DR9.6a: Specification of the System Object Model (revised)

Written by:
Jacopo Cassina, Maurizio Tomasella, POLIMI
Altuğ Metin, Michael Marquard, InMediasP

DELIVERABLE NO	DR9.6a: Specification of the System Object Model (revised)
DISSEMINATION LEVEL	**PUBLIC**
DATE	15.05.2007
WORK PACKAGE NO	WP R9: Development of PROMISE Information management system
VERSION NO.	1.0
ELECTRONIC FILE CODE	DR9.6a.1.0.Doc
CONTRACT NO	507100 PROMISE A Project of the 6th Framework Programme Information Society Technologies (IST)
ABSTRACT	In the present document, an object model describing the attributes of objects required for the representation of the data handled by the PROMISE application scenarios such as BOL, MOL, and EOL product structures and field data and their relations are described in detail. The model primarily addresses semantic issues, i.e. comprehensible representation of domain knowledge for communication between users and applications, but discusses also technical issues, i.e. the efficient storage of data in physical databases. Basically a copy of deliverable DR9.2 “Specification of the System Object Model” that referred to version 1 of the PDKM, the present deliverable updates the model based on the experiences with working with the model.

STATUS OF DELIVERABLE		
ACTION	**BY**	**DATE** (dd.mm.yyyy)
SUBMITTED (authors)	Maurizio Tomasella, Jacopo Cassina[1], Altuğ Metin, Michael Marquard	06.06.2007
VU (WP Leader)	Andreas Edler	06.06.2007
APPROVED (QIM)	D. Kiritsis	06.06.2007

[1] The two authors from POLIMI did not work on DR9.6a itself. However, they were main contributors to DR9.2, which is revised with the present deliverable. Hence, they are listed as authors as well.

Revision History

Date (dd.mm.yyyy)	Version	Author	Comments
09.03.2006	0.1	Michael Marquard	Document initiated (copy of final version of DR9.2 [7])
31.05.2007	0.2	Michael Marquard	Changes all over the document
01.06.2007	0.3	Altuğ Metin	Additions and changes to data model diagrams
04.06.2007	1.0	Michael Marquard	Changes all over the document; document finalised for review

Authors' contact information

Name	Organisation	E-mail	Tel	Fax
Jacopo Cassina	POLIMI	jacopo.cassina@polimi.it	+39 02 2399 3951	+39 02 70638377
Maurizio Tomasella	POLIMI	maurizio.tomasella@mecc.polimi.it	+39 02 2399 4898	+39 02 70638377
Michael Marquard	InMediasP	promise@inmediasp.de	+49 3302 559 420	+49 3302 559 124
Altuğ Metin	InMediasP	promise@inmediasp.de	+49 3302 559 420	+49 3302 559 124

Table of Contents

List of figures

Abbreviations

Abbreviations used in this document:

AAM	Application Activity Model
AIC	Application Interpreted Construct
AM	Application Module
ANSI	American National Standard Institution
AP	Application Protocol
API	Application Programming Interface
AR	Application Resources
ARM	Application Reference Model
ASPI	Assured Product and Support Information
B2MML	Business To Manufacturing Markup Language
BOL	Beginning Of Life
BoM	Bill of Material
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CAM	Computer Automated Manufacturing
CAx	Computer Aided x
CORBA	Common Object Request Broker Architecture
DoW	Description of Work
DSS	Decision Support System
EDS	Electronic Data Systems Corporation
EOL	End Of Life
IAR	Integrated Application Resource
ICAM	Integrated Computer-Aided Manufacturing
ID	IDentifier
IDEF	ICAM Definition Languages
IE	inter-enterprise
IGR	Integrated Generic Resource
ISA	Instrumentation, Systems, and Automation Society
ISO	International Organization for Standardization
MANDATE	MANufacturing DATa Exchange
MIM	Module Interpreted Module
MOL	Middle Of Life
NC	Numerical Control
NURB	Non Uniform Rational B-spline
OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group
PDKM	Product Data & Knowledge Management
PDM	Product Data Management

PEID	Product Embedded Information Device
PLCS	Product Life Cycle Support
PLM	Product Life Cycle Management
PLM XML	A standard on Product Life Cycle Management using XML (see below)
PROMISE	PROduct life cycle Management and Information tracking using Smart Embedded systems
SDAI	Standard Data Access Interface
STEP	STandard for the Exchange of Product model data
UML	Unified Modelling Language
UoF	Unit of Functionality
WBF	World Batch Forum
WP	PROMISE work package
XML	eXtended Mark-up Language
XSD	XML Schema Definition language

1 Introduction

1.1 Work Package objectives

WP R9 "Development of PROMISE Information management system" is part of Research cluster RC-4, Knowledge Treatment and Decision Making, and is dedicated to the design, specification, and implementation of the Product Data & Knowledge Management system (PDKM). As specified in the DoW [1] of the PROMISE project, the roadmap to this goal consists of several tasks and is also closely integrated with the tasks of WP R10 "Design and implementation of an integrated and personalized user interface for the PDKM system".

The first version of the present document, deliverable DR9.2 "Specification of the System Object Model" [7], was outcome of task TR9.2 of WP R9. It was revised in task TR9.6, leading to the present deliverable DR9.6a.

1.2 Scope of deliverable DR9.6a

This document presents an object model describing the attributes of the required objects of the PDKM system, as well as their relations in detail. The object model primarily addresses semantic issues, i.e. comprehensible representation of domain knowledge for communication between users and applications, but also discusses technical issues, i.e. the efficient storage of data in physical databases. A major emphasis is given to semantic issues, modelling "*the entities which the business users are familiar with and can easily communicate about. Entities (…) (are) further described by attributes and relationships to other entities and business terms used in this context. (…) (Such) a semantic model offers a common basis for data communication and exchange between applications, between users, as well as between users and applications"* [2].
DR9.6a does not present the complete data model as required by the PDKM, but is explicitly focused on the pieces of information that the application scenarios are interested in. A certain level of abstraction was required in the modelling process, in order to meet all their requirements.

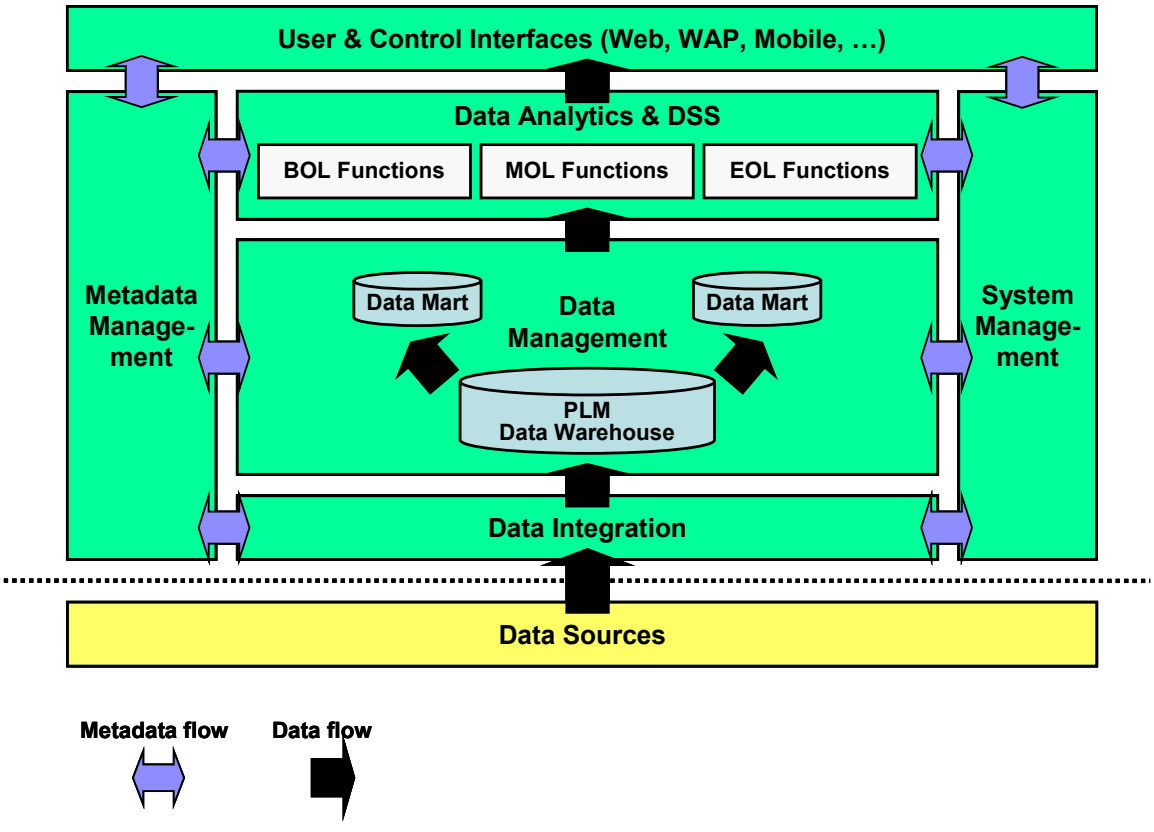

Figure 1: Architecture overview of the PDKM system

Referring to the PDKM components of the PROMISE PDKM architecture shown in figure 1, the present semantic data model describes the core of the Data Management layer, whose main task is to provide a global semantic view on product and product life cycle data for all analysis applications.
Examples of PDKM components not covered by the semantic model are the System Management tower and the Metadata Management tower, which is treated in the development of the technical data model in tasks TR9.5, TR9.11, and TR9.13.
Moreover, this deliverable does not intend to provide the data model that is required by the DSS e.g. to store algorithms even if the DSS is, according to DR9.1 "Design of PROMISE Information Management System (PDKM)" (see also figure 1), one of the components of the PDKM. That part of the data model is developed by WP R8 "Methodologies for decision making for BOL, MOL, EOL".

1.3 Overall approach

The first step to the definition of the semantic object model of the PDKM system was the analysis of user requirements defined by all of the application scenarios. A detailed description of the inputs that have been used for this, and how they have been used, is presented in chapter 2.
At the same time, an analysis of the relevant industrial standards in the field of product life cycle data modelling was carried out. This study provided the partners working on task TR9.2 with many useful ideas for the development of the semantic model.
Thereafter, a first draft of the model was developed during a workshop. Then the application scenarios have been iteratively mapped to the draft model, by this improving it. The final version was developed and checked during another workshop and is reported in the present deliverable. Finally, also the mapping of three exemplifying application scenarios to the semantic model is documented in chapter 6.

1.4 Boundary conditions and limitations

The present deliverable DR9.6a focuses on the Semantic Object Model. The technical data model is only discussed to the extent necessary for the completion of this work package, but important aspects are summarized in chapter 5.

1.5 Document overview

The rest of this document is organized as follows:

- In chapter 2, the connection to other work packages of the PROMISE project is outlined. Some of them provided input about the domain to be covered by the model (e.g. WP A1 to WP A11). Some gave some constraints of different strengths (e.g. WP R7). Some provided reference frameworks, to which the work developed in tasks TR9.2 and TR9.6 had to be compliant (e.g. WP R2).
- In chapter 3, an overview of existing industrial standards for product data modelling across the life cycle is presented. The aim is to identify interesting standards to refer to in the construction of the semantic data model. The most interesting standards are briefly outlined and their main features described. The compatibility of the semantic model with these approaches is considered to be of great impact both from a conceptual point of view and from an interoperability point of view, where comparing the approach of the semantic object model with that of already existing solutions.
- Chapter 4 outlines the semantic object model. Before entering a detailed description of the model, with its classes, attributes and associations, a brief introduction on how the model was developed is provided, in order to outline the used point of view, the pieces of information modelled, and the relations with other key PROMISE issues. Then, an overview on the main classes and relations among them is given. Finally, each class of the

model is presented with its most important attributes, and a more detailed description of the associations with other classes of the model is provided.

- Chapter 5 provides important details on the technical data schema, relating it both to the mySAP PLM software chosen in previous deliverables as the system to be customized for the purposes of the PROMISE PDKM, and to some of the most important technical issues to be considered during implementation work.
- In chapter 6, the semantic data model is mapped to three of the PROMISE application scenarios, providing a first overview of, which are its modelling capabilities for the project's purposes. The compliance of the model with the needs of the different application scenarios represented a big issue to be solved during the development phase of the model: a first exposition of the modelling concepts to solve these issues is given in this chapter with reference to the three chosen scenarios.
- Chapter 7 provides some concluding remarks on the work developed in task TR9.2 and TR9.6.

1.6 Delta to deliverable DR9.2 "Specification of the System Object Model"

The present document is based on the former deliverable DR9.2 [7] and aims at updating it based on the experiences made so far during the work with the semantic object model. Hence, it basically is a copy of DR9.2 with punctual changes. In order to help the reader to identify the differences between DR9.2 and this deliverable, the most significant changes are listed here:

- References to future versions of PDKM or DR9.2, namely DR9.6a, and future tasks of WP R9 have been removed or updated. There are not scheduled any further revisions of this deliverable. Hence, this deliverable tries to cover the most important aspects of the semantic object model. Nonetheless, minor changes to the semantic model not anticipated until now might occur. They will be described in deliverable DR9.13 "PDKM Prototype Version 3", due M40.
- The present section 1.6 has been added.
- Section 4.2.1:
  - The attribute *Product_Type* was removed from the PHYSICAL_PRODUCT class.
  - The attribute *Parent* was moved from the PHYSICAL_PRODUCT class to the PART_OF class.
  - The attribute *Parent* was removed from the AS_DESIGNED_PRODUCT class. In the same turn, the self-association of the AS_DESIGNED_PRODUCT class was changed to be a directed association.
  - The association between the CONDITION class and the AS_DESIGNED-_PRODUCT class was replaced by an association between the CONDITION class and the PHYSICAL_PRODUCT_GROUP class.
  - The attribute *Group_Identifier_ID* of the CONDITION class was renamed to *Group_ID*.
  - Only in figures showing the CONDITION class, the attribute *Reference-_GROUP_ID* has been removed and the attribute *Flag_FD/Condition* was renamed to *Flag_FD/Property*. This was already reflected in the text.
  - Only in figures, the attributes *Property_Set* and *Condition_Set* has been removed from the AS_DESIGNED_PRODUCT class. This was already reflected in the text.
- Section 4.2.2:
  - The attribute *Supply_Environment* was added to the PRODUCT_BOL_SUPPLY class.
  - The attribute *EOL_Environment* was added to the PRODUCT_EOL class.
  - The association between the FD_SOURCE class and the ID_INFO class that was described in the text was replaced by an association between the FD_SOURCE

class and the PHYSICAL_PRODUCT class. However, a composition-association was added between the FD_SOURCE class and the ID_INFO class.
    - Only in figures, the attribute *Definition_Domain* has been removed from the VALID_FD_TYPE class. This was already reflected in the text.
- Section 5.1: The mapping table was replaced by a reference to a corresponding, up-to-date table in deliverable DR9.6b.
- Section 6.1: Most of the examples have been substantially reworked.

All changes to the semantic model have been reflected in all figures and in the corresponding text. Besides that, there have been minor editorial changes all over the document not changing the content.

2 DR9.6a's connection to other PROMISE deliverables

The work in TR9.2 and TR9.6 has been carried out considering the results of other work packages. Consequently, the outcome of this deliverable is closely related to other deliverables. This section makes a brief analysis of the main inputs and explains their impact on this deliverable.

2.1 WP R2: PROMISE generic models

With respect to deliverable DR2.1 "PROMISE generic models (version 1)" [3], which is mentioned in the DoW [2] as input to the task TR9.2, a major objective of TR9.2 and TR9.6 was to ensure that the generic product life cycle models and the generic product information flow models described in that deliverable are fully supported by the resulting object model. Thus, the system object model was designed to enable system functionality that facilitates the use cases and information flow models described for the generic models. To ensure this, several informal walkthrough sessions have been conducted WP R9 internally where the object model has been checked against the generic model.

The use cases and information flow models presented in DR2.1 cover the entire scope of the project including hardware aspects, PEID and middleware issues. Although these topics are not in the focus of the PDKM development phase, they were considered as well in order to ensure the compatibility of the object model with adjacent components and processes.

2.2 WP R7: Information and Knowledge Management Methodologies

In WP R7, and more precisely in deliverable DR7.3 "Selection of Tools and an existing PDM System to support PROMISE specific Knowledge and Information Management Processes" [4], the PLM system mySAP PLM has been recommended for use as a basis for the PROMISE PDKM system. In order not to constrain us, this recommendation has not been considered during the development of the system object model in the sense that the semantic model must easily fit into the mySAP PLM data model. However, it was considered in the sense that we were allowed to focus on selected areas that are important for the communication and alignment with the application scenario WPs A1 to A11 and other work packages such as R6 and R8. Details of areas, which are standard functions in mySAP PLM (e.g. the issue of access rights management), could be left out.

However, the mySAP PLM data model has a significant influence on the technical data model. The main objective with respect to compliance with the SAP system is to enable the development of a SAP-based PDKM system that realises the system object model. This aspect is discussed in more detail in section 5.1.

2.3 WP R9: Development of PROMISE information management system

The design and conceptual architecture of the PROMISE PDKM system was introduced in the first deliverable DR9.1 [2] of WP R9. This document cannot to be seen as an input to DR9.6a in a strict sense; more precisely the present deliverable has to be considered as an important supplement to what was described in DR9.1, since it provides many details of the required object model. Referencing the architecture represented in DR9.1, the object model described here is the core of the Data Management layer. The object model does not cover the other components like the System Management tower, the Metadata Management tower or the DSS, which is, again according to DR9.1, one of the components of the PDKM. These are treated in the development of the technical data model during the activities of tasks TR9.5, TR9.11, and TR9.13, or – regarding the DSS – in WP R8.

Nevertheless, the object model was checked in informal walkthrough sessions conducted WP R9 internally in order to guarantee that it does not contradict but – where applicable – supports the conceptual design presented in DR9.1.

Certainly, also the work on other tasks and deliverables of WP R9 (compare DoW [1]) fostered the refinement of the semantic object model.

2.4 WPs Ax: Application scenarios 1 to 11

The scope of the application scenarios was described in detail within the deliverables DA1.1 to DA11.1 "Scope, definition and description of the current state vs. critical issues"; more details have been provided in further deliverables of the application work packages [5]. These documents served as the main input for the work in this deliverable since they describe the needs of the application scenarios and have been used to check the applicability of the object model. In order to do this, the object model has been iteratively applied to the application scenarios during its development. This process has been carried out in an informal way and helped to identify missing points and improve the data model. Three examples of the application of the object model to an application scenario are documented in chapter 6 of the present document.

3 State of the art on product data modelling across product life cycle

In this section, an overview on the most important standards on product data is provided. The main features of the cited standards are outlined, as well as their capability of modelling product data throughout all, or part, of the product life cycle.

One of the basic requirements for the development of the product data model was that of achieving an easy understanding of the model by people or organizations of different countries, languages and cultures. This, as well as the intention to avoid misunderstanding of concepts, ideas or definitions, are supported by making use of or considering, whenever possible, shared standards. Moreover, the consideration of these standards for the purposes of the model outlined here permits more compatibility of the developed model with existing applications and standards. This enables easier interoperability and sharing of import and export data.

A literature review on standards for product design, manufacturing control and product support was first carried out. This review proved that none of the existing standards is capable of modelling the complete life cycle of a product, at least in the sense of the PROMISE project's vision. The semantic model of the PDKM outlined in the following pages has the purpose of achieving this capability. Nevertheless, though not "complete" in the sense stated above, each of these standards proved to be useful as a reference in the development phase of the semantic model, e.g. because it represents in a simple and compact manner some particular and fundamental concepts of a product's life. So, each of the standards presented in this section helped in building one or more of the bricks the object model is built out of. Some of them only provided concepts, some other gave ideas on the objects to be modelled in order to define a specific aspect

of a product's life, and some other finally inspired some names for the classes and attributes or even some compact and clear modelling constructs.

Among all, the following standards were identified and studied:

- STEP
- STEP NC
- PLCS
- MANDATE
- PLM XML
- ANSI/ISA-95

These standards have some properties and features in common, but are as well distinguishable by a lot of remarkable differences. First, they were designed by different organizations, with different scopes and for different targets. STEP, STEP NC, PLCS and MANDATE can be grouped together at first sight, because they are all ISO (International Organization for Standardization) standards; furthermore, PLCS is an application protocol of STEP.
STEP is an industry standard for product data representation and it is composed of several parts (application protocols) whose aim is to focus on a specific industrial context. There are application protocols for product design, for mechanical and electrical engineering, for sheet-metal manufacturing, for product assembly, for the automotive industry, etc.
PLM XML is an open standard developed mainly by EDS (Electronic Data Systems Corporation) and dealing with the product design phase.
ISA-95 is an ANSI (American National Standard Institution) standard, except for its first part, ANSI/ISA-95.00.01, which is also an ISO standard (ISO 62264-1). All together, ANSI/ISA-95 Parts I, II, and III describe the interfaces and activities between an enterprise's business systems and its manufacturing control systems: it mainly focuses thus on the area corresponding to the production phase of a product.

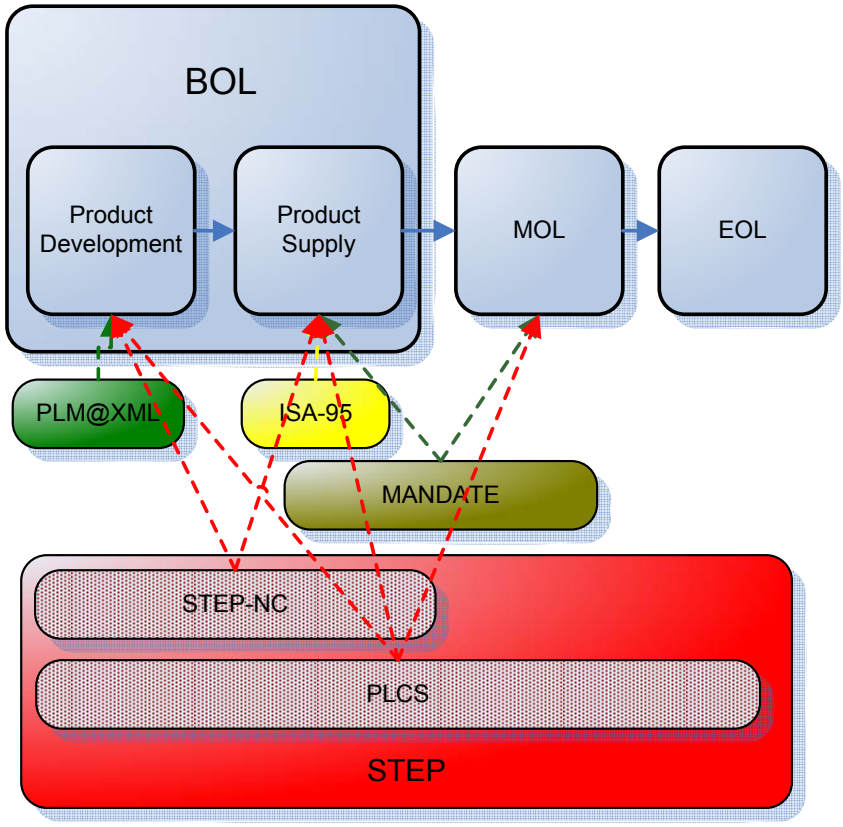

Figure 2: Standards throughout Life Cycle Phases

One more remark concerning another interesting standard should be stated here. In the deliverable DI1.2 “PROMISE Standardization Domains” [6], the standard dubbed PLM Services, published by the OMG (Object Management Group), is mentioned as one of the relevant standards for the PROMISE PDKM system. However, PLM Services is not included in the list given above because it focuses on the exchange of product life cycle data via Web-Services. The underlying data model itself is compliant to STEP AP214. Thus, PLM services might be relevant for other aspects of the PDKM development but it is not of special importance for the development of the object model since it is sufficiently covered by dealing with STEP.
In figure 2, the life cycle phases of interest and their association to the studied standards are shown. The figure also illustrates that none of the considered standards covers the whole product life cycle. The BOL, MOL and EOL phases are reported, as in the PROMISE vision.
The BOL phase is divided into two parts: the Product Development phase and the Product Supply phase. The first one is about the design and development of the product; here the physical product does not exist, it is only an idea represented on drawings and design information. Lots of standards focus here; the PLM XML and the STEP standards were chosen among the others because of their spread, importance and the fact that they are “open” standards.
The Product Supply phase is about the production of the physical product, the management of its supply chain and of its delivery to the final customer. Some of the most interesting standards that focus on this phase are ISA-95, MANDATE, STEP NC, and PLCS. However, some of them focus more on the production phase, such as STEP NC, some other on the delivery and none can be seen as “complete” in the sense stated above.
For the MOL phase, more or less specialized standards exist, such as again MANDATE and PLCS among those studied here. The same holds for the EOL phase, which is dealt with by PLCS and STEP.
In the rest of chapter 3, a short overview on the mentioned standards is provided, focusing on the characteristics that have been considered during the development of the data model.

3.1 STEP – ISO 10303

ISO 10303 – STEP (STandard for the Exchange of Product model data) is an international standard for the computer-interpretable representation and exchange of product definition data. It was developed with the aim to provide a mechanism capable of describing product data as defined in ISO 10303-1 ("*representation of facts, concepts or instructions about one or more products in a formal manner suitable for communication, interpretation, or processing by human beings or by automatic means*"), independently from any particular system. Its natural implementation is that of a computer system and CAD, CAM, CAE software for product design. The way it was designed for describing product data makes it suitable for neutral file exchange among different software solutions, also in a distributed engineering or manufacturing environment. It can also operate as a basis for implementing and sharing product databases and archiving.
One of the most important aspects of STEP [9] is its extensibility. This extensibility is the result of the decision to base STEP on a proper information modelling language, i.e. the EXPRESS language. The standard is organized as a series of parts, each one published separately. The structure is shown in figure 3.

