MISSION.

## 6.5 University Courses and Lectures

### 6.5.1 Loughborough University

LU disseminates the results of MISSION mainly through one-to-one discussions with research students. This is done within the Wolfson School of Mechanical and Manufacturing Engineering.

Four PhD students are directly using aspects of the Mission project in their research. These are:

- A H M Shamsuzzoha
- S Dani
- E I Neaga
- H K Lin

Dr. Harding has also discussed the Mission Research with several other research students whose research relates in some way to information modelling or manufacturing system design, and with a visiting academic from Malaysia.

### 6.5.2 Coventry University

| | | |
|---|---|---|
| WS 2000/01 | Research Seminar on MISSION. | Academic staff and research students of School of Engineering, Coventry University |
| WS 2000/01 | Lecture on MISSION research and projected impact on future manufacturing systems engineering | Students on MSc courses in School of Engineering, Coventry University |
| SS 2001 | Research Seminar on MISSION and other current research | Senior manufacturing management of Jaguar Cars |

* Likely to be included in up to four courses per annum for the foreseeable future.

### 6.5.3 Technical University Berlin

| | | |
|---|---|---|
| WS 2000/01 | Production Technology (German language) 2 Lectures on Simulation | Students of regular Production Technology Diploma Studies |
| WS 2000/01 | "Global Production Management" (English language) Lectures on Manufacturing Engineering | Students of International Postgraduate Course "Global Production Engineering" |
| SS 2001 | "Global Production Management" (English language) Lecture on Global Research | Students of International Postgraduate Course "Global Production Engineering" |

### 6.5.4 Deusto University & Basque Country University, Bilbao

As Mission is considered one of the most important research projects in SISTEPLANT, not only Javier Borda but all people in SISTEPLANT involved in training and with relationship with Universities, have included the focus of Mission in the topics of their lectures:

| | | |
|---|---|---|
| 2000/2001 | Javier Borda: Professor of Logistics and Production Management in "Universidad Comercial de Deusto" | Students of Logistics and Engineering at University of Deusto (Bilbao) |
| 2001 | José M. Borda: Lecturer on Maintenance Management Systems and integration with other Management Tools. | Students of MBA Post graduate course at University of Deusto (Bilbao) |
| 2001 | Alberto García de Salazar. Lecturer on logistics management and integration with productive processes | Students of Engineering at Escuela Superior de Ingenieros Industriales y Telecomunicaciones (Bilbao) |

### 6.5.5 Fachhochschule Vorarlberg (Austria)

Lectures have been performed by vr-architects.

| | | |
|---|---|---|
| 1999 | Michael Jenewein: Logistics simulation with virtual Reality. | Regular Students, R&D Course "Technics" |