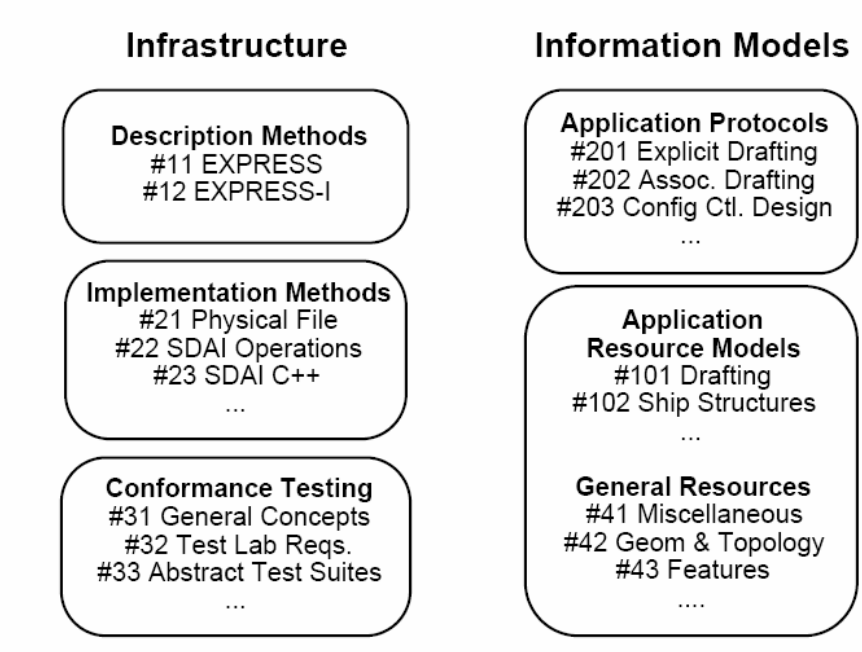

Figure 3: Parts defining STEP implementation

The Information Models, in particular the Application Protocols describe the data structures and constraints of a complete product model. Each application protocol combines one or more information models and places additional constraints on those models. For example, the application protocol for 2D drafting combines parts #42 and #46 and restricts the former so that it only describes two-dimensional data.

The Implementation Methods are protocols that are driven by the EXPRESS language. They are used to move real EXPRESS-defined application data between tools, and to make that data available to application developers.
The first implementation method is the STEP physical exchange file, often referred to as “Part #21 file format”. The Part #21 specification is the medium provided by STEP to move EXPRESS-defined data between databases and CAD systems.

STEP does not attempt to produce a single standard model that applies across disciplines. Rather, the STEP approach aims at producing standard product models for use within specific areas of application, called *Application Protocols* (APs), and strives to harmonize and coordinate these models across application areas to the greatest extent possible. The process of developing an AP is illustrated in figure 4.

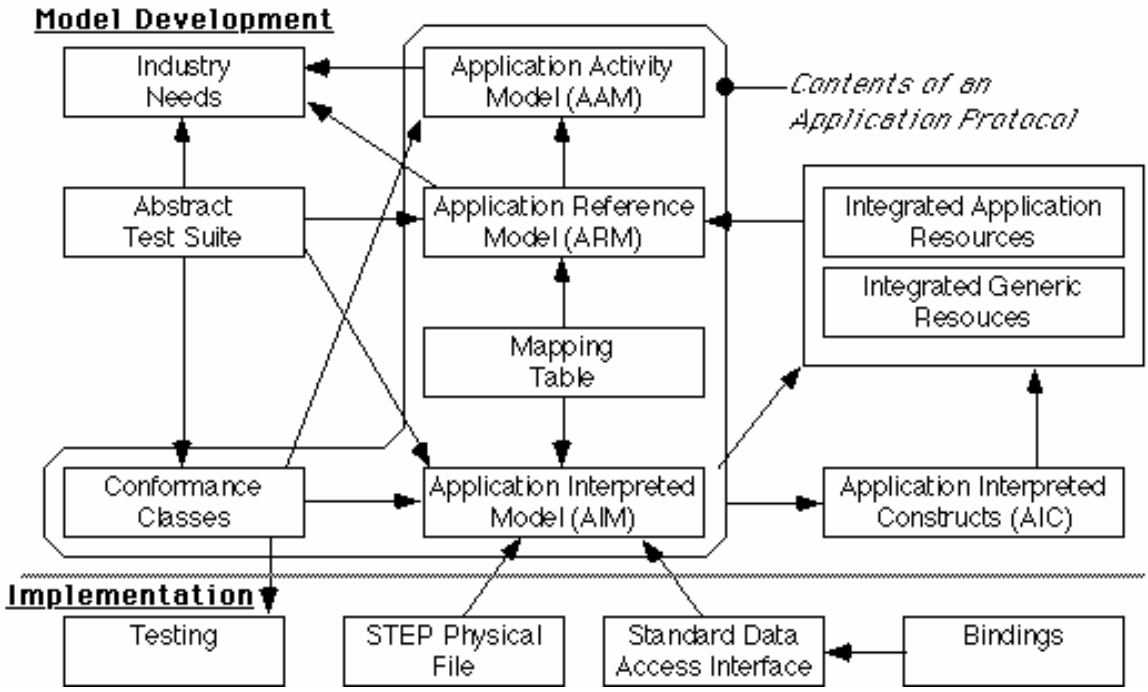

Figure 4: STEP - the Application Protocol Development Process and Implementation Components (arrows indicate reference to other sections of STEP)

To begin, AP's grow out of specific, perceived industry needs, and these needs must be well formulated and understood. Given an industry need, the role of the AP is documented in an *Application Activity Model* (AAM). The AAM identifies the business processes in which the AP is used, and shows the information flows among the processes. The AAM is presented typically using IDEF0 notation, which lists business activities and the information flows between them (see figure 5). The AAM is often the first interface with industry participants in the modelling process and is the primary tool for determining how the model is to be used.

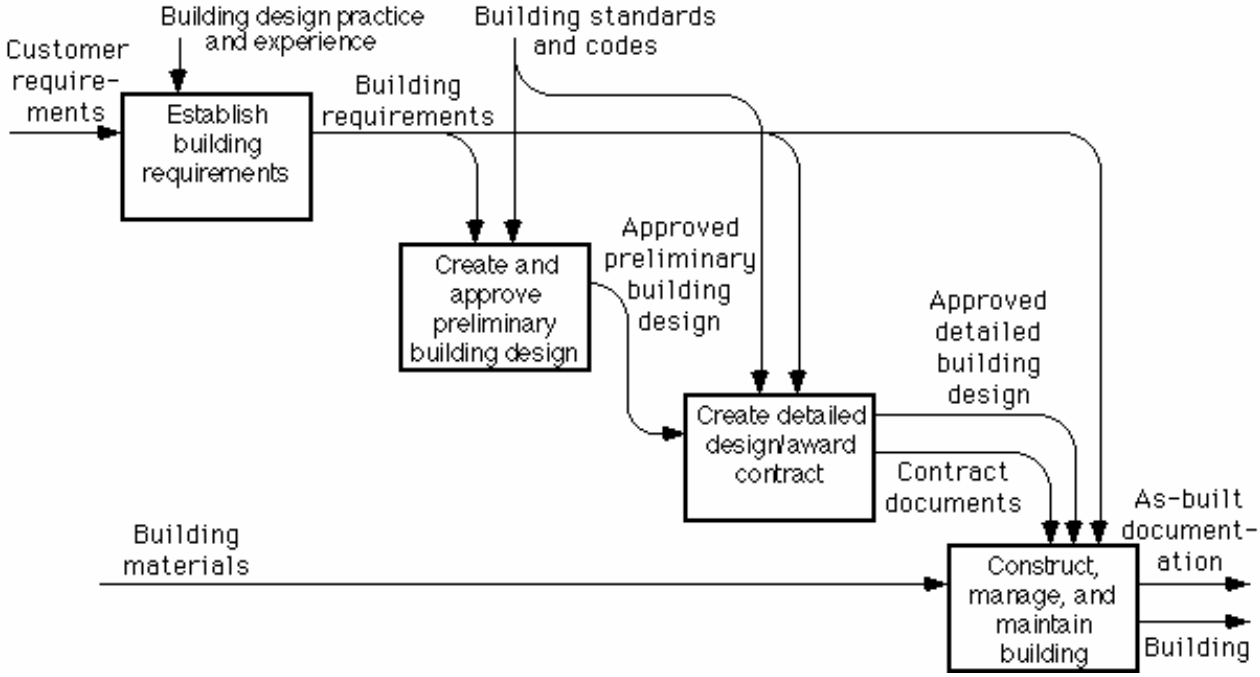

Figure 5: Example of an Application Activity Model (AAM) in IDEF0 Notation (from AP225 (ISO 1995))

Given this understanding of the model's role, the *Application Reference Model* (ARM) is developed. The ARM is a model that depicts the information that needs to be included in the AP using the terminology and concepts of the application domain. The development of the ARM encompasses the bulk of the model development effort, but it is still within the scope of the industry experts (rather than being the responsibility of modelling specialists alone).

Once the AAM and the ARM are developed, the task shifts to developing a model that fully defines all the necessary data representation structures in a way that is compatible with other parts of the STEP standard. This is the *interpretation* process and it results in an *Application Interpreted Model* (AIM), which draws upon the ARM and other reference models, both STEP-wide resources, *Integrated Generic Resources* (IGRs) and *Integrated Application Resources* (IARs). A *Mapping Table* correlates the entities from the ARM and AIM models.

In addition, *Conformance Classes* are defined (i.e. different levels, to which implementations of the model may adhere) and suites of test data are developed, through which implementations can be tested. Finally, where the interpretation process leads to the same basic concepts being represented in two or more AIMs, these model segments are defined in an *Application Interpreted Construct* (AIC) for use in future AIMs.

The actual AP document includes the following sections:

- A statement of the scope of the AP
- A list of normative references (e.g., other STEP parts that are explicitly referred to in the AP)
- A list of definitions and abbreviations used
- The information requirements for the AP: this section describes the ARM in terms of a listing of objects organized into related sets called *Units of Functionality* (UoFs), a complete listing of the *application objects* and their attributes, and a list of *application* assertions that specify the relationships between application objects, the cardinality of the relationships, and the rules required for the integrity and validity of the application objects
- The AIM described as a mapping table that maps ARM elements (objects, attributes, and assertions) to one or more AIM elements and as a complete listing in EXPRESS, a data modelling language that is itself part of the STEP standard.
- Appendices that include graphical representations of the AAM, ARM, and AIM

A fully developed AP, specifically the AIM model presented in EXPRESS language, is intended to be implemented to support information exchange. There are two primary implementation approaches directly supported by STEP. The first is file transfer, in which the data corresponding to some specific project is exported from one computer application, structured according to the AP, and formatted as defined by the STEP *Physical File Format* (STEP part 21). This file can then be imported and interpreted by another computer application. The second implementation method is direct access to the data as stored in a database or similar system. In this case, STEP Part 22 defines the *Standard Data Access Interface (SDAI)*, which provides a programming interface to the on-line data (bindings of the SDAI interface to languages such as C, C++, and IDL are being defined).

Some examples of STEP Application Protocols are:

- ISO 10303-201:1994 Explicit Draughting
- ISO 10303-202:1996 Associative Draughting
- ISO 10303-212:2001 Electro-technical design and installation
- ISO 10303-239:2005 Product Life Cycle Support (PLCS)

3.2 STEP NC (ISO 14649)

STEP NC [10] is a standard, which is explicitly focused on the design and production phases of the product life cycle. In particular, as the acronym NC (Numerical Control) indicates, it is focused on the integration between the activities performed at the backend during the design phase, i.e. the construction of CAD/CAM models, and those directly performed at the shop floor level, by the numerically controlled machine tools. Figure 6 shows the difference between the current situation and the one proposed by the STEP NC approach, in an intuitive way.

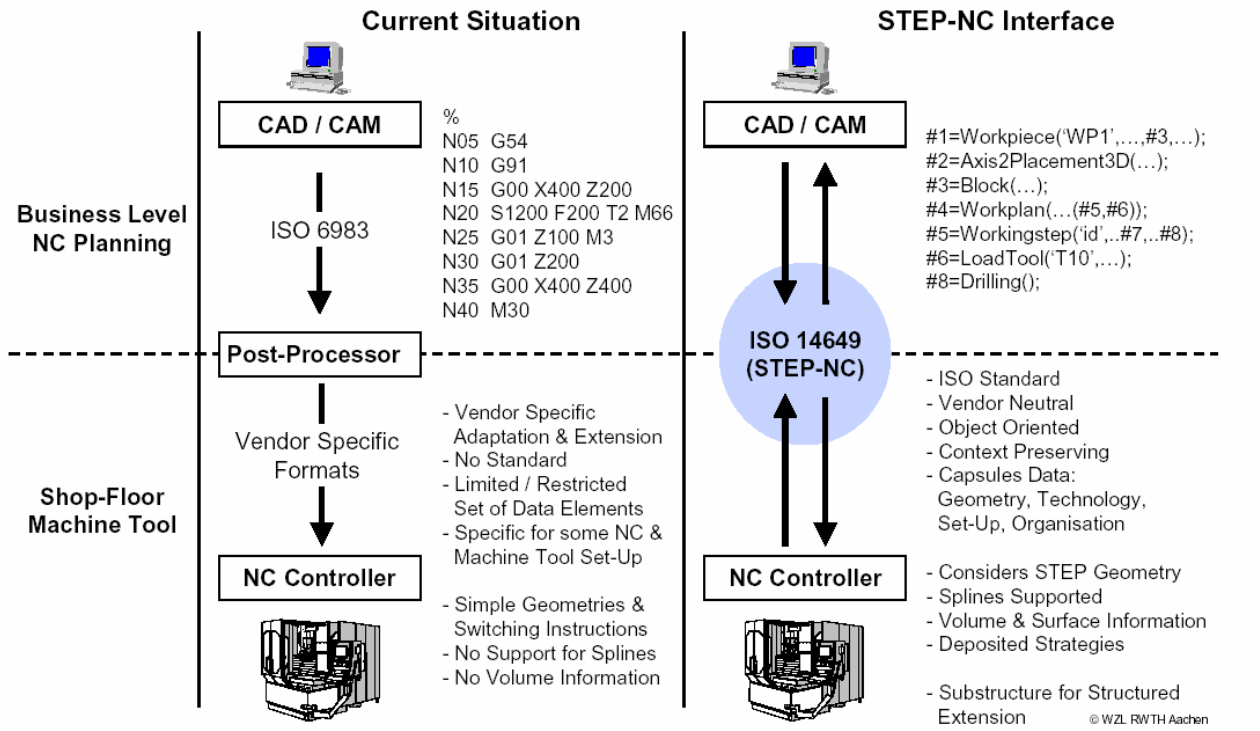

Figure 6: Comparison between the current state of the integration between design at the backend and production at the shop floor level and the STEP NC proposal

The basis for the necessary information technologies are in general as well as in the production phase data models and exchange formats to define how information is represented and to predict what can be processed and in which way. To close one of the last gaps in the complete and continuous information flow, ISO 14649 provides a standard (STEP NC) and an interface for the shop floor with its workflow. By referencing the ISO 10303 series, existing IT developments for the manufacturing sector such as for EDM and CAx can be used further on and be integrated into the modern numerically control production. Functionalities based on complex information are thus no longer limited to off-line working systems. Control vendors as well as software and system suppliers can realise more integrated, intelligent, and powerful applications, which are at the same time easy to handle due to high-level graphical user interfaces.

STEP NC (figure 7) is capable of representing both geometrical aspects concerning the product, i.e. its geometrical features such as holes, pockets, planes, etc., and the information describing how to machine the product (the so-called "workpiece"), i.e. which operations have to be performed at the shop floor, e.g. drilling and milling operations, the tool path to be followed during machining, the machining parameters, the strategy to be used, etc. A powerful hierarchical object model of all these pieces of information is provided by the standard by means of EXPRESS-G diagrams. This object model was used as a reference for the development of the semantic model described in this document, which in turn resulted to be compliant with it in the sense that a STEP NC representation of a product can be inserted into the framework provided by the semantic object model, if needed.

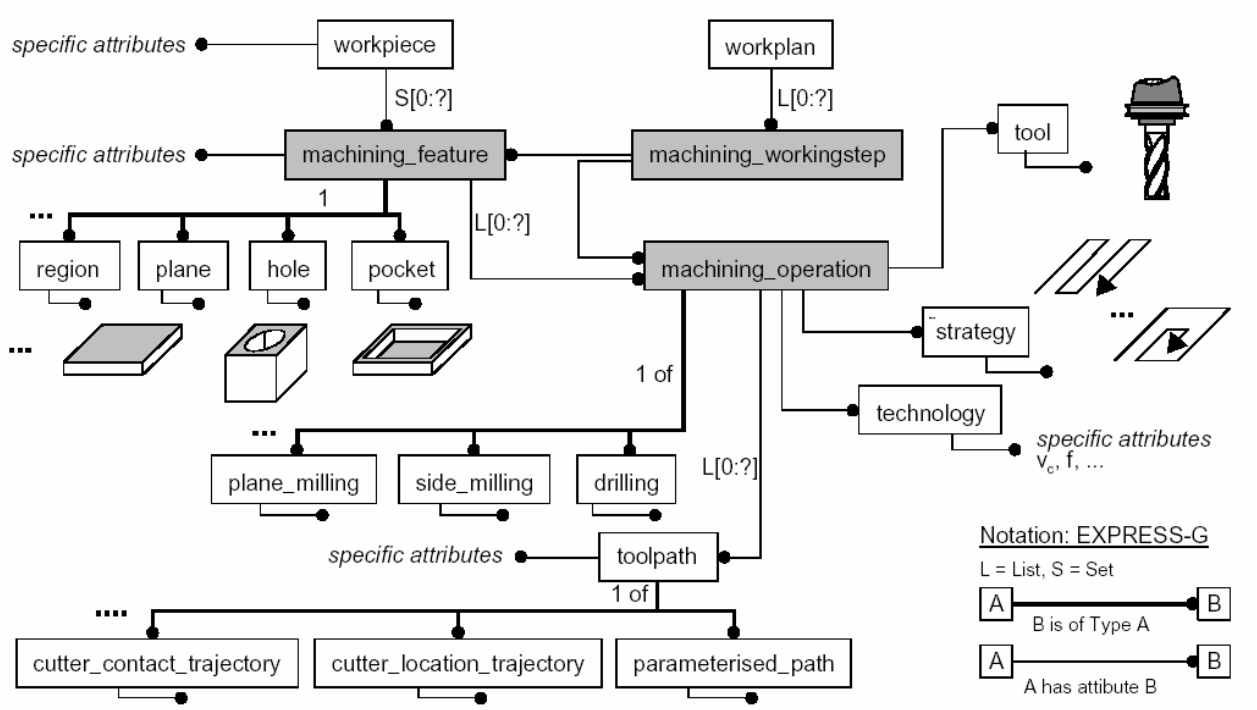

Figure 7: The STEP NC approach - encapsulation of product geometry and machining operations to be performed

3.3 PLCS – ISO 10303-239:2005

Product Life Cycle Support (PLCS) [11] is an Application Protocol of ISO 10303 (STEP). It was born as an initiative supported by both industry and national governments with the aim to accelerate the development of new standards for product support information. Now the work passed to OASIS (Organization for the Advancement of Structured Information Standards), which is a non-profit, international consortium that drives the development, convergence, and adoption of e-business standards.

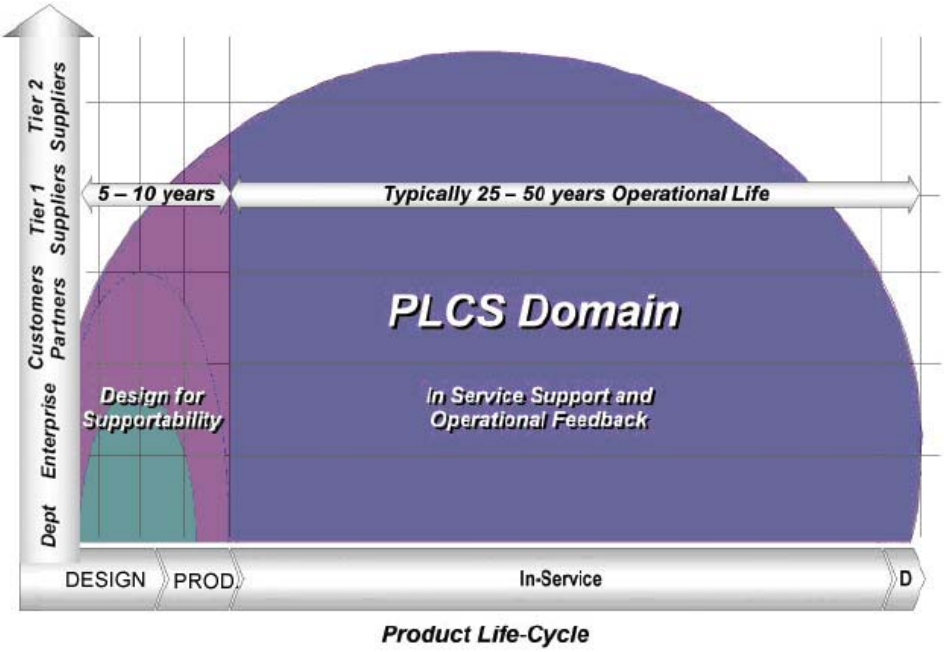

Figure 8: PLCS Domain

PLCS, as represented in figure 8, should be able to describe products in its whole life cycle (Design, Production, Use, and Decommissioning), with particular emphasis on needed support and work required to sustain and maintain the products in operational conditions.
As it was built around STEP, it is easy to integrate PLCS data and applications within complex and heterogeneous software systems, reaching a high degree of interoperability. PLCS, indeed, shares the same common interface of other STEP-based software for product design and development, for maintenance management, for manufacturing scheduling and so on (as shown in figure 9).

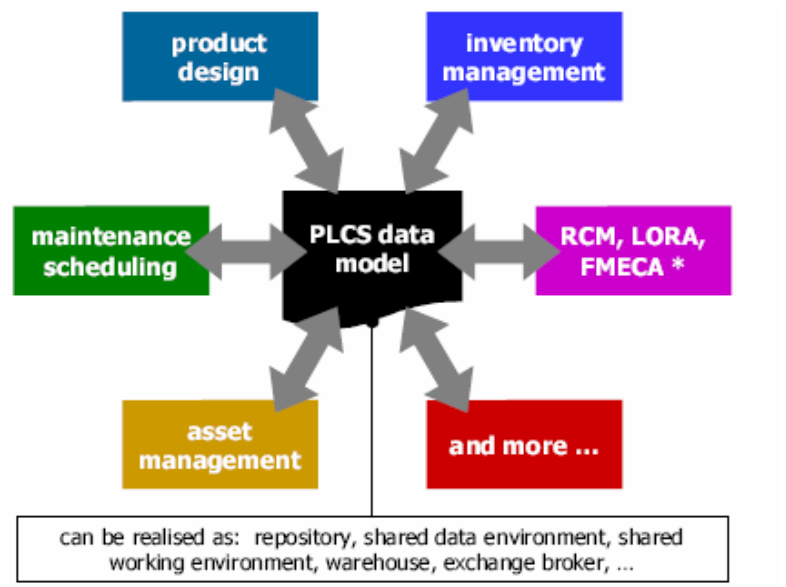

Figure 9: Inter-operability of life cycle software applications through PLCS

For specifying or recording the required support activities throughout product life cycle, a set of Assured Product and Support Information (ASPI) is defined. Life cycle data for a specific product is composed by both ASPI and their related information, such as feedback on product history, activities and resources used, etc. All these pieces of information can be managed in an integrated form, represented by the so-called PLCS Vision (figure 10).

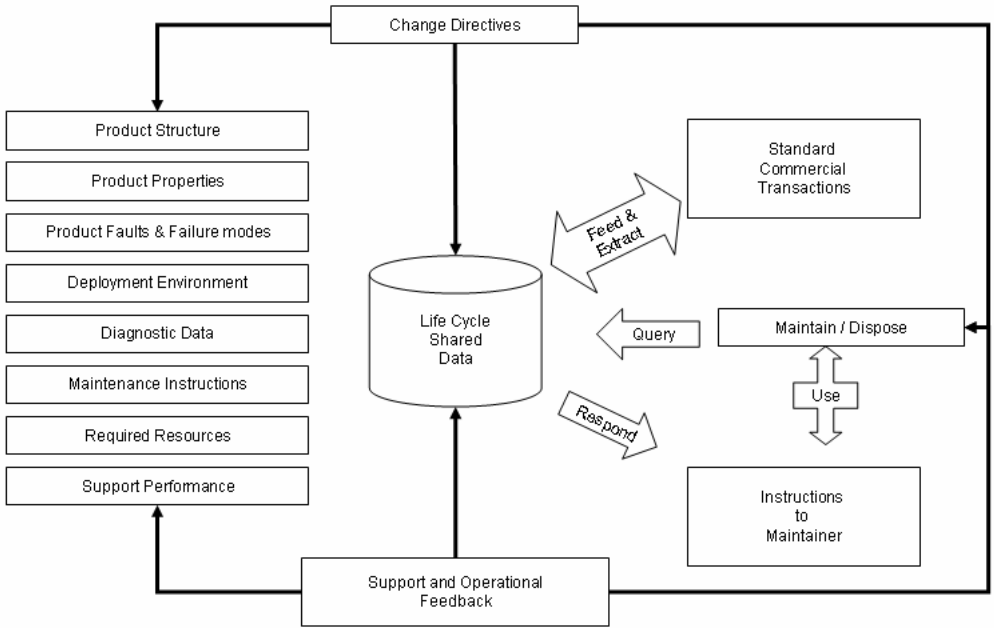

Figure 10: The PLCS Vision

PLCS is based on three top-level concepts (figure 11): Product, Activity and Resource. Each of these concepts is in relation with Properties, States or Locations, and Conditions can be applied to these relationships. The idea of Products, Activities, and Resources has been used also in the developed model (compare section 4.2).

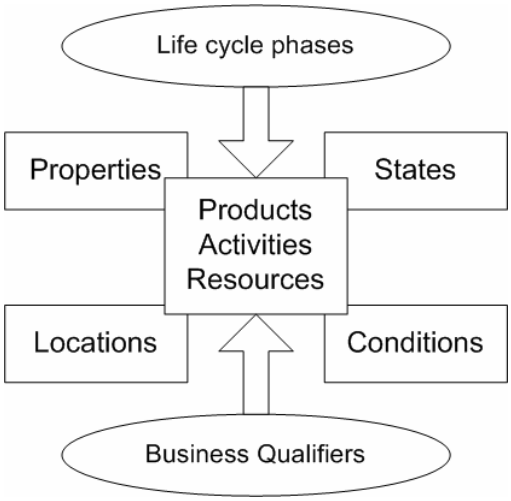

Figure 11: PLCS concepts

Products are described by means of the Application Module Product Structure (AM 1134). This references other AMs to define product sub-components, their relationships, their assembly structure and many types of breakdown, by which a product can be affected.
Activities are defined within AM 1047. Examples of activities are works done by people or organizations, usage of products, planned maintenance, etc. It is noticeable how PLCS distinguishes between future planned activities and activities that have already taken place or recently started (defined in AM 1259, Activity as Realized).
Resources are required to perform a task, can be quantified, specified, and are distinguished between required resources (AM 1267) and resource item (AM 1266). These resources are used by activities involving products and can represent, for example, people providing product support, instrumentation, software, tools for repairing products, and so on.
Products, activities and resources are characterized in terms of location, properties and state attributes.
Starting from this conceptual model, PLCS defines an abstract module hierarchy (figure 12), which implements the previously described concepts. For example, the main module is the Product Life Cycle Support module, acting as the root for the other related modules, such as Product Status Recording, Activity Recording, etc.
In the PLCS Application Protocol, the Application Modules are closely bound in a hierarchical structure, starting with a root AM called Product Life Cycle Support (ISO 10303-439). This main module is the root, for example, of Product Status Recording and Activity Recording AMs (respectively AM 1304 and AM 1287), as displayed in figure 12.

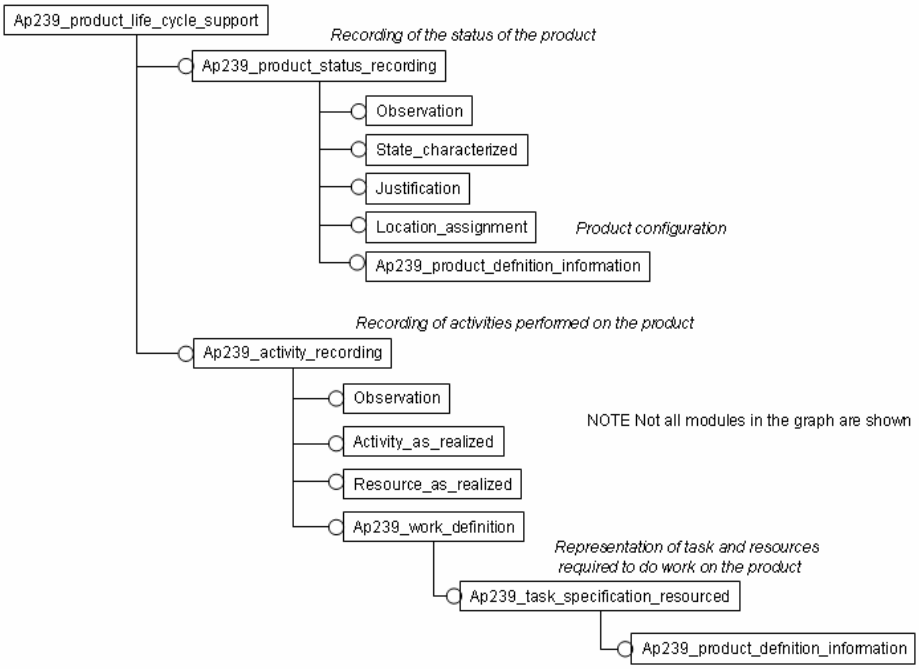

Figure 12: Abstraction of module hierarchy

The PLCS standard uses the same language used in the STEP standard, i.e. EXPRESS, and more precisely its graphical representation, namely EXPRESS-G, which allows a synthetic representation of ARM and MIM within a module. Each AM is defined both using EXPRESS and EXPRESS-G. The result is a collection of AM schemas detailed at two different levels, both for ARM and MIM. The first level is a "Schema Level" focusing on the relationships with other required AMs. The second level is an "Entity Level" and focuses on entities and entity types as building blocks for the AM definition. Figures 13 and 14 show an EXPRESS-G schema of ARM and MIM for a specific module at Schema Level, whereas figure 15 is an Entity Level MIM schema belonging to the same AM.

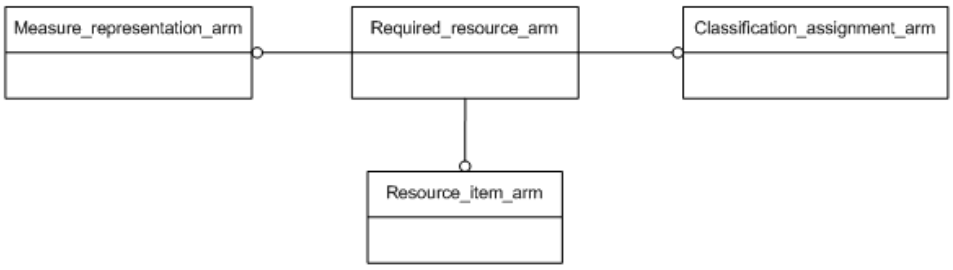

Figure 13: EXPRESS-G schema of the ARM for the "Required resource" Application Module (1267)

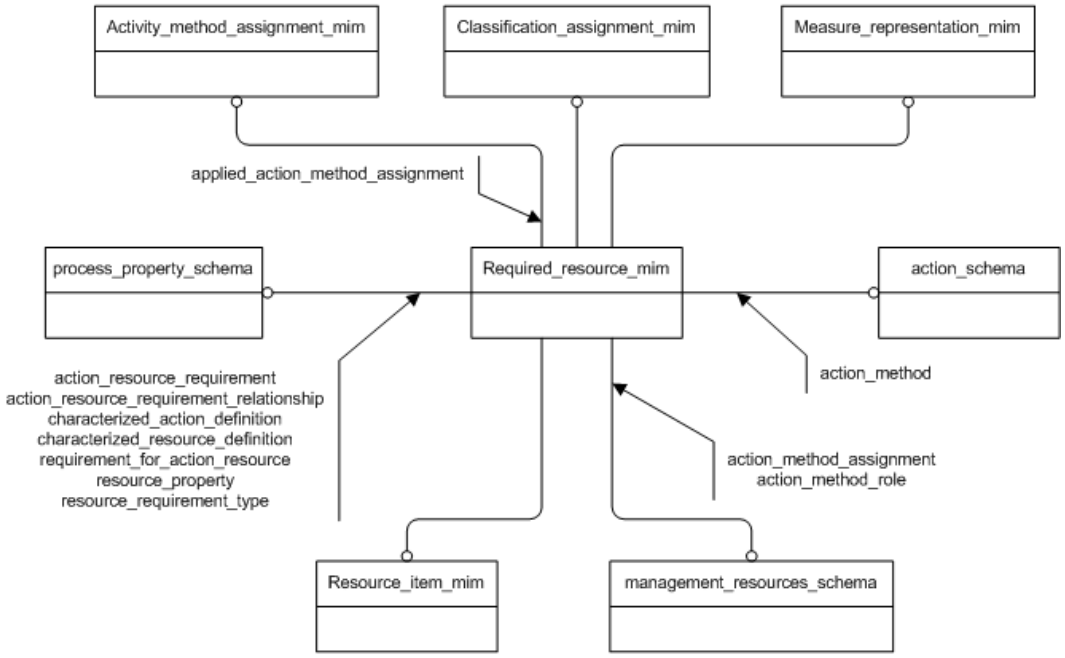

Figure 14: EXPRESS-G schema of the MIM for the "Required resource" Application Module (1267)

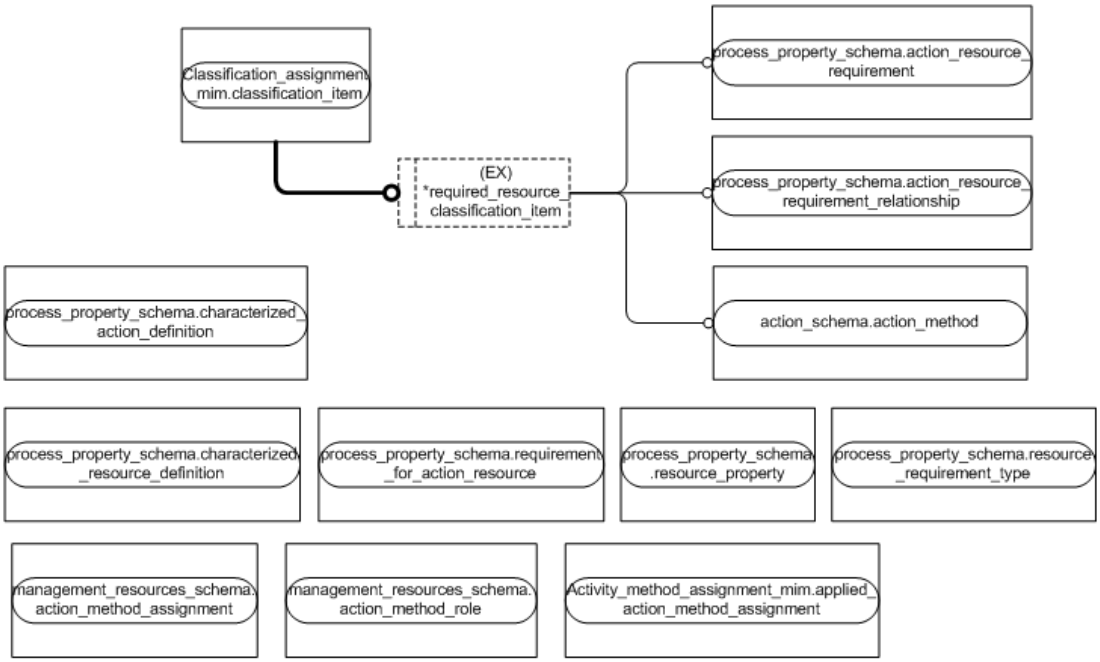

Figure 15: EXPRESS-G Entity Level schema of the MIM for the "Required resource" Application Module (1267)

3.4 MANDATE – ISO 15531

MANDATE (ISO 15531) [13] is a part of the set of standards provided by TC184/SC4; it has been under development for quite a long time, but there are only a few easily accessible results. The main intention of the developers of this standard is to provide capabilities for the description and management of product data throughout the portion of the product's life concerning its production.

The areas of interest for TC184/SC4 are:

- ISO 15531 (MANDATE) Industrial automation systems and integration - Industrial manufacturing management data
- ISO 13584 Industrial automation systems and integration - Parts Library
- ISO 14959 Industrial automation systems and integration - Parametrics
- ISO 10303 (STEP) Industrial automation systems and integration - Product data representation and exchange

MANDATE (MANufacturing DATa Exchange) tries to define a common representation for all of the manufacturing pieces of information.
The scope of the standard is the representation of production information and resources information including capacity, monitoring, maintenance (from a global point of view in relation with their impact on the flow control), and control, as well as the exchange and sharing of production information and resources information including storing, transferring, accessing, and archiving.
STEP takes a product-oriented view of manufacturing while MANDATE is concerned with the processes of the organization, which are carried out to produce the products.

MANDATE is divided in three series of parts based on a common overview and fundamentals:

- ISO 15531-1 : Overview and fundamental principles
- ISO 15531-2x : Production Data: External Exchange
- ISO 15531-3x : Manufacturing Resources Usage Management Data
- ISO 15531-4x : Manufacturing Flow Management Data

Some more information on the three series of parts:

- **Parts 15531-2 series (Production Data: External Exchange):** These parts refer to the representation of production information, which has to be exchanged with the external environment. The aim is to model the main production information exchanged between industrial companies, using the EDI protocols. The domain includes all information and functions necessary to support quality and order management issues, such as planning, executing, controlling, and monitoring of product quality, orders, and shipments.
- **Parts 15531-3 series (Manufacturing Resources Usage Management Data):** These Parts refer to the resource usage management, such as resource configuration and capabilities, operation management of manufacturing devices, installation and facilities. They also include quality features, maintenance-features (regarding the availability), and safety-features.
- **Parts 15531-4 series (Manufacturing Flow Management Data):** These parts refer to the material flow control and are intended to standardize data and elements, which support the control and monitoring of the flow of material in manufacturing or industrial processes. This includes all the elements describing the material flow, including inventory. They need strong relationships with resources usage management data.

ISO 15531 addresses operations dealing with product manufacturing, and uses components that will be consistent with STEP. In particular, MANDATE will be compliant with the STEP architecture and the different parts of MANDATE will use as far as possible the EXPRESS language. As cited above, this standard is – though of interest – still under development and was used therefore for the purpose of the development of the semantic object model only as a general guide.

3.5 PLM XML

PLM XML [12] is an open standard proposed by EDS to facilitate high-content product life cycle data sharing. This standard especially focuses on the design phase of the product's BOL.
The functional objectives of PLM XML are:

- to orchestrate product structure, product information and geometric representation data in an open, lightweight and extensible form,
- to share relevant information through the product life cycle by offering a standardized protocol for data interoperability,
- to integrate collaborative product life cycle processes by providing a coherent flow of data within a heterogeneous application environment.

Figure 16: Grouping of product information in PLM XML

All the information is grouped in one unique file that can also contain the open solid model format Parasolid XT and the visualization format JT (see figure 16).

PLM XML schemas define a hierarchy of product information and relationships; the supported data includes the following:

- **Evaluated product structure**: PLM XML schemas define a mechanism for exchanging an evaluated product structure, suitable for product development, BoM (Bill of Material), and assembly visualization. The product structure can be represented via an instance graph structure as well as multiple occurrence-based product views. The product views can reference the instance graph to enable display configurations based on an internal structure. For part representation, PLM XML includes the concept of a part and its metadata, but does not include schema definitions for the explicit geometric component representations. Instead, parts within PLM XML can reference multiple external representations such that the receiving application can decide which representation is suitable for loading.
- **Visualization properties:** PLM XML provides visualization property exchange in the form of view controls (view directions, view ports, view characteristics) and display controls (lights, backgrounds, etc.).
- **Reference geometry:** PLM XML can exchange reference or wire frame geometry via its geometry schemas. The schemas are based on Parasolid geometry definitions and include NURBs and analytic forms.
- **Visualization Features:** PLM XML supports the representation of product features, including feature hierarchy, suitable for visualization and feature browser applications.
- **Associativity:** PLM XML defines elements to enable associativity back to the sending application. This associativity is an optional element that may be added to virtually any PLM XML element that describes the name of the sending application and a persistent label for the object itself (an optional version may also be specified). The label is specific to the sending application but may be used by the receiving application to label the equivalent object.
- **Component reference:** PLM XML uses a standard URL/URI mechanism for specifying the location of data elements within a distributed data environment.
- **Entity reference:** PLM XML has extended the URI mechanism via a "PLM XML pointer" syntax to enable entities within data files or components to be referenced.

This initial set of capabilities is being expanded to encompass a broader range of data that passes between applications during collaborative working in the product life cycle, including additional product structure information, product manufacturing information, and product data management (PDM) metadata.

PLM XML can be useful for enterprises for the following reasons:

- **Interoperability**: For example with respect to part file or assembly data exchange, PLM XML can provide a vehicle for sharing product information, allowing common access to structure, attribute data, and referenced representations. Figure 17 shows PLM XML being used both as a directly supported format in sending and receiving applications, and as a tool for abstracting the native part file format to extract the essential product data.

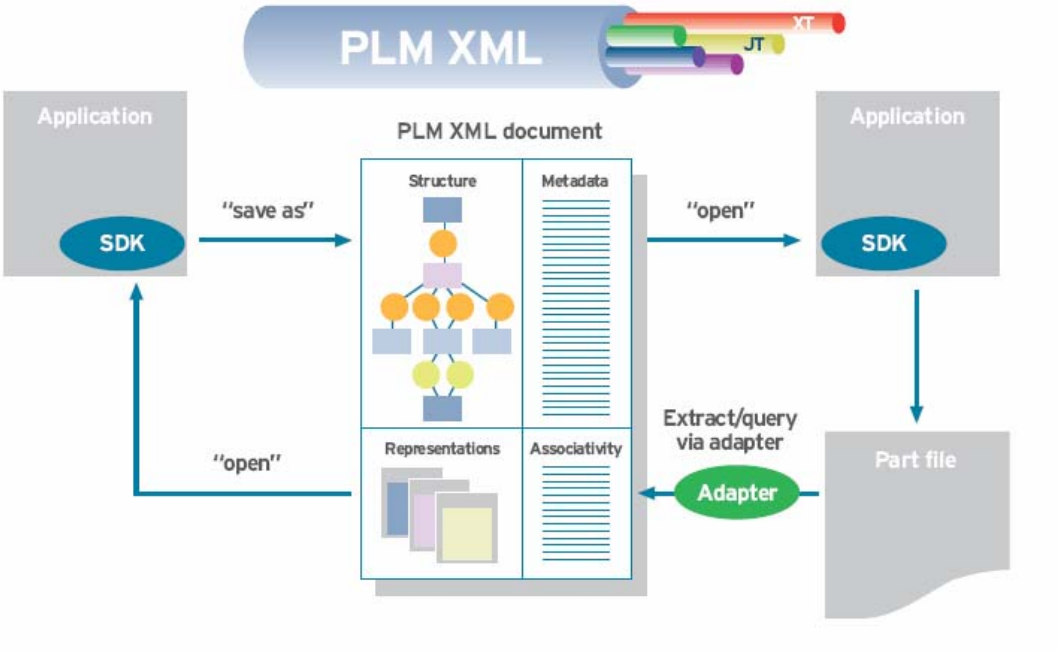

Figure 17: PLM XML Interoperability

- **Collaboration**: to deliver a packet of information to an extended team of consumers for collaborative review, while the receiving applications can identify and consume the data most appropriate to their needs and application capabilities. PLM XML will allow receiving applications to consume selectively the contents of the PLM XML file, enabling the same payload to be used for collaboration with a variety of participants (figure 18).

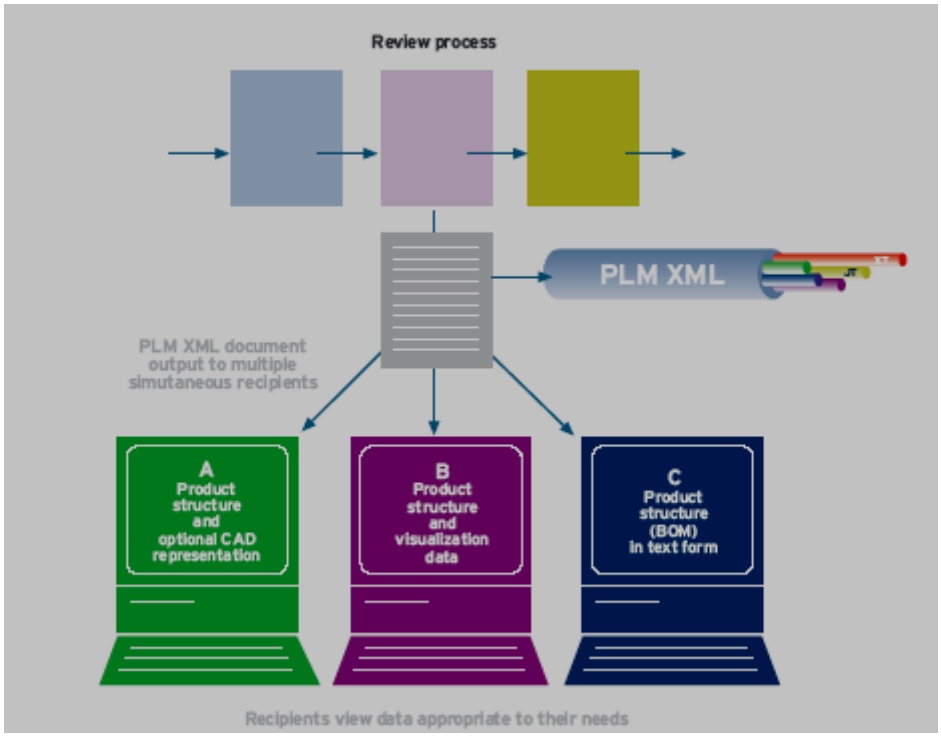

Figure 18: Group level shared collaboration

- **Integration**: PLM XML can embody the data payload in more formalized enterprise application integration scenarios, where there is a need to compose data payloads from potentially multiple data sources and communicate to multiple recipients within a given transaction context. Using PLM XML means that the payload can be high level (for example, product structure as opposed to single objects) and can be received and processed in a variety of application environments to maintain consistency across an extended enterprise. In this way, the PLM XML payload protocol can be used in formalized transactions between tightly integrated systems and at the same time be distributed to more loosely coupled systems, which need to be connected (figure 19).

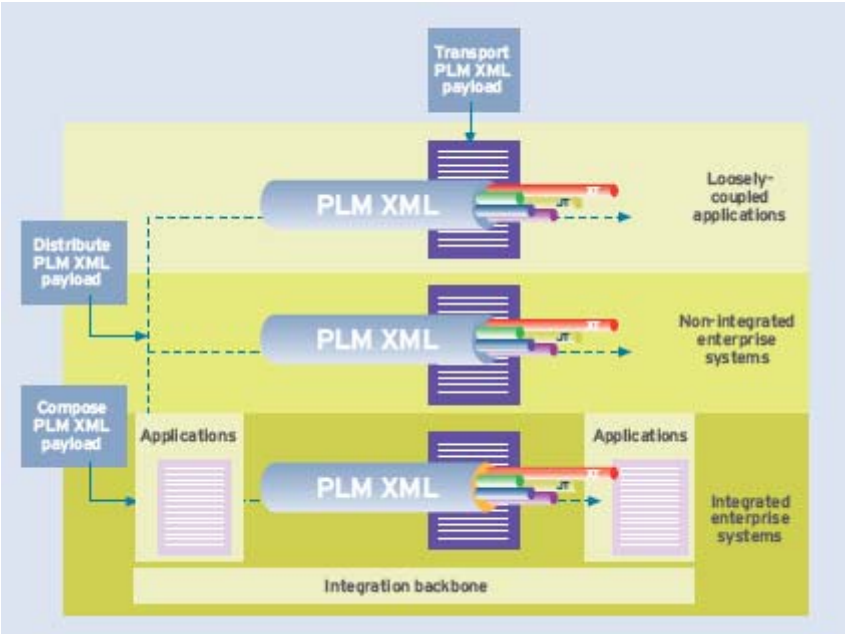

Figure 19: Integration with business applications

3.6 ANSI/ISA-95 (ISO 62264)

ANSI/ISA-95 [14], which has been accepted by ISO community and published as ISO 62264, is a standard composed by different parts aiming at the definition of the interfaces between enterprise activities and control activities.

Part1 (Enterprise/Control System Integration) describes the relevant functions within an enterprise and within the control domain of an enterprise, stating, which objects are normally exchanged between these domains. In detail, this first part concerns with the interface between two levels of the functional hierarchical model proposed in figure 20: in particular the interfaces between "Level 4" (Business Planning and Logistics) and "Level 3" (Manufacturing Operations and Control).

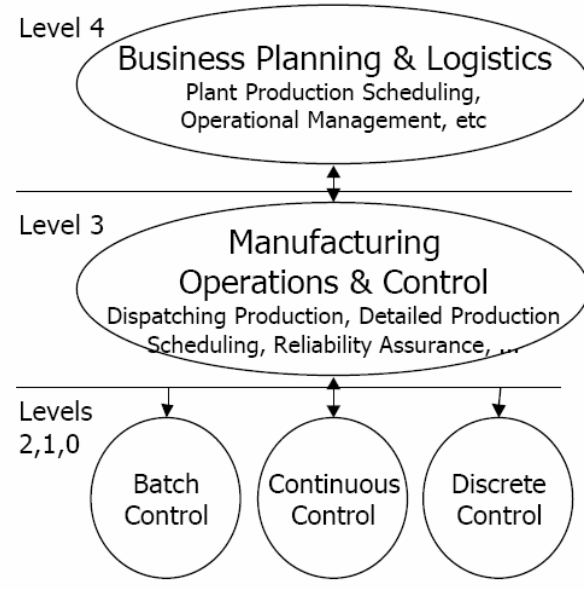

Figure 20: Functional hierarchy

Taking into account this purpose, it is necessary to define, which are basic functions involved at this interface, which is their organizational structure, and which is the information flowing among these functions. ANSI/ISA-95 part 1 defines, in other words, a functional model and a related information model. Figure 21 shows the functional model, highlighting the boundary of the enterprise/control interface, which in turn corresponds to the interface between the previously shown levels 3 and 4.

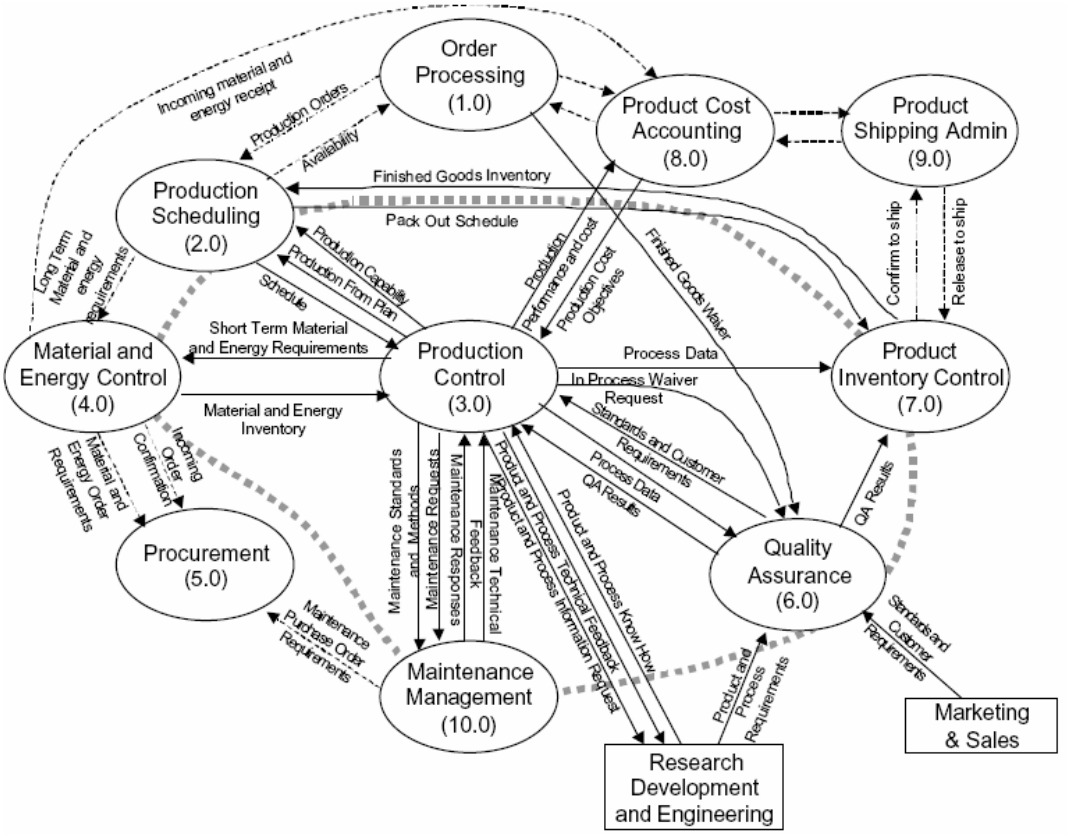

Figure 21: Functional enterprise/control model: the dotted line shows the boundary of the interface

On the other side, ANSI/ISA-95 defines general categories of information shared by the functional model and their specific structure. Figure 22 represents areas of generic information exchange.

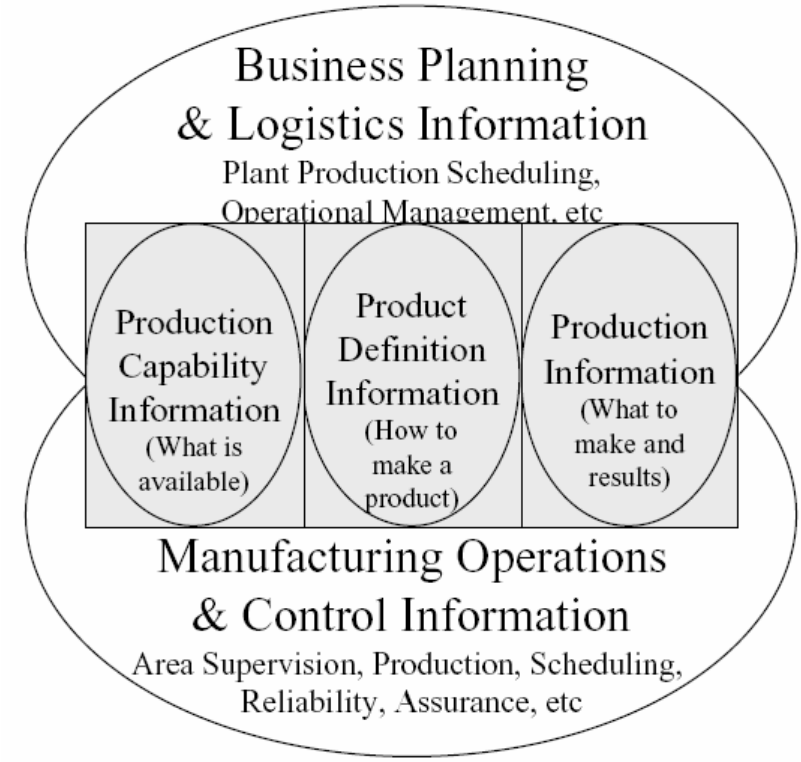

Figure 22: Areas of information exchange

Each of these areas of interest is then treated in further details. For instance, Production Capability (figure 23) is a collection of information related to the production capability of personnel, materials, and equipment for a specific manufacturing area or site. This information is completed with maintenance data for each equipment within the same production area and by information regarding the availability of capability for single process operations or grouping of operations, which define a process segment. All this information and data define, in other words, for each production unit, site or area, the up-to-date capability for each product-processing phase. This area takes care of enterprise-side information.

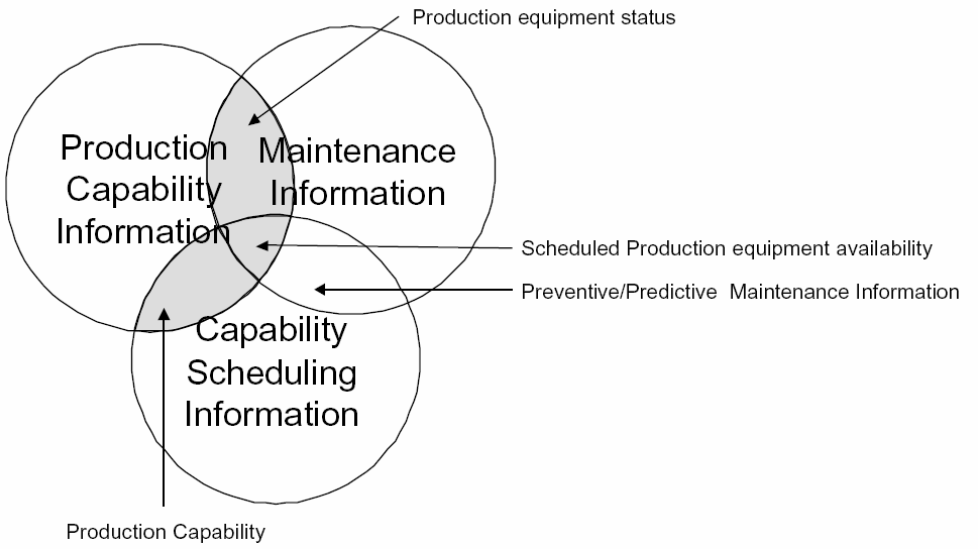

Figure 23: Schema for Production Capability

Product Definition (figure 24) describes for each product type its product production rules, bill of materials, and resources. This information area contains all data needed for technically defining a product manufacturing operation, specifying, which product subcomponents are required, which resources as machines, personnel, tools, and so on shall be used and how. The point of view of this area is focused on the product.

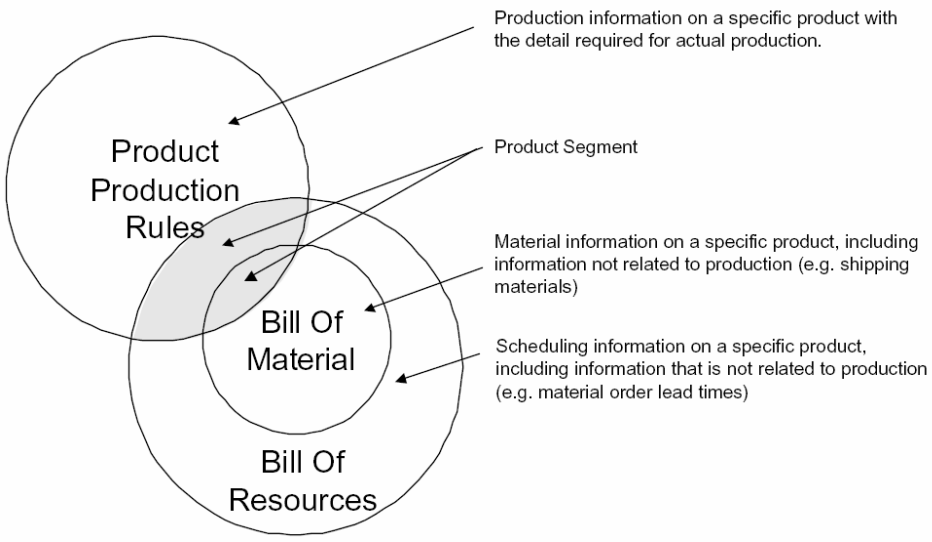

Figure 24: Schema for Product Definition

The last main area is that of Production Information (figure 25). Here, information on product production history (log), on production inventory of consumed and produced materials, and information on production scheduling are collected. This area is interested in describing how

products are produced and displaying production performances. This way, it put together a product-based view with an enterprise-side based view.

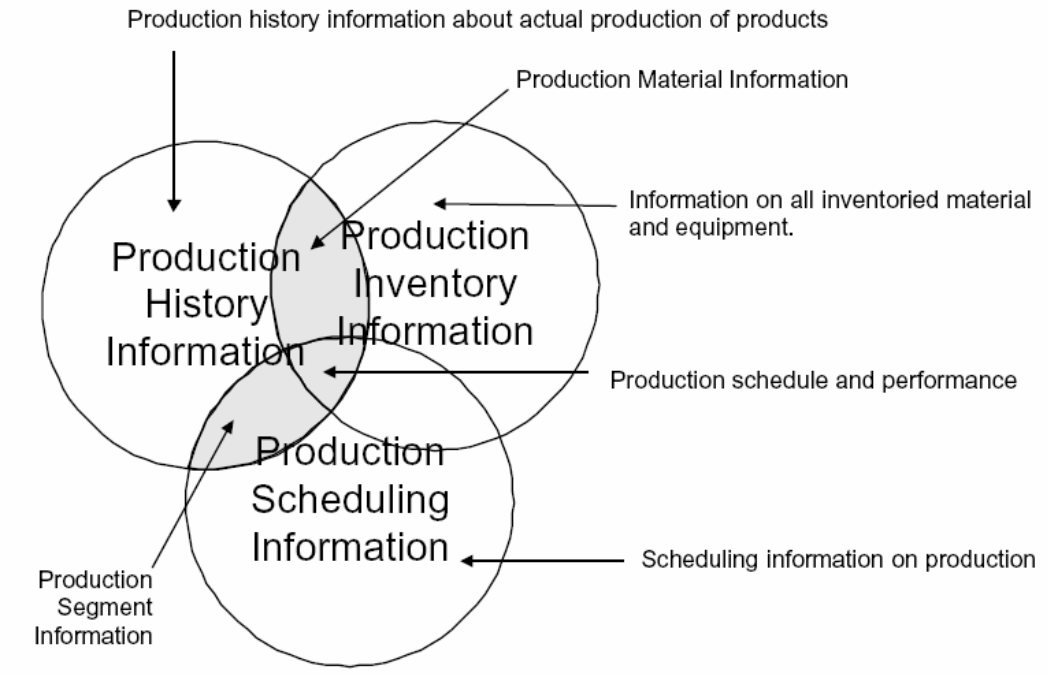

Figure 25: Schema for Production Information

Within all these areas, there are types of information, which are shared among several areas as well as specific types of information used for a single area only. ANSI/ISA-95 makes use of UML 1.0 (Unified Modelling Language) representation for displaying each class of information and its relations with other classes. Detailed object models are developed for these pieces of information (see for instance figure 26) and also a detailed description of the most important attributes belonging to the classes of the object models are provided (part 2 of the standard).

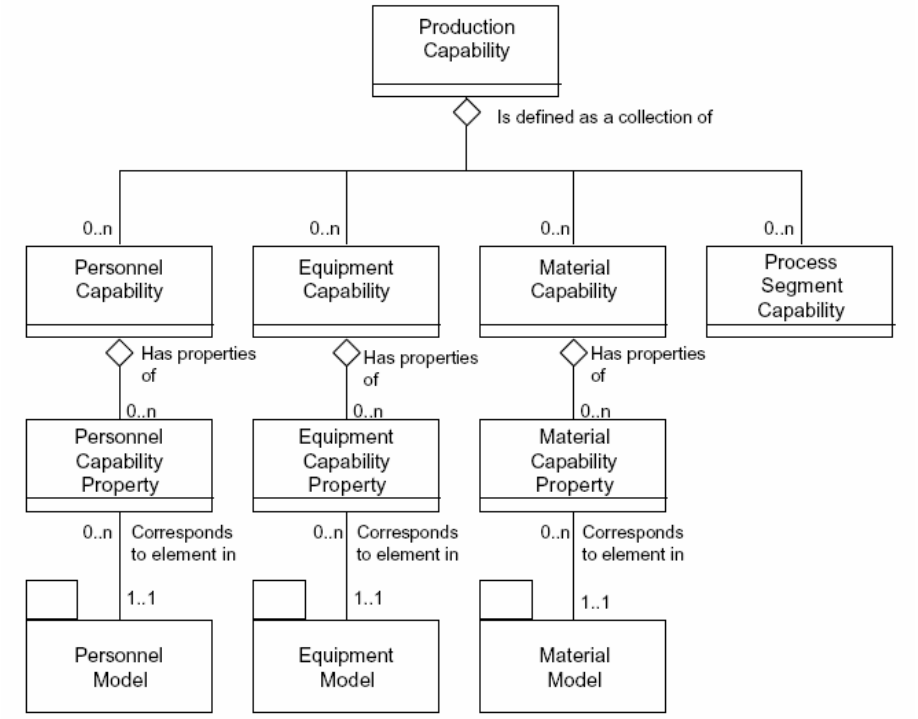

Figure 26: Production Capability model

4 General System Object Model (Semantic model of the PDKM)

In this section, the General System Object Model of the PDKM system is outlined. Before entering a detailed description of the model with its classes, attributes, and associations, a brief introduction on how the model was developed is provided in order to state the used point of view, the pieces of information modelled, and the relations with other key PROMISE issues.
Then, an overview on the main classes and relations among them is given as a first guidance. Finally, each class of the model is presented with its most important attributes and a more detailed description of the associations with other classes of the model is provided. Attributes of minor importance or with an obvious intention that does need further explanation were sometimes omitted. Where considered as helpful for the understanding of the concept, examples are given.

4.1 Important notes on the modelling criteria adopted

The semantic model of the PDKM system presented here is capable of providing the PROMISE consortium with a conceptual view of the system, representing the main concepts belonging to the domain of interest. These concepts are in a natural correspondence with the rest of the PDKM system and with the way, all of them are implemented, but such links are not developed further here; instead, they are studied in other activities of WP R9. The model, however, was designed with the aim of providing the system with a basis for representing product data throughout the whole product life cycle.
The focus is more on data modelling than on complete software systems modelling. The completeness of the model, in the sense of a complete specification of all required classes and of all of their attributes, is both not intended and not required to show in a clear way the domain of interest of the PROMISE vision and approach. Thus, there might be classes of minor importance as well as attributes of listed classes not be mentioned at all.
In the Semantic Object Model presented in the next pages, particular importance is given to the representation of the information whose loops across the product's life have to be closed by the PROMISE project in the different application scenarios. This requires a proper modelling activity for the information one wants to be attached to each product instance or even product components and subassembly instances where requested by means of smart product embedded devices.
A certain degree of abstraction was required in order to make the present model capable of representing the different needs coming from the whole set of PROMISE application scenarios. Considerable importance was given to the pieces of information describing each of the product life cycle phases, such that the PDKM system is able to manage data and knowledge in the way required by each application scenario. Different scenarios focus on different life cycle phases, in different ways and for different purposes.
The pieces of information concerning product types rather than those concerning product instances are also included, even if they do not represent the focus here. Thus, only a small portion of the model refers to pieces of information on products "as-designed", since most PDM Systems (mySAP PLM, used for the PDKM, included) widely treat this data much better than the way it could be done here. Nevertheless, this kind of data has not to be excluded from the model, since most of the PROMISE application scenarios clearly require them.
Another key aspect of the Semantic Object Model is that it describes field data, relating them to the different life cycle phases where they are first collected and then analyzed. To develop this portion of the model, some abstraction was required again in order to be compliant with the different needs of the different PROMISE scenarios.

These are the most important, high-level features of the PDKM Semantic Object Model. Concerning the modelling methodologies, languages, and tools used in the development phase, the statements in the rest of this section hold.

The model is explicitly object-oriented, by this following today's most used paradigm for software development. UML (see [21]), an important standard language for graphical modelling purposes was chosen and used, by this also following the general tendencies in the PROMISE project (refer e.g. to DR2.1 [3]). The model is compliant with UML 2.0 [18].
An existing approach from the literature on product data modelling throughout the product life cycle was used as a first reference (refer to [15], [16], and [17] for example). The model presented there is also object oriented and represented in UML language and provides an interesting representation of the basic concepts one should define in order to enable the complete product traceability, both "forward" and "backward" (see again [15]), along a product's life.

Since the Semantic Object Model only contains the static view on the PDKM system, in the terms defined above and further treated below, the UML Class Diagram was chosen as the main UML tool to be used in the modelling activity. Only a subset of the whole UML syntax for class diagrams is used here, since there are some constructs, which enable a clear description of the concepts behind a graphical model, without the need of using more complex constructs with low value added to the semantic description of the domain of interest. The focus is on the classes building up the model as well as on the attributes describing their main features. Associations among classes are used as well as the generalization construct. Sometimes associations are described as compositions to show where complex objects (e.g. more complex products) are built out of components and subassemblies, for which information must be stored and managed. Some other constructs, such as the aggregation, have intentionally not been used, since the object community has not reached a common agreement on its meaning and utility yet, despite of the fact that aggregation is sometimes widely used (but with different meanings, as reported in [18]). Therefore, instead of aggregations, normal associations have been used.
Cardinalities of associations and attributes are shown, while only some generic reference is reported on data types corresponding to the defined attributes. For instance, if an attribute refers to the date and time that a certain event on the product has occurred, e.g. the date and time when the clutch of a car broke, it is stated that the attribute is of type "Date" even if "timestamp" would be more appropriate. The "Boolean" data type is also indicated, as well as some other data types. However, often no data type is shown. This is especially the case if the required data type would be some complex object e.g. representing the BoM for a product. In these cases, most often the attribute's name or its description indicates what the respective data type should look like. Thou, it was not in the scope of this deliverable to detail all this complex data types. Latter is dealt with in other tasks of WP R9. In the model, sometimes a generic "string" data type was specified for those attributes as default, which should not be misunderstood as string being indeed the corresponding data type. In the appendix is a short reference guide to the meaning of the subset of UML syntax used to describe the semantic object model.

4.2 Class Diagram of the Semantic Model: an overview on classes and associations among them

A first insight into the Semantic Object Model can be given as follows (figure 27). Regarding the viewpoints cited in the previous section, two main areas are found in the whole model:

- A first area, bounded by the red, continuous line, comprises on the one hand the basic pieces of information on each product instance, whose information loops the PROMISE end-user wants to be closed, when making use of the technologies developed by the PROMISE project. These pieces of information should reveal some important information such as the serial number of the product instance, the product type, to which it belongs, the product structure of the product if needed, the main properties valid for the product instances, the conditions to be checked on them, etc. In addition, this area also describes the product as a product type. The latter does not represent a focus of the present model,

but the information modelled by it is clearly requested to be managed in most PROMISE application scenarios, and thus must be properly represented. The product type is closely related to the other elements in this area of the diagram, as stated by many associations, and models pieces of information such as the different BOL structures (see DR9.1 [2]), properties, and conditions applicable to the different product types etc.

- A second area, bounded by a blue, dotted line, aims at modelling the pieces of information connected to the different life cycle phases, in which the PROMISE end-user is interested. This enables the description of the main events, out of which a certain life cycle phase is composed (i.e. product failures or breakdowns, replacements of components of a complex product, etc.), of the PROMISE end-user's resources involved in the scenario concerning that life cycle phase (i.e. the garage crew, the designer, the production manager, etc.), and also of the activities performed by these resources in that life cycle phase (e.g. dismantling of a car's components, maintenance of a truck, etc.). Besides that, an important portion of this area is dedicated to the representation of field data, one of the crucial elements in the PROMISE approach.

In the following, a more complete explanation of each of these areas is given, considering the classes involved and the associations among them as well as the most important attributes describing each class in detail. Attributes that are not mentioned explicitly in the explaining text represent information that is more generic. They are usually implemented in very different ways depending on the chosen PDM system underlying the PDKM system.

4.2.1 Product instances across their life cycles: from the identification problem to the representation of the BOL, MOL, and EOL product structures

The PROMISE approach to PLM is a "product instance-centric" one. Each instance of a certain product type should be followed all along its life cycle in order to close the desired information loops, thereby creating value. The concept of PEID is capable of enabling the link to all these product items and their related information. A central portion of the semantic model should thus reflect this approach and properly represent the information on each product at the item level.

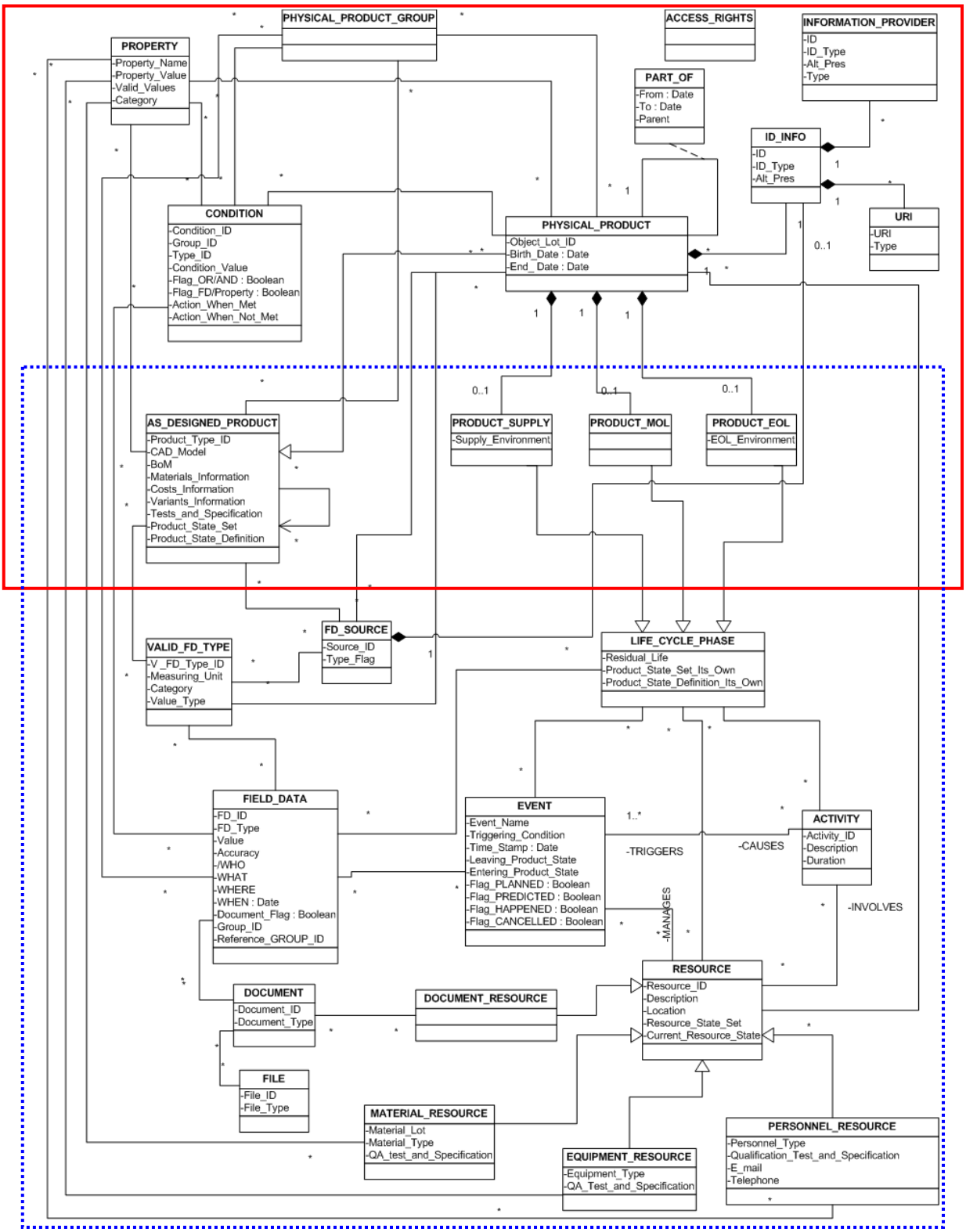

Figure 27: Complete schema of the semantic object model

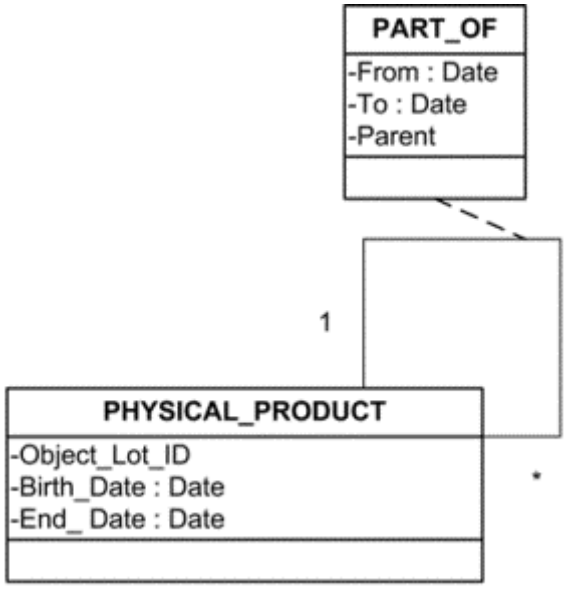

Figure 28: The product structure of physically existing products and the PHYSICAL_PRODUCT class

The PHYSICAL_PRODUCT class (figure 28) is intended to cover this issue by modelling some basic information on each product item as reflected by its attributes:

- *Product_Type_ID* (inherited from AS_DESIGNED_PRODUCT, not shown in figure 28): This is the name of the product type/model/variant (depending on the meaning given to these words in different industrial contexts), to which the product item belongs. It directly corresponds to the value of the *Product_Type_ID* attribute of one and only one object of the AS_DESIGNED_PRODUCT class. More precisely, it states the name of the object of the AS_DESIGNED_PRODUCT class, from which the general attributes of the product as an instance of a product type are derived, as indicated in the diagram by the classification linking the AS_DESIGNED_PRODUCT class and the PHYSICAL_PRODUCT class.
- *Object_Lot_ID*: This shows the ID of the production lot to which the product item belongs, in case it is important to store this information in the specific application considered. A typical example of a situation, in which this information turns out to be important, can be that of a recall campaign that a company must carry out when a serious defect has been detected in one or more products of the same lot, such as some cases happened in the last years in the automotive industry.
- *Birth_Date*: This models the date, at which the existence of the product item actually starts. This can correspond to different time instants in different applications, e.g. the moment at which the product item exits the last station of the production line, or the moment when the product item enters the first stage of that line, etc. The first can be the case of a company, which is not interested in following specifically each product item in the different phases of the production cycle, but e.g. is interested in tracking and tracing the product inside the warehouse or/and across its supply chain. The second can be instead the case of a company, which gives importance to the tracking and tracing of the product all along the production cycle, such as those companies, which daily apply RFID technologies for this reason at the shop floor level.
- *End_Date*: This represents the date, at which the product item reaches its end of life. The cardinality, differently from the previous attributes, which have cardinality of one and only one, is in this case zero to one. This reflects the fact that this attribute is not instantiated until the product item reaches its end of life.

In the case of more complex products, it is very important to track the history concerning the physical components/subassemblies belonging to a single physical product entity. To understand the importance of such an information think as an example on a car whose life cycle has to be managed up to the end of life phase, as well as that of its main components, e.g. the clutch, and subassemblies, e.g. the engine, such as in the FIAT A1 EOL scenario. The PART_OF association class tracks this kind of history.
At the beginning of the car's life, the *From* attribute of the PART_OF association class related to the link between each of these components/subassemblies and the whole product is set to the current date. Then, if at a certain point in the car's life a component has to be replaced by another one of the same type, e.g. because its residual life has expired, its *To* attribute is set to the new current date and, as soon as the new component is attached to the complex product, a new object of the PART_OF class is created and the *From* attribute related to the new component is set to the proper value. This enables the PDKM system to keep an updated list of components/ subassemblies of each product item, as well as a list of old components/subassemblies, together with the related periods of relative belonging:

- *From* (Date): This attribute keeps record of the date and time when the component/subassembly is attached to the product. The cardinality of 1 says that there can only be one value for this attribute.
- *To* (Date): This attribute keeps record of the date and time when the component/subassembly is detached from the product. The cardinality of 0...1 is intended in the sense that this attribute is set to no value at all for the whole time span, during which the component/subassembly is attached to the product, and then is set to its value when it is detached from it.
- *Parent*: This attribute gives the name, i.e. the ID of the father node in the tree corresponding to the structure of a product as a whole. Each node of this tree corresponds to one and only one product item, i.e. a PHYSICAL_PRODUCT. So, it is important to notice that at this level only the product structure for the phases BOL supply, MOL, and EOL, i.e. the structure of physically existing products is actually addressed (see in the following pages for an explanation of these terms; refer also to DR9.1 [2]), while the BOL as-designed structure of the product (see again DR9.1) is treated in an analogue way by the AS_DESIGNED_PRODUCT class. The cardinality of this attribute is one since a PART_OF object is only instantiated if there is indeed a parent-child- respectively assembly-part-relationship between two PHYSICAL_PRODUCTs.

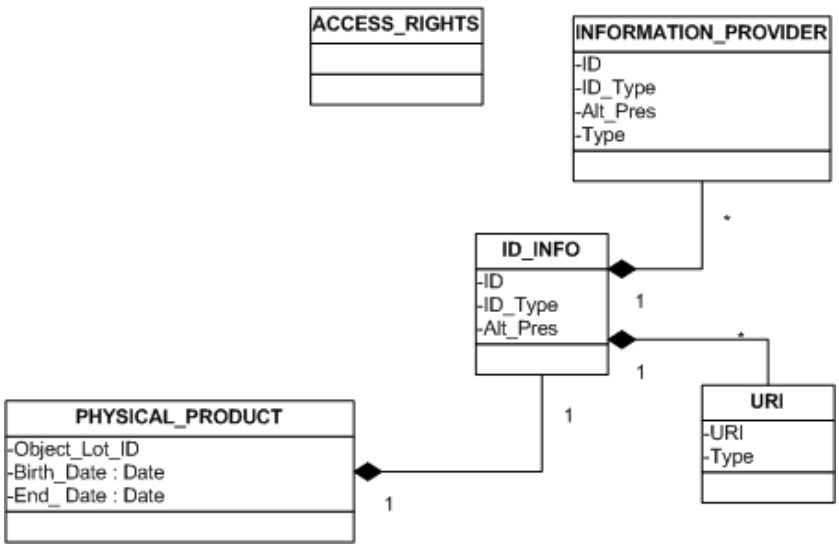

Figure 29: The "identification problem" of product items and the ID_INFO class

The ID_INFO class (figure 29) is important to enable the traceability of the product. The identification of the product instance and the information on where it is possible to retrieve other information on the same product instance are enclosed in this class. Many traceability systems have been developed up to now (e.g. the Dialog System developed by the Helsinki University of Technology, the WWAI-World Wide Article Information concept, the AUTO-ID proposal, etc.), and the ID_INFO class should be compatible with all of them, at least from a conceptual viewpoint. There are two types of links to additional information, URI and INFORMATION_PROVIDER objects. Each ID_INFO object may be linked to zero or many of these objects. URI is a pointer to a source for or a recipient of information regarding the respective PHYSICAL_PRODUCT. Besides that, the URI class can be used for presenting the ID of the respective PHYSICAL_PRODUCT in URI format if needed. The INFORMATION-_PROVIDER object contains information that can be used to control the request for information from e.g. a traceability system, including the inter-enterprise communication system in the PROMISE Middleware (developed by WP R6). However, often such systems, like the PROMISE Middleware, take care of identifying the respective information providers, by this taking this task from the PDKM. The same is possible with the URI information sources.

The ID of a PHYSICAL_PRODUCT is represented as a string in the ID_INFO object. The ID_Type attribute identifies the coding schema of the ID. An attribute Alt_Pres (alternative presentations) can be useful if the ID is shown in different formats for different purposes (machine communication, human readability etc.). The URI object has got a Type that identifies the purpose of the URI.
An ID_INFO object may be linked to zero to many URI objects or zero to many INFORMATION_PROVIDER objects. The URI objects are used for linking external information sources to the ID if relevant (such as in HUT's Dialog approach). INFORMATION_PROVIDER objects are used when the backend application wants to address explicitly a specific information source in the inter-enterprise (IE) communication layer (e.g. another backend system) when requesting information for an instance of PHYSICAL_PRODUCT. Alternatively, this is taken care of by the IE layer of the PROMISE Middleware.

The ACCESS_RIGHTS class (figure 29) represents the part of the PDKM infrastructure that aims at realising and controlling user access control, realising rights and capabilities associated to user profiles and user roles.
These access rights should be managed at different levels and concerning many of the classes represented in this semantic object model. Thus, the presence of the ACCESS_RIGHTS class in the class diagram without connections to any other class is only to underline at a semantic level the need for managing these pieces of information in the PDKM.
The chosen constellation of software tools (refer to the final chapters of DR9.1), especially mySAP PLM and SAP NetWeaver Portal as core systems, provides the PDKM developers with the needed infrastructure. This issue is covered in detail in other tasks of WP R9.

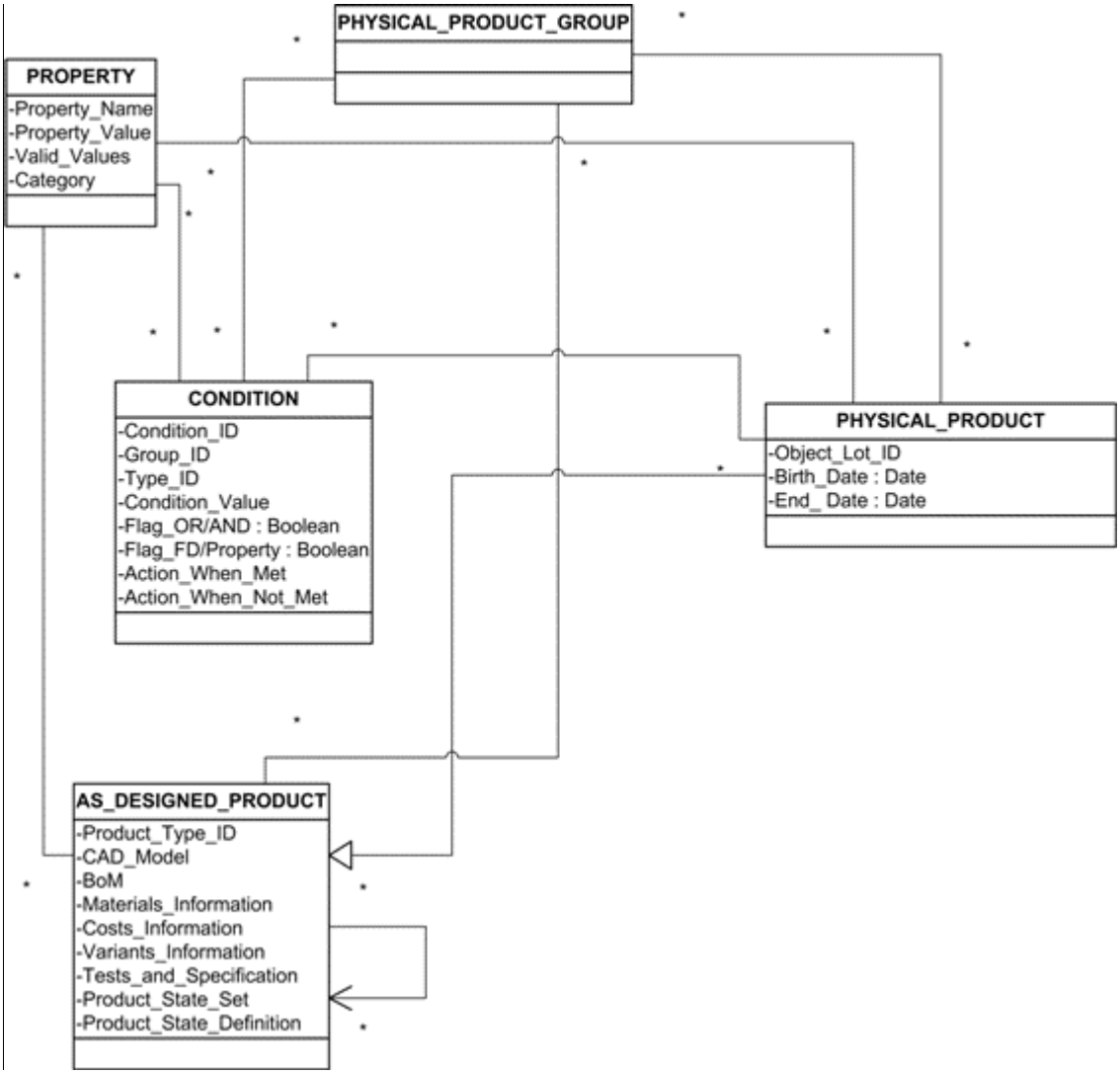

Figure 30: The BOL as-designed product structure and the AS_DESIGNED_PRODUCT class

The AS_DESIGNED_PRODUCT class (figure 30) represents the BOL as-designed structure of the product (compare DR9.1 [2]). The important pieces of information at this level of the model concern the following:

- The product structure has to be represented, if needed. This is permitted by the self-association of the AS_DESIGNED_PRODUCT class.
- The states a product can pass through all along its life must be defined, both stating, which are the different valid states, and also defining the information, from which the present state of the product can be derived, e.g. which is the set of properties of the product that together must be used to understand the current product state. For instance, the fact that the door switch of a home appliance such as a refrigerator is on or off, and the fact that the light inside the same refrigerator is also switched on or off, together with the fact that the power currently absorbed by the refrigerator is "X" watt, can help to understand if the product state is "ALL IS OK", or maybe if it can be something like "LAMP BROKEN" etc.
- The set of conditions (refer to the CONDITION class) and of properties (refer to the PROPERTY class) valid at the product type level, i.e. for each product item belonging to that specific product type, must be referenced.

- CAD models of the product type as well as other common representations following e.g. the STEP or PLCS standards, just to cite two of them, must also be referenced at this level, depending on the kind of representation needed in the single application.
- Other information on materials, on the product variants, on the Bill of Materials, on the tests a product must overcome during its life etc. must also be referenced if needed.

All of these pieces of information are reflected by the attributes and associations of the AS_DESIGNED_PRODUCT class. The most important attributes are:

- *Product_Type_ID*: This attribute shows the identifier of the product type. The cardinality is one and only one.
- *Product_State_Set*: This attribute defines the valid set of strings describing the product state that each product item of that product type can adopt during its life.
- *Product_State_Definition*: This attribute defines the set of information objects/sources i.e. field data types or product properties to use in order to derive the currently valid product state-string out of the *Product_State_Set* (see the refrigerator example above).

In contrary to PHYSICAL_PRODUCT with the PART_OF-classified self-association, there is no dedicated *Parent* attribute for an AS_DESIGNED_PRODUCT. Such an attribute could have been added to a respective association-class for the self-association of the AS_DESIGNED-_PRODUCT. However, it was omitted since this would have been the only attribute of such a class and since it is not required from a modelling approach. Instead, to underline the different roles, parent and child, that AS_DESIGNED_PRODUCTs play in such an association, the self-association is *directed*. In this way, the self-association allows representing a tree corresponding to a BOL as-designed product structure if required. As indicated, this approach differs slightly from the one chosen for the case of the structure for physically existing products discussed in the description of the PHYSICAL_PRODUCT and PART_OF classes. This is mainly because it is not worthwhile to store the information about the duration, for which the product type components/subassemblies are part of a given BOL as-designed structure. When a component is changed from type 'A' to type 'C', usually a new BOL as-designed product structure is easily created e.g. as a variant starting from the previous one, but having the component type 'A' replaced by the component type 'C'. Nevertheless, PLM systems usually provide some kind of revision history for the as-designed product structure. However, this is not in our focus since the structure of a physically existing product, which is in the focus of PROMISE, is derived from one fixed product variant, i.e. one fixed as-designed product structure, respectively. It should also be noted that in contrary to structures for PHYSICAL_PRODUCTs one AS_DESIGNED_-PRODUCT could be part of many as-designed product structures.

The PROPERTY class (figure 30) is intended on the one hand to define properties, which are valid for specific product types or only for some specific product items, as indicated by the associations between the PROPERTY class and the AS_DESIGNED_PRODUCT class and the PHYSICAL_PRODUCT class respectively, but are not common to almost all product types or product items, and thus are not modelled as attributes of the AS_DESIGNED_PRODUCT class or PHYSICAL_PRODUCT class.
Additionally, this class is also intended to model the properties one should know about the resources of the company involved in the PLM scenario and are not common to all resources of the respective resource type, and thus are not modelled as attributes of the respective resource type class. In particular, the class is able to model both general and specific properties of personnel, equipment, and materials (refer to section 4.2.2).
The attributes of the PROPERTY class state the name of the property, the actual value, the allowed/valid values, and the category/type of the property. The latter indicates if the value of this

property is fixed, might be changed arbitrarily, might be changed according to the given Valid_Values etc.
The CONDITION class is intended to represent conditions one wants to check about product types and/or entities. For this reason, the associations between the CONDITION class and the PHYSICAL_PRODUCT_GROUP class (see below) and the PHYSICAL_PRODUCT class respectively are defined, with a cardinality of zero to many, for the sake of generality. Note, that a PHYSICAL_PRODUCT_GROUP object might represent an AS_DESIGNED_PRODUCT. The CONDITION class is linked to both the PROPERTY class and the FIELD_DATA class via associations, since elementary conditions involve the validation of a property or a FIELD_DATA value respectively, i.e. the evaluation of a Boolean expression in the sense that the value of the respective PROPERTY or FIELD_DATA object is checked against a range of values. An example of the semantics of an elementary condition where the measurement coming from a sensor i.e. a FIELD_DATA object is involved might be "if the temperature sensor gives a measurement of more than 150 °C then the product is operating in unsafe conditions: claim for immediate maintenance action".
The class can be used to model both such elementary/atomic conditions and complex conditions. In the latter case, a grouping mechanism is used.
The attributes of the CONDITION class are:

- *Condition_ID*: This identifies the condition among all the objects of the CONDITION class.
- *Flag_FD/Property* (Boolean): This attribute states if an atomic condition involves a field data type or a property of a PHYSICAL_PRODUCT object.
- *Type_ID*: This references, depending on the value of the *Flag_FD/Property*, either the *Property_Name* of the PROPERTY object or the *FD_Type* of the FIELD_DATA object that should be validated. The cardinality is zero to one, indicating that this attribute is only used for elementary conditions.
- *Condition_Value*: This attribute contains the singular value, set of values, or interval that the PROPERTY object or the FIELD_DATA object respectively should be checked against.
- *Group_ID*: This references the *Condition_ID* of a CONDITION object that groups the actual CONDITION object with other CONDITION objects and so building a more complex condition (see the *Flag_OR/AND* attribute).
- *Flag_OR/AND* (Boolean): This attribute is used if the object of the CONDITION class links as a grouping object to other objects of this class in order to build more complex conditions to be evaluated. The linked CONDITION objects are concatenated with AND or OR respectively as indicated by this flag. By using this flag as well as the means for identifying groups of conditions (i.e. the *Group_ID* attribute), one can – following his specific purposes – build whatever kind of complex Boolean expressions built up of elementary conditions in the sense described above. See the example below illustrating the concept.
- *Action_When_Met*: This attribute defines what action has to be carried out when the condition is met, i.e. when the Boolean expression evaluates to TRUE.
- *Action_When_Not_Met*: This attribute defines what action has to be carried out when the condition is not met, i.e. when the Boolean expression evaluates to FALSE.

The cardinalities are, except from the *Condition_ID* attribute with cardinality one, zero to one for every attribute of this class, mostly depending on if the respective CONDITION object models an elementary or a complex condition as described exemplarily for the *Type_ID*.

An example illustrates how to group more than one condition together in order to build up conditions that are more complex. Imagine that the following condition has to be checked for a product: "If the *residual life of the component X is less than 30 days* and *either the maintenance crew A* or *the maintenance crew B is available*, then …"

This is equivalent to say that the following composed condition has to be checked:

C1 AND (*C2* OR *C3*)

with

- *C1*: *The residual life of the component X is less than 30 days.*
- *C2*: *The maintenance crew A is available.*
- *C3*: *The maintenance crew B is available.*

This is represented by the object model with five objects of the CONDITION class, briefly outlined in the following with the respective set of attributes (only the ones directly concerning the creation of the complex/composed condition are mentioned):

Object #1 representing *C1*:

- *Condition_ID*: 1
- Essence of *Flag_FD/Property*, *Type_ID*, and *Condition_Value:* "Residual Life of component X is < 30 days"
- *Group_ID*: 5
- *Flag_OR/AND*: (none)

Object #2 representing *C2*:

- *Condition_ID*: 2
- Essence of *Flag_FD/Property*, *Type_ID*, and *Condition_Value:* "Maintenance crew A is available"
- *Group_ID*: 4
- *Flag_OR/AND*: (none)

Object #3 representing *C3*:

- *Condition_ID*: 3
- Essence of *Flag_FD/Property*, *Type_ID*, and *Condition_Value:* "Maintenance crew B is available"
- *Group_ID*: 4
- *Flag_OR/AND*: (none)

Object #4 representing the OR-join:

- *Condition_ID*: 4
- Essence of *Flag_FD/Property*, *Type_ID*, and *Condition_Value:* (none)
- *Group_ID*: 5
- *Flag_OR/AND*: OR

Object #5 representing the AND-join:

- *Condition_ID*: 5
- Essence of *Flag_FD/Property*, *Type_ID*, and *Condition_Value:* (none)
- *Group_ID*: (none)
- *Flag_OR/AND*: AND

The value 4 of the *Group_ID* of *C2* and *C3* references to the *Condition_ID* of CONDITION object #4. Taking into account the value of *Flag_OR/AND* of object #4 evaluating this object is equivalent to evaluate

C2 OR *C3*

Similarly the value 5 of the *Group_ID* of *C1* and object #4 references to the *Condition_ID* of CONDITION object #5. Thus, again taking into account the respective value of *Flag_OR/AND* evaluating object #5 is equivalent to

C1 AND (evaluation of object #4)

which is equivalent to

C1 AND (*C2* OR *C3*)

In this manner, arbitrary Boolean expressions can be modelled as long as the basic elements are of the form of the elementary conditions mentioned above.

Finally, the PHYSICAL_PRODUCT_GROUP class is intended to model the possibility for the PDKM system and its users to group together a set of PHYSICAL_PRODUCT objects on the basis of some commonalities e.g. concerning their BOL as-designed structure or the field data collected. This is stated by the associations linking the PHYSICAL_PRODUCT_GROUP class with the AS_DESIGNED_PRODUCT class, the PHYSICAL_PRODUCT class, and the FIELD_DATA class. Think as an example on a quality assessment report modelled as field data (see below for the explanation of the FIELD_DATA class). Most of this kind of data is valid for a whole set of PHYSICAL_PRODUCTs, i.e. a PHYSICAL_PRODUCT_GROUP.

4.2.2 Life cycle related information: description of life cycle phases and the related field data

The information on product life cycle phases has to be modelled from a conceptual viewpoint. This section presents the classes, attributes and associations implied by the PROMISE PLM vision and approach. Some of this information is common to all life cycle phases of a product and can thus be modelled by the same class and set of attributes, whereas some other information cannot. Three different classes, namely the PRODUCT_BOL_SUPPLY class, the PRODUCT_MOL class and the PRODUCT_EOL class, have been defined for this purpose.
The PRODUCT_BOL_SUPPLY class (figure 31) contains the pieces of information, which are specific to the SUPPLY phase of the product, and which cannot be described by using the LIFE_CYCLE_PHASE class or the EVENT, RESOURCE or ACTIVITY classes (figure 32). The attribute *Supply_Environment* represents a reference to the environment/information required further during the SUPPLY phase; that might be arbitrarily complex. Thou, the *Supply_Environment* is not in the focus of this deliverable since it is usually sufficiently covered by the PLM system underlying the PDKM or other, connected systems. The zero to one cardinality of the association between the PHYSICAL_PRODUCT class and the PRODUCT_BOL_SUPPLY class states that in application scenarios where the SUPPLY phase is not explicitly managed, the PRODUCT_BOL_SUPPLY class has not to be instantiated.

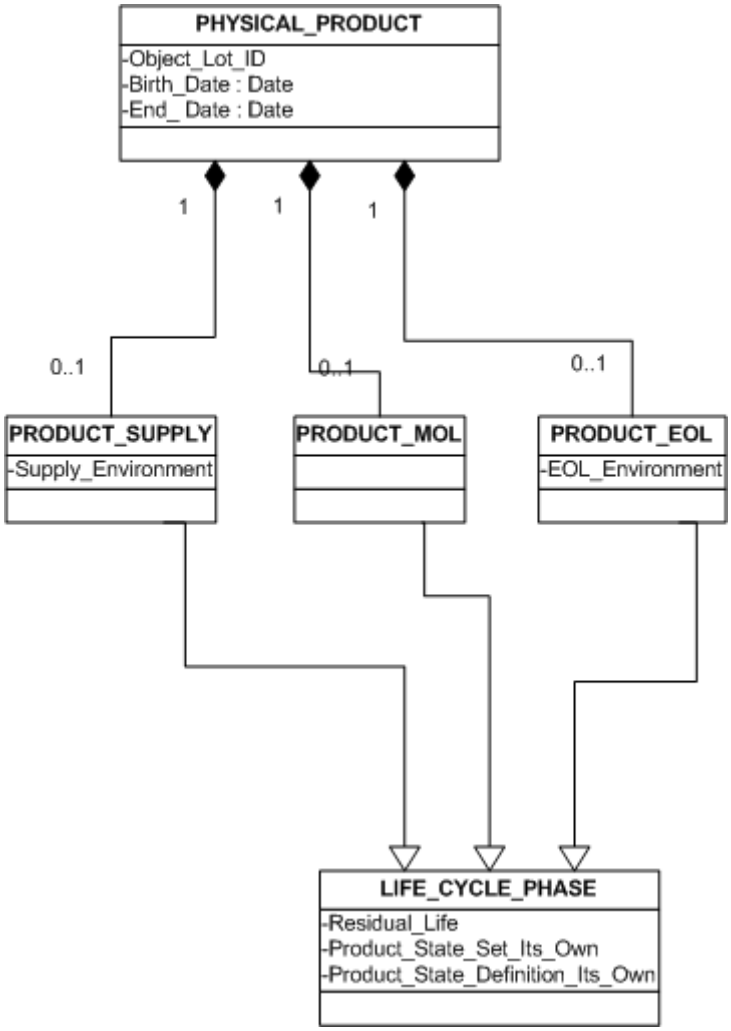

Figure 31: Product life cycle information: different classes for different purposes

As SUPPLY phase, it is intended the set composed of the product production phase and the product delivery phase. From the BOL phase as intended by the PROMISE consortium, only these two phases have been modelled in this sense, while the product design phase has not been explicitly modelled. This was mainly due to the following reason:

The life cycle phases covered by PRODUCT_BOL_SUPPLY refer to existing product items, while the design phase is related to the product type concept, which is addressed by the AS_DESIGNED_PRODUCT class. These two categories essentially differ in the sense that for existing product items there can be retrieved field data in the sense of measurements, events that the physically existing product is involved in, and similar, which is a core input in all PROMISE application scenarios. This is not the case for products that exist "only electronically" as a product design.
In order to collect all this data that is created during the life of a physically existing product, the LIFE_CYCLE_PHASE class with its sub-classes has been introduced. For an AS_DESIGNED_PRODUCT no data needs to be collected during the design phase but all relevant data already exist when the PROMISE application scenarios start and thus can readily be associated to the respective AS_DESIGNED_PRODUCT object, respectively – as indicated in the model – assigned to the corresponding attributes of this object.
This is not in contradiction to the application scenarios where the generated knowledge influences the design process since these influences are covered by new AS_DESIGNED_PRODUCT objects e.g. variants of already designed product types, while the existing

AS_DESIGNED_PRODUCT objects are not changed. Please also note that modelling the design process of a product is not in the focus of the PROMISE project.

It is worth noticing that these two phases that are covered by PRODUCT_BOL_SUPPLY are in the focus of most of the current applications of product identification technologies, of which the RFID applications constitute one of the most known examples. Also in the PROMISE project some application scenarios are closely related to e.g. the production phase or the logistics management inside the warehouse.

The PRODUCT_MOL class and the PRODUCT_EOL class play the same role as the PRODUCT_BOL_SUPPLY class but are related to respectively the MOL and to the EOL phase of a product's life. The attribute *EOL_Environment* of PRODUCT_EOL corresponds to *Supply_Environment* of PRODUCT_BOL_SUPPLY, thereby representing a reference to the respective further required environment/information. The same remarks apply as for *Supply_Environment*.

Two last notes on this set of "life cycle classes":

- Each scenario needs a different number of objects of these classes to be instantiated, e.g. a scenario purely devoted to predictive maintenance may need only the object of the PRODUCT_MOL class to be created such as in the MTS case of WP A7. On the contrary, another scenario might need to model only the production phase of the product, thus having the need of instantiating the object of the PRODUCT_BOL_SUPPLY class, e.g. the CRF BOL case of WP A11. Again, another scenario may need both the PRODUCT_BOL_SUPPLY class and the PRODUCT_MOL class to be instantiated, e.g. the WRAP case of WP A8.
- The more the application scenario is complete, i.e. the longer the portion of the product life cycle considered in the scenario is, the greater is the number of these classes to be instantiated. In case all of them are needed, they are instantiated one after another, following the product item's life, with the related shifts from one life cycle phase to another, acting as the trigger for the instantiation of the object of the immediately following life cycle phase.

The FIELD_DATA class (figure 32) is another crucial class in the semantic model in the sense that it enables the overall PROMISE approach to Product Life Cycle Management by collecting data from the field, thus, also enabling the improvement of product performance and in general the creation of economic value from PLM activities.
Field data can be of different types (VALID_FD_TYPE class), and is collected by means of sources like e.g. sensors (FD_SOURCE class). It might be organized in documents (DOCUMENT class) with attached physical files (FILE class). The associations among these classes show the most important existing links; the cardinalities are all zero to many, to take into account the most general cases.

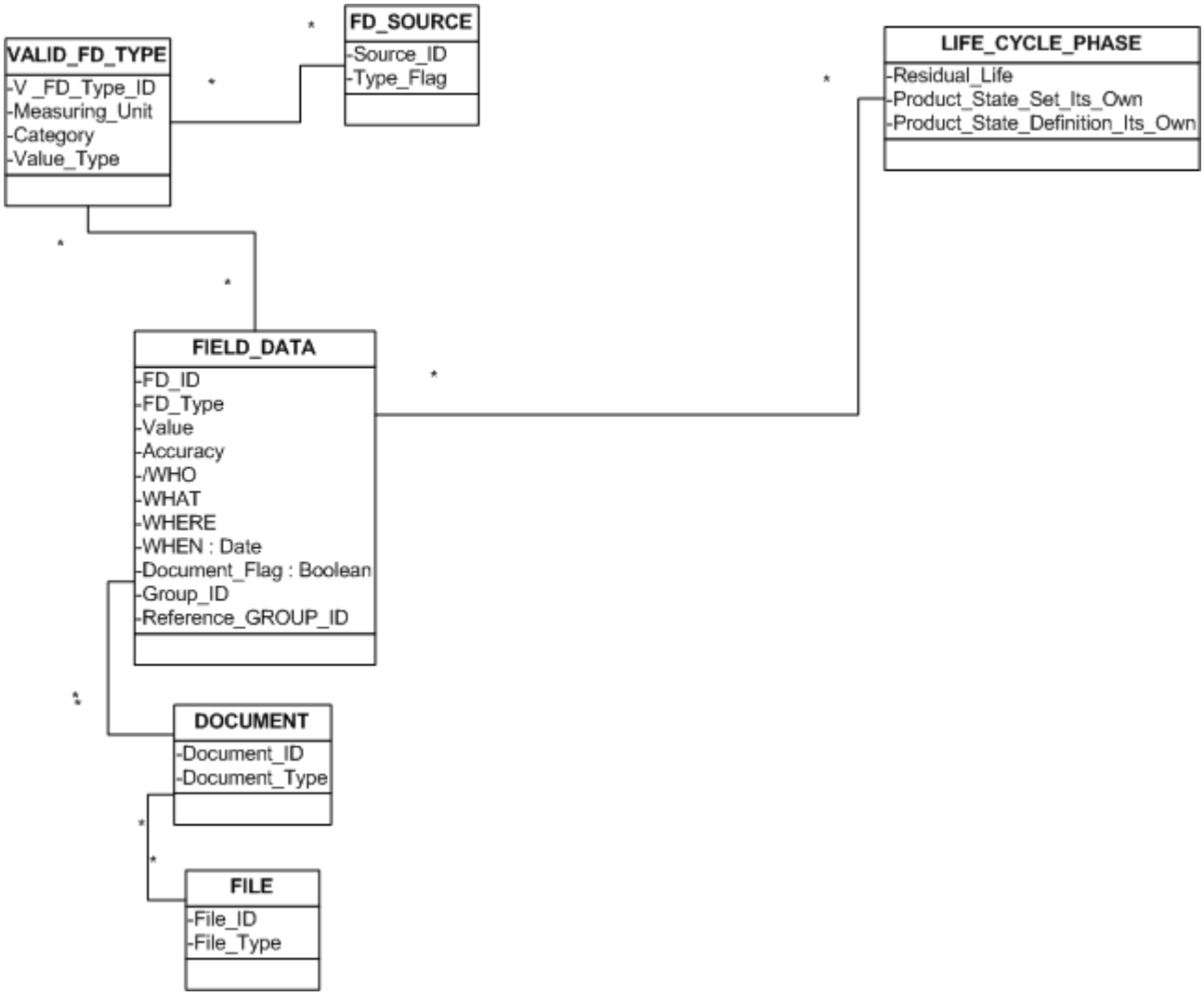

Figure 32: The FIELD_DATA class and its relationships with other model components

The attributes belonging to the FIELD_DATA class are:

- *FD_ID*: This attribute represents the identifier of each single field data record. The cardinality is one and only one.
- *FD_Type*: This attribute shows the type of the field data, and directly corresponds to a specific object of the VALID_FD_TYPE class, i.e. the string defining the FD_Type must be equal to the string defining the ID of an object of the VALID_FD_TYPE class. The cardinality is also one and only one.
- *Document_Flag* (Boolean): This attribute states if the field data is contained in or represented by an attached document (refer to the DOCUMENT class for more information).
- *Value*: This contains the value of the field data record. The cardinality is zero to one depending on the *Document_Flag*.
- *Accuracy*: With cardinality zero to one, this attribute states the accuracy of the field data measurement if needed.
- */WHO*: This attribute shows who is responsible for the field data measurement, i.e. which is the source of the field data. It can be derived (in UML indicated by "/" preceding the attribute name) e.g. from a corresponding object of the FD_SOURCE class linked with the object of the VALID_FD_TYPE class that is associated to the present FIELD_DATA object.

- *WHAT*: This attribute can for instance explain what the field data stands for, i.e. the meaning of the data itself.
- *WHERE*: This attribute states the location where the measurement was made if needed by the specific application scenario (the cardinality is zero to one), i.e. the location where the product was situated when the present field data was collected.
- *WHEN* (Date): This is simply the timestamp indicating the moment when the field data was recorded.
- *Reference_GROUP_ID*: The usage of this attribute is very similar to that of the *Group_ID* attribute of the CONDITION class (see related example above). It references the *FD_ID* of an associated FIELD_DATA object that keeps some information regarding a group of FIELD_DATA objects. On the contrary to the CONDITION class, FIELD_DATA objects might be grouped without such a group-representing object if it is not necessary to keep additional information with respect to the group. In these cases, the following *Group_ID* is used. Thus, the cardinality of the attribute is zero to one.
- *Group_ID*: This attribute gives the ID of the group of field data, to which this field data record belongs, if such a group exists. The cardinality of the attribute is zero to one. In contrary to the *Group_ID* attribute of the CONDITION class, it does not reference another FIELD_DATA object (see the preceding attribute *Reference_GROUP_ID*).

Some last considerations on the FIELD_DATA class are:

- The FIELD_DATA class is a very generic class, which can be utilized in different applications, for different purposes and different meanings. For instance, *information*/*knowledge* generated by the DSS and to be stored in the PDKM might be treated in the present model as a single field data record or a group of field data records; if suitable, it might also be stored in a proper format in documents attached to field data records. Another example, which is treated in the model as a field data record, is the current product state-string e.g. derived from a group of field data records by applying the state definition valid for that product type/product item.
- The *WHERE* information can become very useful, together with the */WHO*, *WHAT* and *WHEN* pieces of information, when trying to realize the complete (and, in some cases, the "real time") product traceability.

The VALID_FD_TYPE class is intended to model the information concerning the type of a given field data object e.g. “Temperature_Sensor_1” or “Average_Temperature_Sensor_2”, its set of categories e.g. {”Temperature Measurements”, “Calculated Values”}, its measuring unit, e.g. K, W, N, kg, m, etc., and the data type, e.g. integer, double, float, string etc. These pieces of information are provided all with cardinality of one and only one or zero to one depending on the data type.
The VALID_FD_TYPE class is associated to both the PHYSICAL_PRODUCT class and to the AS_DESIGNED_PRODUCT class. Both associations have zero to many cardinalities, to state that a field data type might be defined on the one hand on the level of a product type, and so is also inherited by all derived product items, and on the other hand for a specific product item only. These associations also support the case that recorded field data e.g. a test report is attached to an AS_DESIGNED_PRODUCT object via a PHYSICAL_PRODUCT_GROUP and thus this field data is also inherited by product items derived from this AS_DESIGNED_PRODUCT object after the field data was recorded. The VALID_FD_TYPE class is also attached to the PHYSICAL_PRODUCT_GROUP class, e.g. for the case that all products (locomotives, boilers, etc.) of this group have attached a specific sensor.

The FD_SOURCE class is used to define the identity of the source of a given valid field data type. This is shown in the following attributes:

- *Source_ID*: This is the ID of the source of field data; for instance, it can represent the identity of the sensor of the PEID being responsible of the measurement of that specific field data, e.g. "sensor #5 linked to the on-board computer", which performs measurements over time of the temperature, at which the product operates. The cardinality of this attribute is one and only one. Besides that, there is an optional aggregation-association (cardinality zero to one) to the ID_INFO class (see figure 27) allowing for FD_SOURCEs the same options regarding addressability beyond the PDKM boundaries as discussed for the association between PHYSICAL_PRODUCT and ID_INFO.
- *Type_Flag*: This attribute defines, if needed in the application scenario (cardinality is zero to one), the type of the source for the considered valid field data type, e.g. "thermocouple sensor".

Finally, note that the FD_SOURCE class is also linked with the AS_DESIGNED_PRODUCT class and to the PHYSICAL_PRODUCT class via associations (see figure 27). This is to state that for each product type or even each product item, the list of e.g. sensors, with which it is equipped, can be given if required by the application scenario.

The DOCUMENT class defines in this context a document containing field data if the *Document_Flag* in FIELD_DATA is set to TRUE. A DOCUMENT object serves as a kind of container for one or more FILE objects. Respective metadata can be assigned to its attributes. The FILE class finally models the physical file that is attached to the DOCUMENT and actually stores the field data content.

The DOCUMENT and FILE classes are also used in the same way in other contexts.

The LIFE_CYCLE_PHASE class (figure 33) contains the life cycle information of the product item that is common to all the different life cycle phases. The attributes shown in the class are, as also in other parts of the present model, not intended to be a complete list of all the possible attributes, but on the contrary they aim at representing a good example of the information to be included in this class, when instantiating the model in a single application scenario. In figure 33, the following attributes are considered:

- *Residual_Life*: This represents the residual life of the product item. This information can be updated, maybe in different ways and at different steps of a product's life, and for different reasons. In some cases, it is one of the most important pieces of information to be considered as the main support to life cycle decisions, such as in some EOL scenarios, even in the PROMISE project. The fact that in each moment of a product's existence it is possible to define a value for this attribute drove the modeller to put it into the LIFE_CYCLE_PHASE class instead of putting it into one or more of the "single phase" classes. Interesting information can be the number of time units, e.g. seconds, minutes, or hours, depending on the case, representing the residual life of the product, and the time stamp, expressed by a “date” data type, from which the residual life must be counted. The cardinality of this attribute is one and only one.
- *Product_State_Set_Its_Own*: This attribute defines the set of product state-strings, which are valid specifically for the current PHYSICAL_PRODUCT object in the current life cycle phase, in addition to the states common to all the product items of the same product type if this is needed in the considered scenario. Thus, the cardinality is zero to one.

- *Product_State_Definition_Its_Own*: This attribute defines the pieces of information needed to define the current product state-string of a product item, but specifically defined for the current PHYSICAL_PRODUCT object in the current life cycle phase if needed in the considered scenario. The cardinality here is also zero to one.

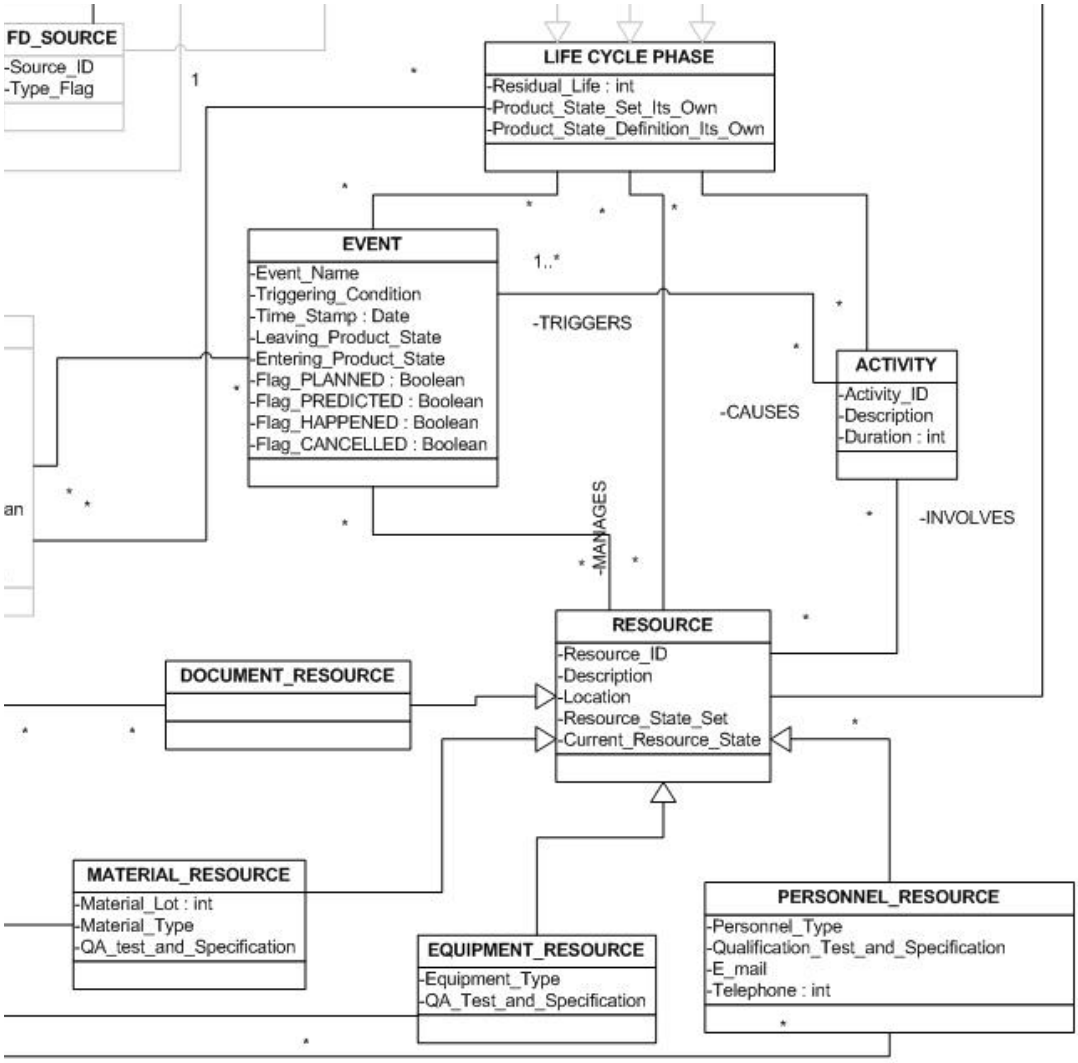

Figure 33: Describing the life cycle phases of a product: events, resources, activities, and related classes

It should be noticed again that the state describing attributes mentioned above are in the LIFE_CYCLE_PHASE class and not in the PHYSICAL_PRODUCT class since these attributes might have different instantiations in each life cycle phase.

Up to this point, none of the mentioned modelling elements has been intended to describe each single life cycle phase in the way that it is intended to be managed, i.e. no element represented the main events happening during a life cycle phase, the people and other kinds of resources of the company, which are involved in the life cycle phase, and the activities performed during this phase. These issues are addressed by the EVENT, RESOURCE, and ACTIVITY classes, which provide the PROMISE user with powerful means of describing everything he wants concerning one specific phase.
For instance, in a predictive maintenance scenario, such as those in the PROMISE project, one would like to model the event of breakdown of a component/subassembly or even of the entire

product as a whole as well as the maintenance activities and the resources, human and not human, involved in such activities. The aim could be for example to predict the point in time when the product will probably break down, to plan the related maintenance activities, and finally to record the actual time instant when the breakdown really occurs, or even to delete the breakdown event from the "list of predicted events", just because the component causing the possibility of product breakdown has been replaced by a new one, which eliminates the main cause of breakdown. The repair activities at the garage of the maintenance provider should also be first described and then properly managed in accordance with the corporate strategy of the company and in a PLM vision. So the availability of resources like free hours of the garage crew to be possibly allocated, or even the availability of the needed materials and equipments to perform the maintenance activities should be checked, and eventually, such as in some PROMISE application scenarios, the maintenance activities should be economically planned and managed.

For these purposes, the RESOURCE, ACTIVITY, and EVENT classes with a certain set of attributes have been added to the model. In addition, three associations, one for each pair of these classes, have been added to state that an event triggers an activity, which involves some resources, which in turn manage the event as an important part of a specific life cycle phase. The attributes state that an event is something related to a specific time instant, while an activity generally concerns a time interval and is thus associated to a duration in terms of time. An activity has at least two events associated with it: the event "ACTIVITY STARTS" and the event "ACTIVITY ENDS". The event is triggered by some kind of condition and causes in general the shift of the product state from some "STATE A" to another "STATE B". Again, one should have the possibility, as written above, to mark with a proper flag if the event is a planned event, or if it is a predicted event, or again if the event has already happened, or has been cancelled because it cannot happen anymore (refer in figure 33 to the Flag attributes in the EVENT class). In addition, an activity can cause an event, such as the maintenance activity can cause the "REPLACEMENT OF COMPONENT *XYZ* PERFORMED", with a consequent update of the product's residual life that, if not anymore under the minimum threshold, causes the "PRODUCT BREAKDOWN" event to be cancelled, such as in the example above. Finally, the resources can be human beings (PERSONNEL_RESOURCE class), equipments (EQUIPMENT_RESOURCE class), materials (MATERIAL_RESOURCE class) and documents (DOCUMENT_RESOURCE class). Some of the information related to these resources is given as attributes, and some other kind of information is specified as objects of the PROPERTY class. Some important examples can be the maintenance crew as objects (e.g. one for each person) of the PERSONNEL_RESOURCE class, the tools for performing the maintenance activities as objects of the EQUIPMENT_RESOURCE class, the spare parts needed as objects of the MATERIAL_RESOURCE class, and, finally, the product user manual, the maintenance manual, or the CAD model of the product layout as objects of the DOCUMENT_RESOURCE class.

Moreover, for each resource a set of possible states is defined, and the current state is recorded. This is required for example in cases where the information on the availability of the garage all along a certain time period can be very important to plan the maintenance activities at the garage, such as in the CRF MOL scenario WP A4 about predictive maintenance of a fleet of trucks. Hence, two states such as AVAILABLE and NOT AVAILABLE can be defined as exhaustive and mutually exclusive states, and the setting of the product state of the garage to one or the other of these values can be used to understand if a given time interval can be assigned to the maintenance of a specific product item or not.

In conclusion, there also exists an association between the RESOURCE class and the PHYSICAL_PRODUCT class to state that it sometimes can be possible that the object of the PLM system, which is a resource for one company, e.g. a truck used for the delivery of the products produced by the company, may be a product item for another company, e.g. it can be part of a fleet of trucks, on which the truck builder/dealer performs predictive maintenance. Such a scenario is up to now not so realistic since both companies probably would not share the same

PDKM. Nonetheless, it is interesting to notice that the boundary between an object as a resource and an object as a product itself, as implied by the PROMISE vision, is not so well defined and even not so thick as one would think.

5 The Technical Data Schema

As indicated in section 1.4, the focus of this deliverable is the semantic object model. Nevertheless – as already mentioned in DR9.1 – when talking about data models, there has to be distinguished between the semantic and the technical data model. The latter is the focus of this chapter. However, the driving question for this chapter is not concerning "what the technical data model is", but rather, "why to distinguish between a semantic and a technical model" and "what are the relevant aspects with respect to this".

A starting point for developing a technical data model is that it is identical to the semantic model and then necessary deviations are identified. This chapter gives an impression of important aspects with respect to the development of the technical data model but has not the ambition of complete coverage of this subject.

5.1 Mapping to mySAP PLM

A driving factor of the technical data model is that it is not intended to build the solution from scratch but to base the PDKM on an already existing PDM system, here the mySAP PLM (see chapter 2.2). Thus first deviations are due to that the use of the abilities from mySAP PLM has been decided, hence there is a mapping from the semantic model described in the chapter 4 to entities of mySAP PLM.
The mapping of the most important entities is given in table 1 of deliverable DR9.6b "Specification of System Functions (revised)" [8]. However, this table is not, and does not intent to be, complete in the sense that it covers *all* elements of the semantic model. It has also to be understood that this table shall not be understood as a *1:1* mapping. E.g., the SAP-entity *Material* that is listed there does not readily cover all attributes from the referenced AS_DESIGNED_PRODUCT, but *Material* can be associated with all relevant information.
The entities of mySAP PLM that are mentioned there have themselves an associated technical data model since these semantic entities of the mySAP PLM have a corresponding implementation using a technical schema. However, from the PDKM point of view – not concerning the DSS as a component of the PDKM – this underlying technical schema is not of interest, since the corresponding API provided by mySAP PLM is used.
As far as other work packages, such as WP R8 developing the PROMISE DSS, need more detailed knowledge about the underlying technical data model of mySAP PLM they get the required information on detailed request to WP R9. There is no document readily available for public use that covers the whole technical data model from mySAP PLM but there exist mechanisms to retrieve the necessary information by using the system itself.

5.2 Purely technical information

Besides this aspect of the technical data model concerning the mapping from the developed semantic model to mySAP PLM entities there is another aspect regarding the technical model, i.e. the purely technical information.
Again, this deliverable is not the right place to list all aspects regarding this, instead there are examples given to illustrate what this aspect is covering:

- Each association given in the semantic model has to be realized by referencing from one object to the other by using its unique keys. If such a key is not defined in the semantic model, a technical ID has to be introduced. Uniqueness in this context is meant with respect to the object-type; i.e. the tuple [object-type, key] uniquely identifies the object in

the technical data model. Such a technical ID might also be introduced if this is easier to handle than e.g. a key composed of several attributes.

- There have to be technical entities that ensure the uniqueness of newly generated object-keys.
- Often, new objects have to be introduced e.g. for implementing *n:m* relations, new tables, that realise these associations by storing the tuples [ID of object 1, ID of object 2].
- What is treated in the semantic model as an object-attribute-relation is often mapped in the technical model to an object-object-relation. E.g., the object AS_DESIGNED_PRODUCT has as an attribute CAD_Model, but from a technical point of view, these CAD-models are not dealt with as attributes but as objects that are referenced from the AS_DESIGNED_PRODUCT.
- The same holds if there are alternative specifications given for an attribute type e.g., /WHO in FIELD_DATA might be a string or a reference to a FD_SOURCE. In such cases, it is sometimes easier from a technical point of view to treat /WHO as a separate object instead of an attribute in the FIELD_DATA object. Another possible solution is introducing a flag that indicates the usage of an attribute.
- In an analogue way, it is often more efficient to associate an object to each member of a group of information instead of saving a set of information as an attribute of that object.
- Semantic objects sometimes drop out completely since all their information is easier stored in objects referenced by them. Nevertheless, these objects ease the understanding of the semantic model, so they are kept there.
- Often, objects are added for storing all allowed values e.g. for the state-string for a specific product.
- Objects might be introduced to avoid duplicating contents in many instantiations of a class.
- Usually, there are purely technical data schemas that are necessary for formatting the output presented to the user e.g., how a generated report is formatted.
- As mentioned in the introduction the focus of this deliverable's semantic model is the management of any product related data. But as described in DR9.1, there is also a lot of more or less technical information such as:
    - the mapping between the data model of the Data Management layer and the data model of an input source
    - information required to run import jobs
    - information needed for data processing
    - data for the technical connection to PDKM-external systems etc.

  Due to its technical nature, this is seen as part of the technical data model.
- Performance issues might suggest deviations between the technical and the semantic data model. An example is that it might be critical for "real life" applications to store all field data in a single data base table as indicated by the semantic model. If this is the case, a solution might be to distribute the data over several tables. For example, there can be single tables for each type of values, i.e. string, integer, float, etc. However, data can also be spread over separate tables, e.g. by the ID of the PHYSICAL_PRODUCT, to manage the size of these tables. Thou, it shall be mentioned that performance is not focus of the PDKM prototype targeted in the duration of the PROMISE project.
- Frequently, a direct mapping of inheritance of objects to the technical schema is inefficient. A more pragmatic approach is then to treat objects inheriting from the same class as completely independent objects.
- Last but not least, it should be noted – as mentioned in section 4.1 – that the semantic model does not need to cover all details since they are not necessary for its purpose. The technical model, however, implements every necessary detail.

6 Mapping the Semantic Object Model to real cases: some examples from the PROMISE application scenarios

In this section, a first verification of the semantic object model is shown by applying the General System Object Model to some representative cases chosen among the eleven PROMISE application scenarios. Since no formal verification technique can be applied to the conceptual model proposed above, this way of proceeding represented the best way to demonstrate to the reader, e.g. to the PROMISE application partners, the real modelling capabilities of the proposed model for the purposes of the whole PROMISE project.

Three cases were chosen among the application scenarios, one case for each of the three macro-life cycle phases (BOL, MOL, and EOL). In particular, the same applications already chosen in other PROMISE work packages for prototyping purposes were considered. In the following subsections, a description of how the proposed model can provide the means to represent the most important issues related to the studied scenarios is shown by using the Object Diagram framework of UML 2.0 (refer to the appendix for more information on object diagrams).

To be noted that, as already mentioned in section 1.3, the semantic model was developed using an iterative approach by applying the model under development alternatively to all the different application scenarios and for the purposes defined by the PROMISE technology's end-users in [5] with good results in the whole set of scenarios.

In the following, the section about the first application scenario contains some more examples described less detailed in order to give in form of a static snapshot an overview of the modelling power of the semantic model. On the contrary, the sections about the two remaining scenarios depict certain aspects but describe those much more detailed in order to foster the understanding of how the chosen entities "work" in the respective scenario.

To keep the figures easy to understand, objects and attributes of minor importance for the respective example have been omitted respectively have not been filled with corresponding values.

6.1 BOL scenario: the BOMBARDIER case from WP A10

This chapter gives an example for a possible application of the semantic object model to a BOL application scenario. The Bombardier scenario (see [5]) has been chosen to demonstrate the applicability of the model. The example does not provide a complete instantiation of the model, but gives possible applications of the model to important domains.

6.1.1 As-Designed Product Structure

The object diagram in figure 34 illustrates a part of a possible as-designed structure for a converter. The same object type (AS_DESIGNED_PRODUCT) is used to represent design information of the components of a product at any level.

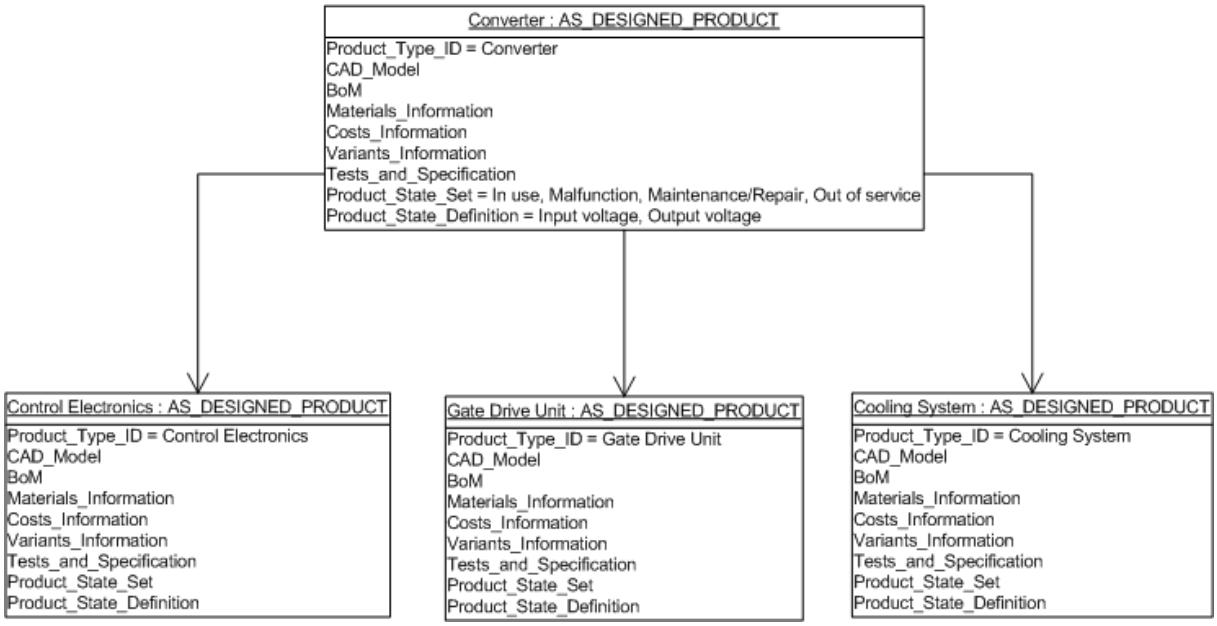

Figure 34: As-designed structure for a converter

6.1.2 As-Used Product Structure

The as-used product structure holds information about the "product instance" (actual, physically existing product). The example in figure 35 shows a possible, simplified instantiation of the converter. The components of the product instance are connected via an association object (PART_OF), which holds historical configuration information. It can be seen that the cooling system has been replaced once.

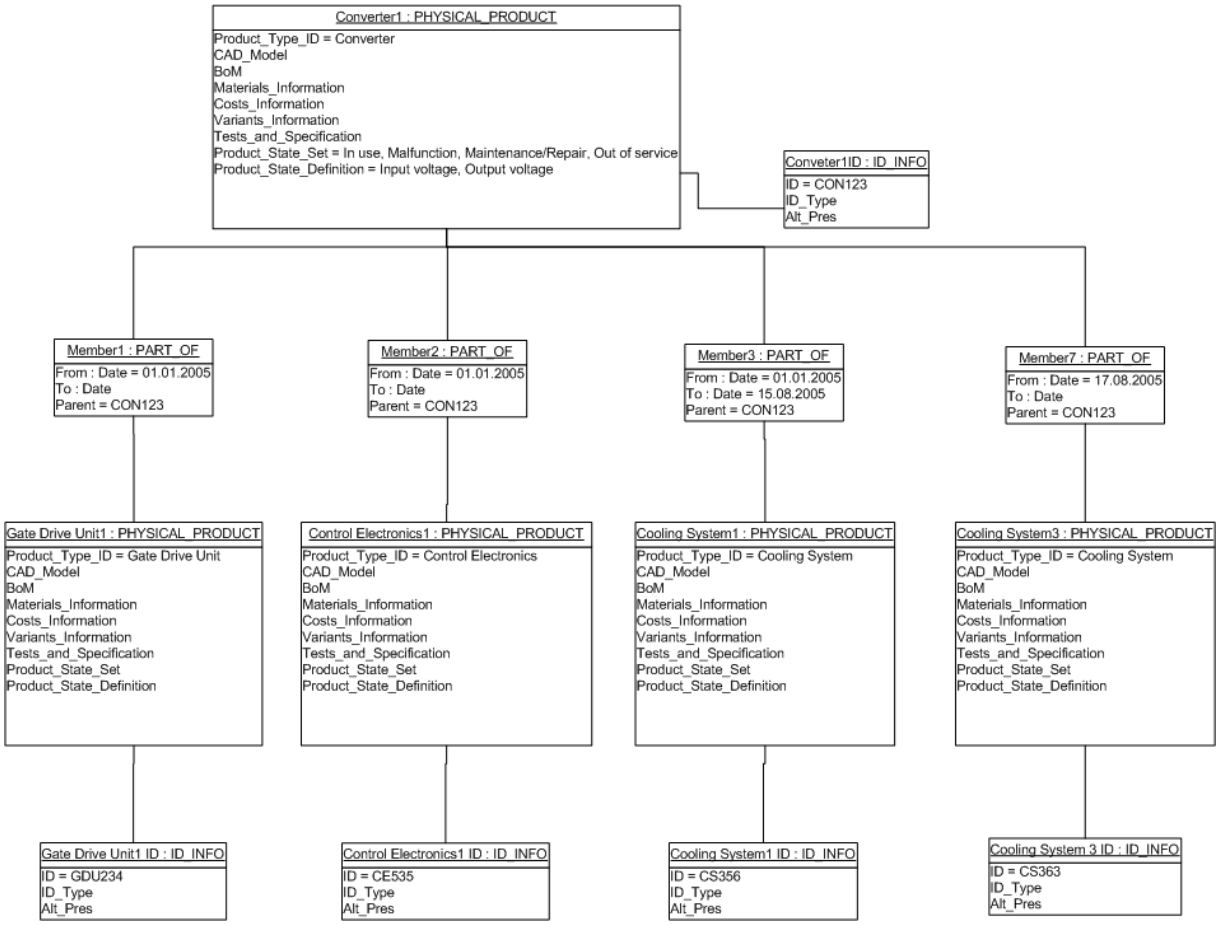

Figure 35: As-used structure for a converter with component replacement

Figure 36 shows the instantiation of a second converter to illustrate that there might be numbers of instantiations not related to each other for the same AS_DESIGNED_PRODUCT.

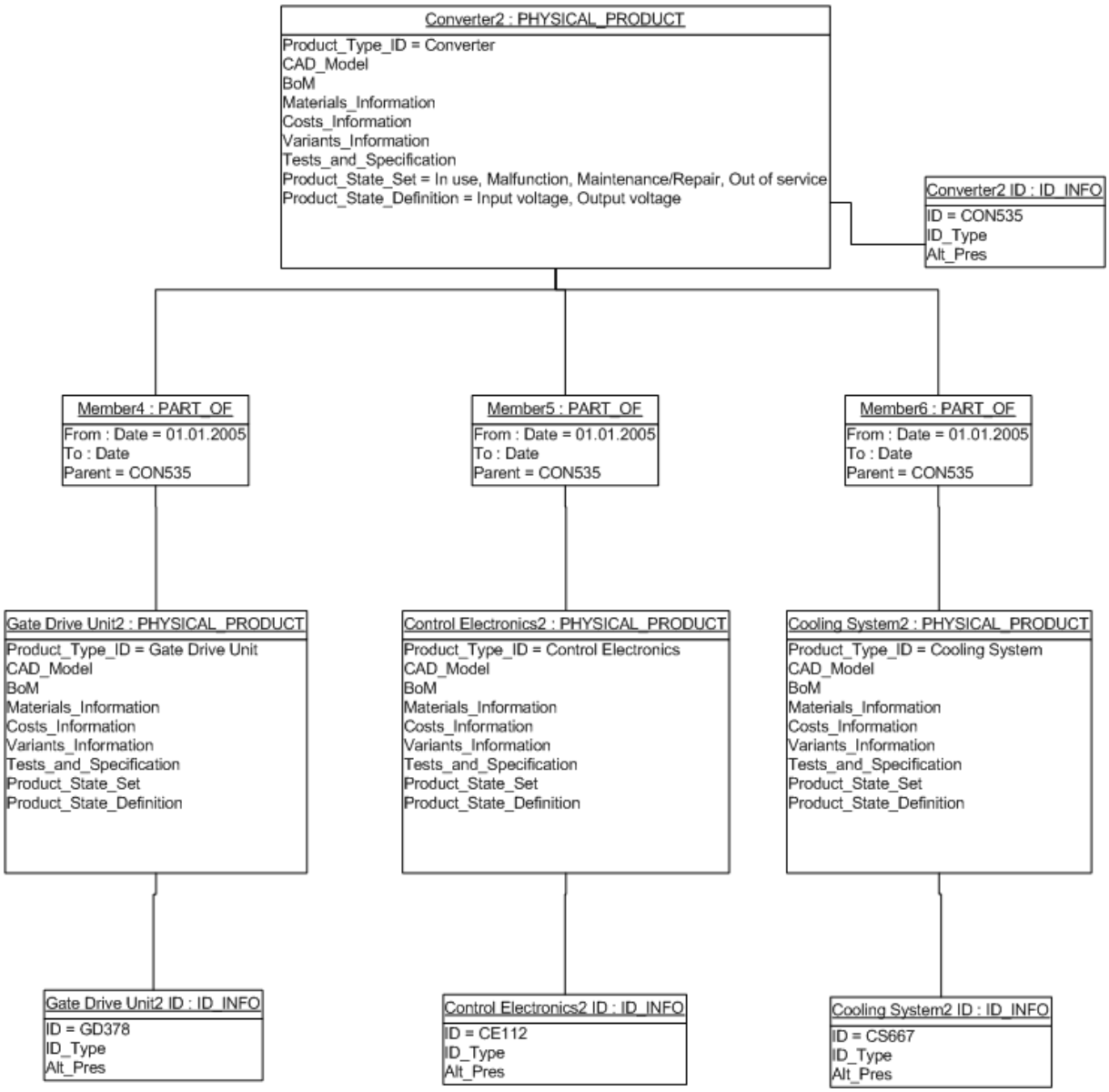

Figure 36: As-used structure for a second converter

6.1.3 Instantiation

The example in figure 37 shows the relationship between the as-designed structure of a wheel and its two instantiations. There are also shown examples of the use of the classes PROPERTY and CONDITION.

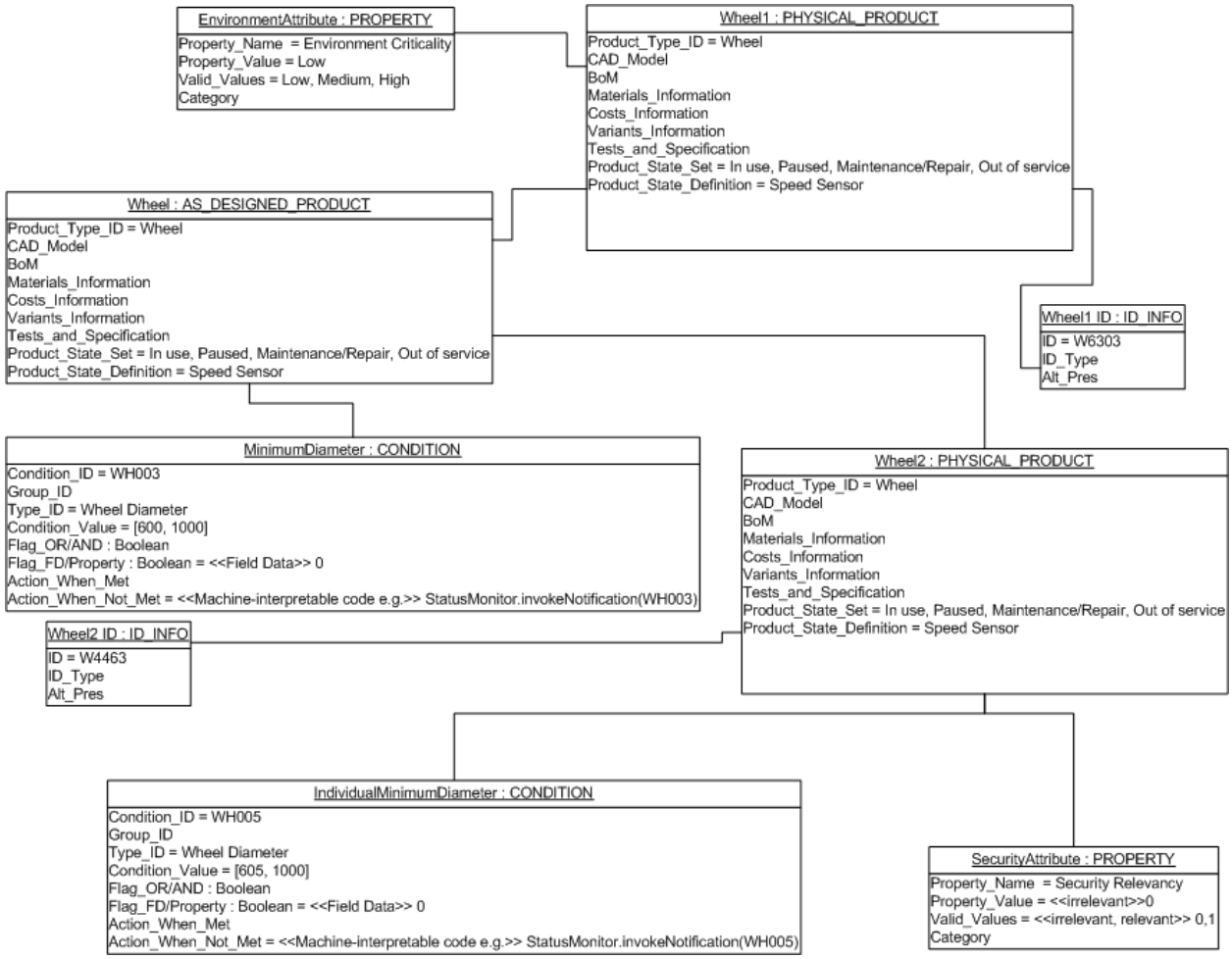

Figure 37: Two instantiations for a wheel

6.1.4 Field Data

The simplified object model[2] in figure 38 illustrates possibilities of attaching field data from different life cycle phases to a product instance.

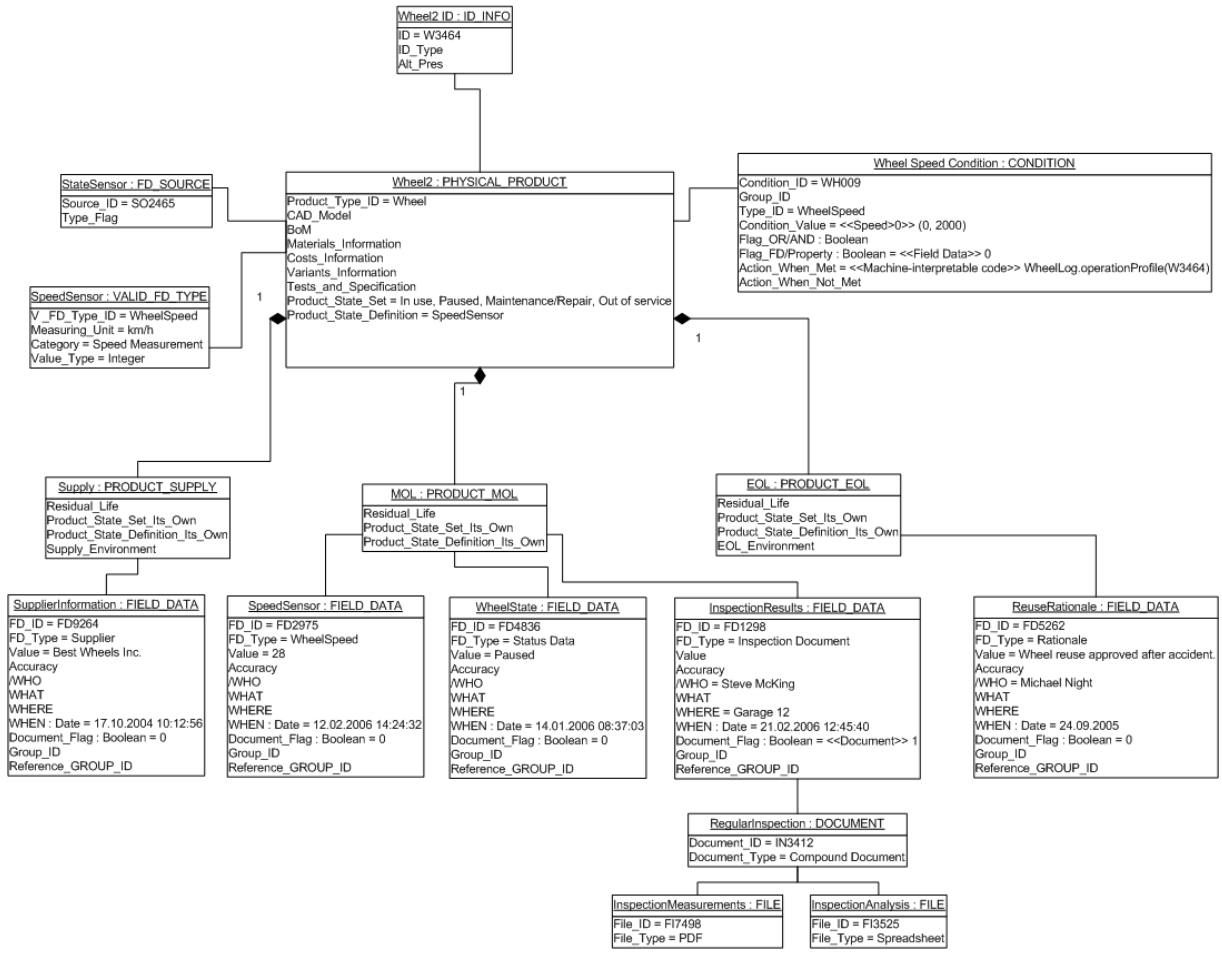

Figure 38: Field data attached to a wheel instance

[2] Due to readability, not all VALID_FD_TYPE instances are shown.

6.1.5 Maintenance Event

The example in figure 39 demonstrates a possible maintenance event, which consists of re-profiling a wheel.

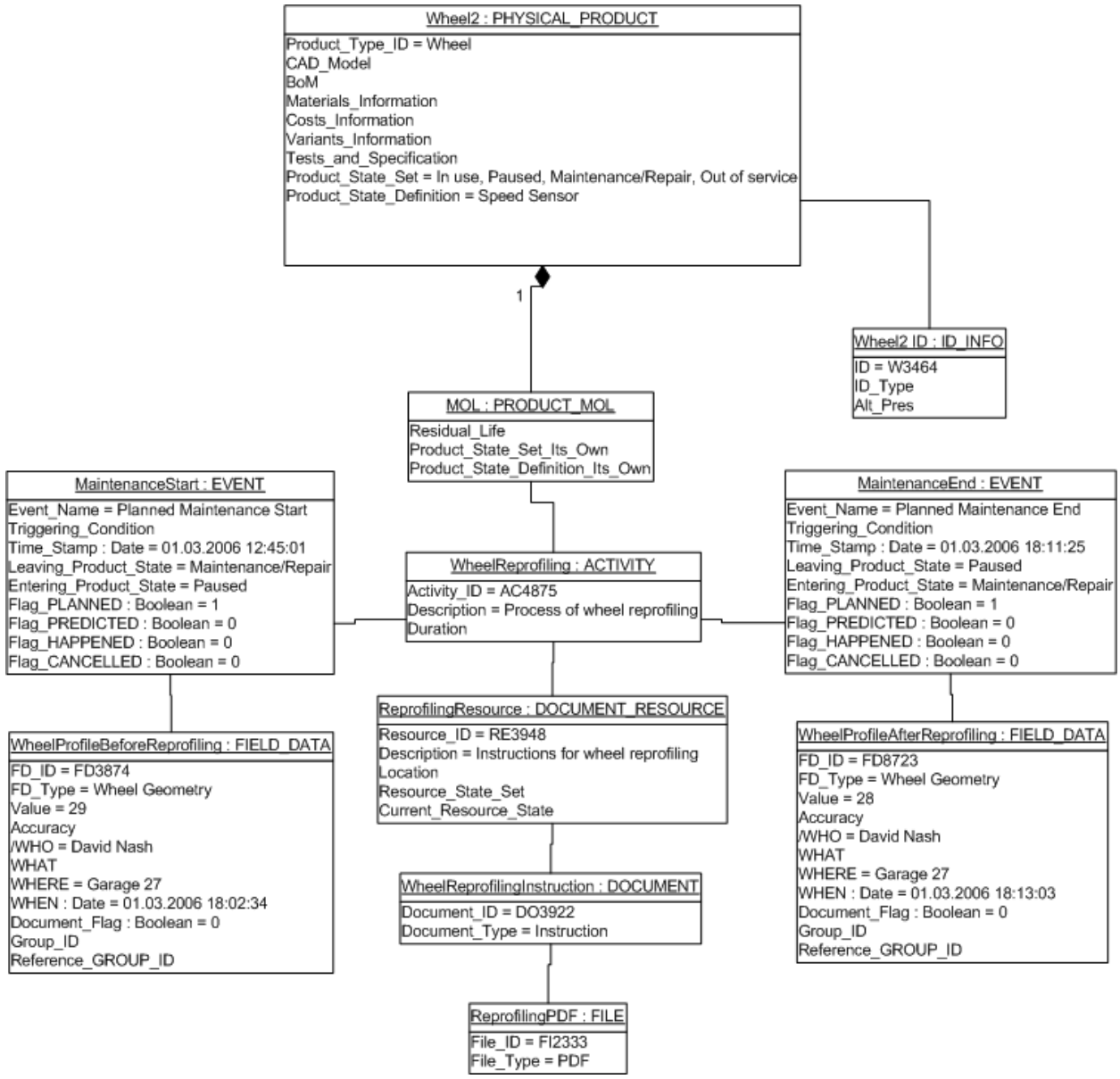

Figure 39: Re-profiling of a wheel as maintenance event

6.1.6 Storage of Product Knowledge

Figure 40 shows examples on how knowledge is represented in the PDKM system.

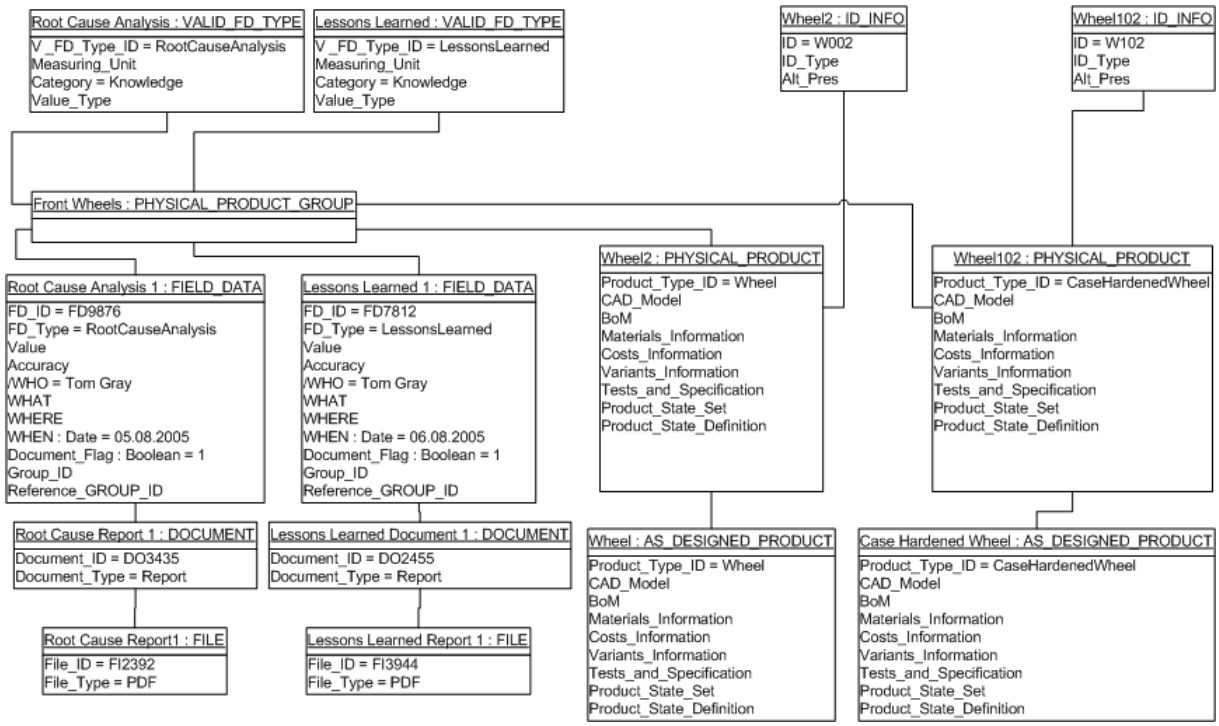

Figure 40: Knowledge associated to wheel design

6.1.7 Concluding remarks for section 6.1

More aspects of the application scenarios have been modelled in an informal way using the provided semantic model. In conclusion, it can be said that the model is capable of covering all aspects as required by the application scenario. The focus was laid on giving in form of a static snapshot an overview on how the model's semantic entities are used to cover the semantic entities of the application scenarios.

In the following sections the focus is more on selected aspects but going into more detail about the usage of the applied entities in order to foster the understanding of "how the entities work" in the respective scenario.

6.2 MOL scenario: the MTS case from WP A7

One focus of the A7 application scenario and of the related demonstrator was according to [5] to assess the predictive maintenance capability of the PROMISE DSS. The reason for this is that a major interest of MTS was to improve the after sales service, in particular maintenance and repairing actions, by being able to predict in advance, which failure is going to happen, and after which time. The chosen product of reference was a boiler, namely the ACO COMBI 27 KW. The after sales service should be able to schedule in advance a visit to a customer in order to replace a component or to perform maintenance on the gas boiler before it goes into lockout state. The technician will also be able to reach the customer with the right spare part needed, solving the real problem in the first visit, without any future need of e.g. revisiting the customer and change other parts (the ones defective since the beginning) just because at the first visit the real problem was not detected and the "wrong" part was replaced. In other words, again as stated in [5], the main scope of the A7 demonstrator was:

- Improving the efficiency of after sales service operations

- Reducing the amount of spare parts replaced
- Increasing the availability of the boiler for the end-user
- Reducing the cost for the service contract paid by the user
- Reducing the environmental pollution due to non-optimal working conditions for a long time and by waste of material generated by "wrong" parts replaced without actual motivation

Since in such a scenario predictive maintenance plays a central role, the following subsection is devoted to the representation of some important objects of the semantic model, providing the overall PDKM system with the needed information to enable the needed closure of information loops. An example is given, whilst the aim is mainly to show the modelling capability of the classes defined above rather than to give a complete and exhaustive picture of the A7 MOL scenario.
It should be noted that the A7 demonstrator was stopped since MTS withdraw from the PROMISE project. Nevertheless, it was kept as the MOL example since the modelling capabilities are demonstrated nonetheless.

6.2.1 Predictive Maintenance Management in the MTS scenario: an example

In figure 41, the way, in which the semantic model defined in chapter 4 can be used to model the predictive maintenance actions, is reported. The event of interest is the breakdown of the boiler's water pump: MTS aims, by using the appropriate algorithms developed in RC-4 activities, at predicting this event with a certain time in advance. This concept is represented with an object of the class EVENT called Pump_BREAKDOWN_predicted. Another important object of the diagram is the Maintenance object of the ACTIVITY class. This object can be of two different types: the PM_Action_PUMP, which defines all, that is necessary to know about how to perform *predictive* maintenance on the pump of the boiler, and CM_Action_PUMP, which on the contrary defines all that is interesting about the *corrective* maintenance actions on the same component. Then two instances of the Operator object of the RESOURCE class, which are of the PERSONNEL_RESOURCE type, are shown, as well as a property connected to the Operator object, called Personal_Schedule, representing the schedule of maintenance visits to customers of each single operator working for the after sales service company.

Now the following situations can be described. The breakdown of the boiler's pump has been predicted to happen on 05/08/2007. The prediction is performed by the DSS component of the PDKM system and can be read on the diagram as the value of the Time_Stamp attribute of the object Pump_BREAKDOWN_predicted. A closer look at this object says that the breakdown causes the boiler to shift from the UP state to its DOWN state. In addition, the collections of flags say that the event has been predicted but has not happened yet. Then, MTS is interested in managing the predictive maintenance action on that specific boiler, and in particular has to organize the visit to the customer. Let's say that the after sales service company can count on two different operators: Operator #231 and Operator #232. From the diagram, one can see that they are objects of the PERSONNEL_RESOURCE class, and thus also of the RESOURCE class. From their attributes, one can also note that their state can be of two types: AVAILABLE and NOTAVAILABLE. Anyway, at the current time only one of the two operators is actually available (namely Operator #232). Since the breakdown event has not occurred yet, but is predicted to happen in the future, MTS can plan the maintenance activity with time in advance. Thus, the availability of its two operators in the future is checked and, together with the other predicted events, a schedule for each operator is organized and eventually updated.

In the case that at a certain point in time the pump of another boiler breaks down before the maintenance activity is carried out, an object of the EVENT class called

Pump_BREAKDOWN_happened is created, and its Time_Stamp attribute is set to the current date and time. The Leaving_Product_State and the Entering_Product_State are the same as in the Pump_BREAKDOWN_predicted object, but in this case, the flags show that this event was not predicted but that it has actually happened. Therefore, there is a sudden need to visit the customer and perform a corrective kind of maintenance (which corresponds to the today's approach in after sales services). Now, the information on the availability of the two operators turn out to be very useful and, after a quick check of the operator's capabilities (who should be able to perform the needed action), the maintenance action can be performed.

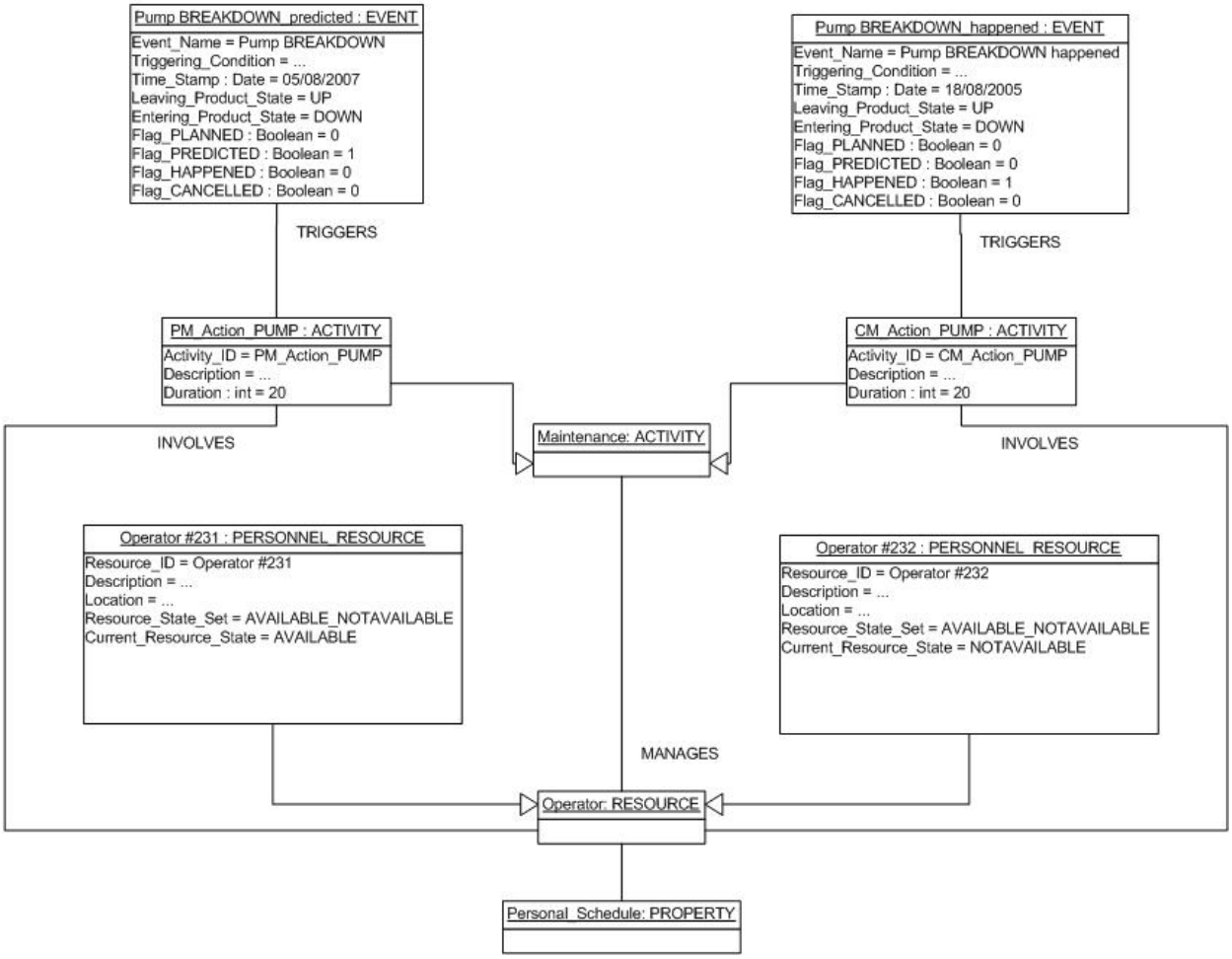

Figure 41: A7 MOL scenario – events, activities, and resources

This was only a brief description of the modelling capabilities of the semantic object model. One should also check if the classes and attributes are capable of defining what happens after the maintenance action has been performed, e.g. what attributes of the different classes have to be updated, what objects may be cancelled, and finally, what objects have to be created. It can be stated here that also for these other purposes the performance of the semantic model was satisfactory.

6.3 EOL scenario: the CRF case from WP A1

The overall scope of the A1 application scenario and of the related demonstrator (see [5]) is to reduce costs during deregistration and dismantling of ELV (End of Life Vehicles), in particular identifying the components of the vehicle, which are worthwhile being re-used or re-manufactured. The motivation to this scope derives directly from the EU Directive on ELV of

September 18th, 2000. Besides that, a relevant business opportunity is expected to come out of this.
Here, the product of interest is a passenger vehicle, whose BOL as-designed structure is represented in figure 42. The main object in this diagram is represented by the product as a whole, i.e. the Passenger Vehicle "V1" object of the AS_DESIGNED_PRODUCT class. This is the root of the tree representing the BOL as-designed product structure, and corresponds to the father node of all of the other objects in the diagram, representing the five most important car components/subassemblies of interest in the A1 scenario, namely the clutch, the starter, the battery, the air conditioning compressor, and the on board computer. The associations state, that Passenger Vehicle "V1" object is the one and only parent of all of them. All of these components/subassemblies are interesting for the A1 scenario both because the EU Directive forces to manage the end of life of some of them (e.g. the battery) and because in all the cases a sensible cost reduction is expected from the EOL operations management.

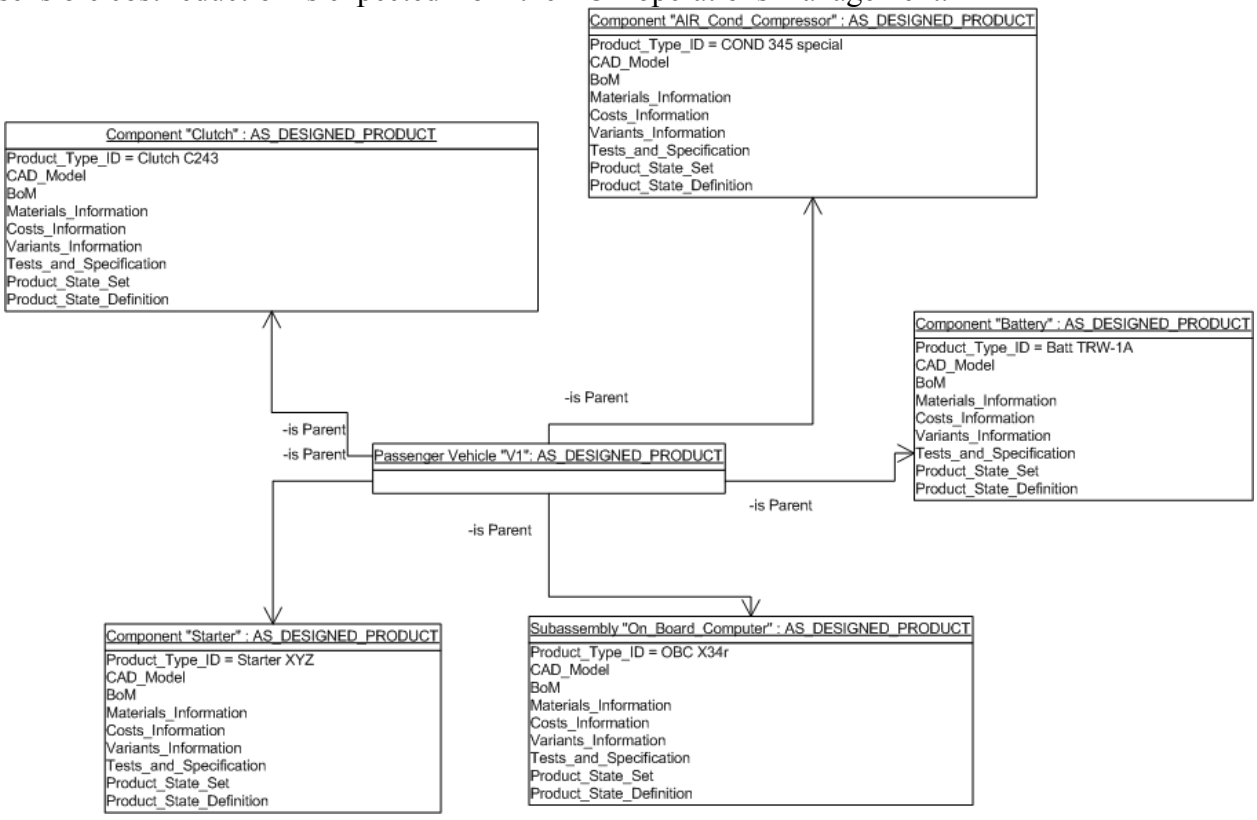

Figure 42: BOL structure of the passenger vehicle considered in the A1 application scenario

6.3.1 ELV Management in the CRF scenario: the decision strategy and its implementation

The contribution of the PROMISE DSS in the scenario reported above is very important in order to evaluate the best solution for the end of life of the identified components/subassemblies. The decision strategy in the A1 application scenario is reported in figure 43. For each component/subassembly, a series of tests has to be carried out. First, one should check if the end of life of the component must be managed because of the currently active laws or not, such as in the case of the car's battery. If yes, then the component must be surely removed from the car and, with the aid of the DSS, a statistical analysis of BOL and MOL data/information recorded for that specific component is carried out, and the decision of reusing/recycling or remanufacturing the component is made. If, on the contrary, the dismantler is not forced to remove the component, then the opportunity for the removal process to be cost-effective must be evaluated. Therefore, a series of tests is performed, namely it is first checked if the component satisfies a minimum quality level requested for its removal to be worthwhile, then, if the removal cost is lower than the

most probable sales price, and finally, if the constraint on the minimum stock level required for that component is also satisfied. Whenever one of these tests is not passed by the component, then the component is not removed from the car and is destined to go directly to the shredder with the rest of not reusable/not recyclable/not re-manufacturable parts of the car. If all of these tests are passed by the component, then the component is first removed, and then, again with the help of the DSS, the best end of life solution for it is derived.

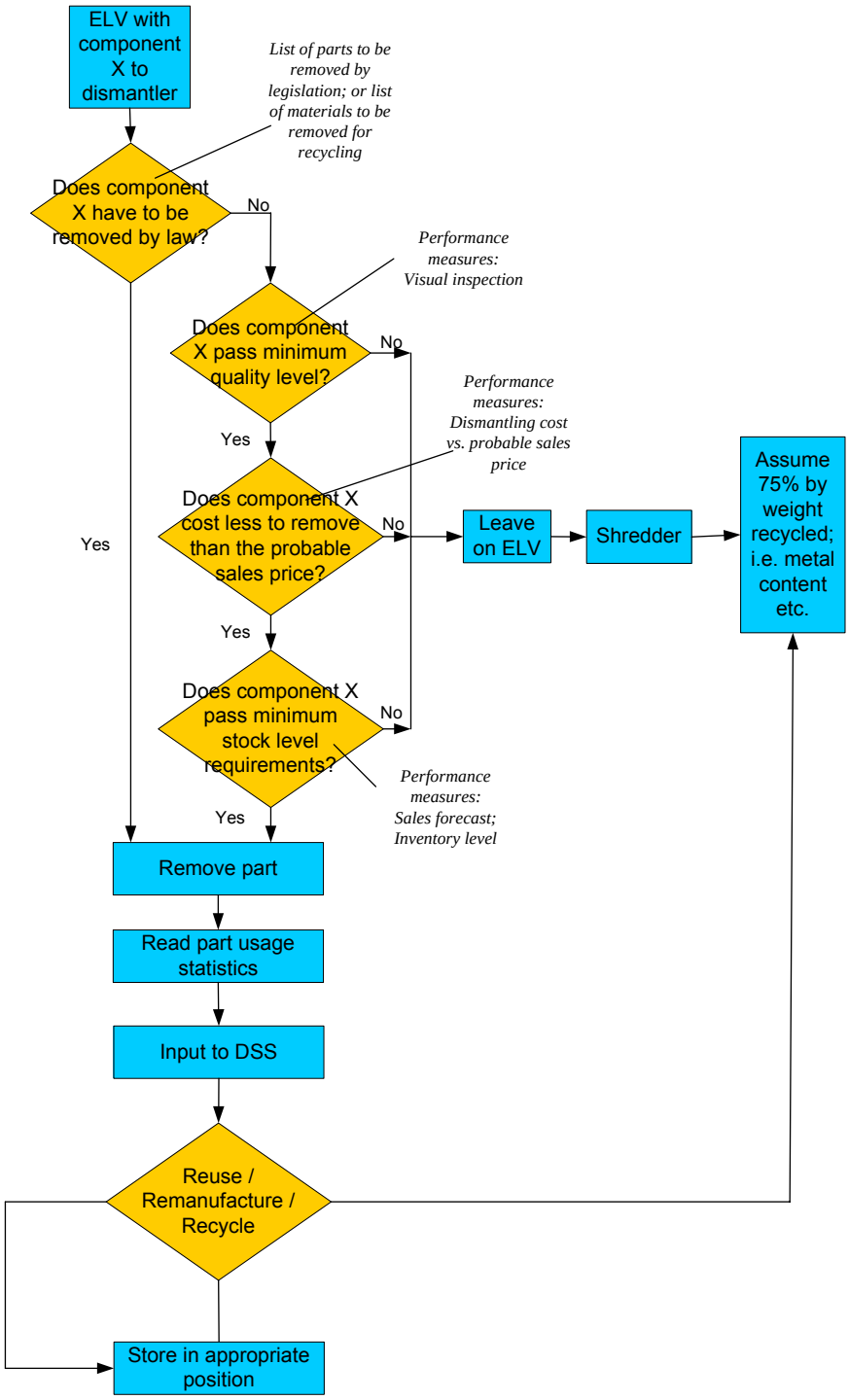

Figure 43: decision strategy of A1 application scenario

In figure 44, some examples of properties to be defined in order to manage the end of life of the car's components, as requested by the strategy reported above, are reported. In the figure, one can see the object describing a certain clutch, namely the clutch number 0132434765 (see object Component "Clutch 0132434765": PHYSICAL_PRODUCT), which is an instantiation of the clutch "as-designed" (see object Component "Clutch": AS_DESIGNED_PRODUCT). The clutch item inherits all of the attributes and properties eventually defined for the "as-designed" product. In the figure, only some properties directly instantiated for the clutch number 013234765 are reported as well as their attributes. The property called Flag_By_Law says if the end of life of the component must be managed according to some law/rule derived by the EU Directive. In this case, the only two possible values are YES and NO and for the clutch number 013234765 the attribute Property_Value is set to YES, thus, stating that the end of life of the clutch must be managed. Then, the Sales_Price property tells that the most probable sales price for the clutch is € 100, and that sales prices above € 50 can be entered. The Dismantling_Cost property says that the dismantling cost for a clutch, calculated as (Labour_Cost + Equipment_Cost) x (Dismantling_Time_of_the_clutch), is € 90 and that acceptable values range from zero up to € 180. Finally, the Inventory_Level property gives the value of the inventory level constraint for the clutch, which is in this example 450 pieces with an acceptable range of zero up to 600 pieces. All of these classes give a first idea of the pieces of information needed, at a first stage of development, to face the end of life problem of the A1 scenario.

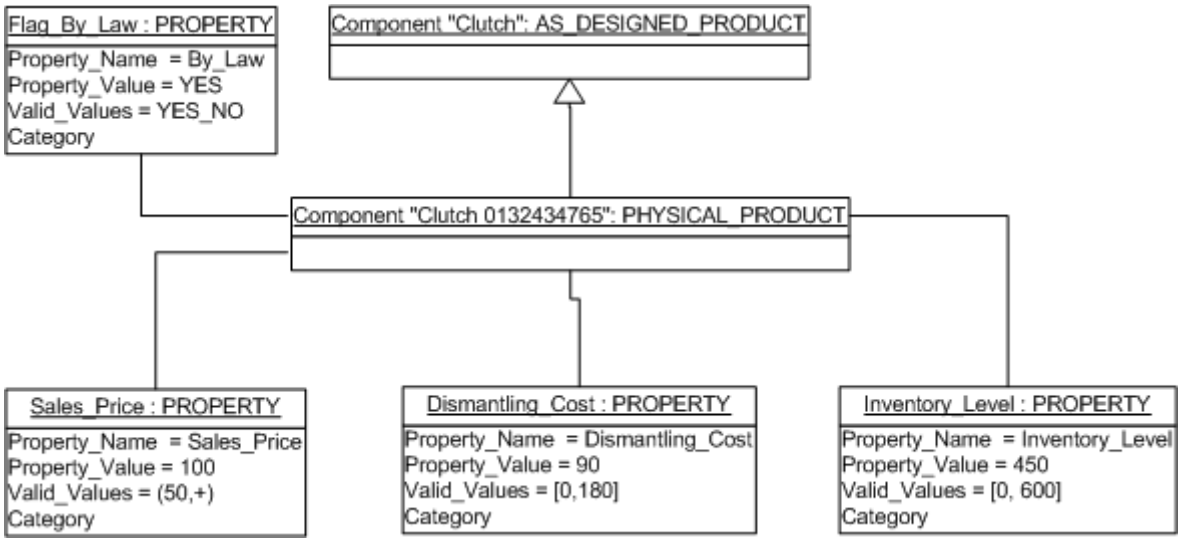

Figure 44: Properties of the clutch component – some examples

In figure 45, an object of the CONDITION class is reported in order to give an example of how the condition class can be used in the PDKM system to check if an important condition is satisfied before doing something with the object in focus of the condition. In this case, an elementary condition for the component "Clutch" is shown. In particular, the condition on the minimum stock level requirement is checked, i.e. if the current stock level is below the value decided previously as the threshold value for the component "Clutch". If the condition is satisfied, the component, following what was written above concerning the decision strategy of the A1 scenario, has to be removed from the car; otherwise, it has to be thrown to the shredder. These two different "exit points" are shown by the values of the *Action_When_Met* and *Action_When_Not_Met* attributes.

Again, this was only an example of how the semantic model can be used to model one aspect of the A1 scenario. Some other aspects were considered among those cited more or less explicitly in [5], e.g. the capability of the semantic model of providing the data basis needed by the DSS to establish the statistics needed to take all the necessary EOL decisions. The modelling capability of the Semantic Object Model was considered satisfactory also in these cases.

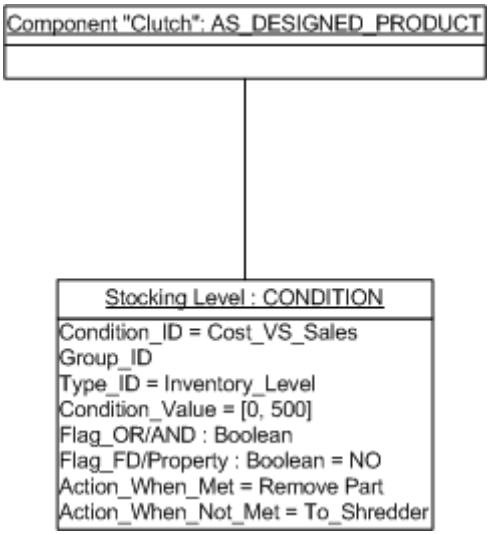

Figure 45: Conditions on the clutch component – a simple example

6.4 Concluding remarks for section 6

In conclusion of section 6, one more statement should be made. The other application scenarios as well have been used for verifying the applicability of the developed semantic model, in a more informal way, however.

7 Concluding remarks

This document presents an object model describing the attributes of the required objects of the PDKM system, as well as their relations in detail. The object model primarily addresses semantic issues concerning the comprehensible representation of domain knowledge for communication between users and applications, but also discusses some technical issues, such as the efficient storage of data in physical databases, to be considered when bringing the conceptual model presented here to a real and implemented technical data schema. A major emphasis was given to semantic issues, modelling "*the entities which the business users are familiar with and can easily communicate about. Entities (…) (are) further described by attributes and relationships to other entities and business terms used in this context. (…) (Such) a semantic model offers a common basis for data communication and exchange between applications, between users, as well as between users and applications"* [2].

DR9.6a is explicitly focused on the pieces of information that the application scenarios are interested in. A certain level of abstraction was required in the modelling process, in order to meet all their requirements.

Referring to the PDKM components of the PROMISE PDKM architecture, the presented semantic data model describes the core of the Data Management layer, whose main task is to provide a global semantic view on product and product life cycle data for all analysis applications. PDKM components not covered by the semantic model like the System Management tower and the Metadata Management tower are treated in the development of the technical data model in tasks TR9.5, TR9.11, and TR9.13.

Moreover this deliverable does not intend to provide the data model that is required by the DSS e.g. to store algorithms even if the DSS is, according to DR9.1, one of the components of the PDKM. That part of the data model is developed by WP R8.

References

PROMISE deliverables and documents (all of them are available in the PROMISE eRoom at the website https://project.sintef.no/eRoom/indman/PROMISE)

1. PROMISE DoW (Description of Work) document (versions 4 and 5)
2. DR9.1 - Design of PROMISE Information Management System (PDKM)
3. DR2.1 - PROMISE generic models (version 1)
4. DR7.3 - Selection of Tools and an existing PDM System to support PROMISE specific Knowledge and Information Management Processes
5. DA1.x – DA11.x - Deliverables describing the demonstrators of the eleven application scenarios
6. DI1.2 - PROMISE Standardization Domains
7. DR9.2 - Specification of the System Object Model
8. DR9.6b - Specification of System Functions (revised)

Industrial Standards (websites of major interest; also some of the figures have been taken from these sources)

9. STEP
    a. http://www.iso.org
    b. http://www.tc184-sc4.org
10. STEP NC
    a. http://www.step-nc.org
11. PLCS
    a. http://www.plcs.org
    b. http://www.plcsinc.org/whitepapers/03pdt-eu.pdf
    c. http://stepmod.sourceforge.net
12. PLM XML
    a. http://www.eds.com
13. MANDATE
    a. http://www.tc184-sc4.org
14. ISA-95
    a. http://www.isa.org
    b. http://www.wbf.org

Papers

15. Sergio Terzi, Jacopo Cassina, Hervé Panetto; “Development of a meta-model to foster interoperability along the product life cycle traceability”; INTEROP 2004
16. Jacopo Cassina, Sergio Terzi, Hervé Panetto, Marco Taisch; “Development of a holonic meta-model for life cycle support and product extension”; ICE 2005
17. Jacopo Cassina, Marco Taisch, Sergio Terzi; “Towards an Intelligent Extended Product”; IFIP2005

UML

18. Grady Booch, Jim Rumbaugh, Ivar Jacobson; “UML User Guide”; Addison-Wesley, 1999
19. James Rumbaugh, Ivar Jacobson, Grady Booch; “The Unified Modelling Language Reference Manual”; Addison-Wesley, 1999
20. Martin Fowler; “UML distilled”; Addison-Wesley, 2004
21. http://www.uml.org

Appendix: UML notation

In this appendix, a short guide to the UML notation used above is provided in order to help the reader not used to systems modelling better to understand the previously described diagrams.

UML (Unified Modelling Language) is a "de-facto" standard providing a family of graphical notations, all basing on a single meta-model, able to support the description and design of software projects, in particular those following the object oriented paradigm. The standard is "relatively open" and it is controlled by the OMG (Object Management Group), a consortium of companies, which was formed in order to support interoperability between systems, in particular the object oriented ones (another known OMG standard is CORBA - Common Object Request Broker Architecture).

UML was born after the fusion of many graphical languages for the description of software systems, developed from the late eighties to the mid-nineties, and [20] definitively cancelled the "Babel of languages" formerly under use.

UML can be used in many different ways, namely to build a *sketch* of a project (both for the forward and the backward engineering), a *detailed project,* or finally, as a real *programming language*. Throughout this document the standard is used to build a sketch of the overall PROMISE Information Management System (PDKM), i.e. a representation of a selected portion of the system directly related to the modelling of product data across the product life cycle (refer to the first sections for a brief outline of the intentions of the model presented here). This appendix is not meant to provide a detailed description of the differences among these three ways of using the language. For the interested reader, the reference is again [20].

Another way of differentiating the usage of the UML standard is to distinguish between the *conceptual perspective* and the *software perspective* (see [20]). In the former, UML is used to represent the concepts belonging to the domain of interest, thus moving away from the representation of "real software" elements, aiming on the contrary at the definition of a common vocabulary for a particular domain. The latter reflects the exactly opposite situation. The semantic model provided with this deliverable is a conceptual model as described in the introductory sections of this document.

Since a complete description of the UML syntax is clearly out of scope for this appendix, only the couple of diagrams really used to describe the object model are reported below.

Class Diagram

The UML class diagram represents the most used graphical notation of the whole standard, mainly because it is capable of modelling most of the concepts required by the UML user. Its basic elements are well known and widely used, while some others are not. In the semantic model, elements from both sets are represented. They are briefly described below.

A class diagram describes the *objects* belonging to a system as well as the different *relationships* among them. It also shows the *features* of each class, i.e. its *properties* and *operations*, as well as the *constraints* applicable to the different links between couples of objects.

The *operations* of a class explain what a class is able to do and how it can do it. They are often thought to be in a one to one correspondence with the methods of the class, but generally, it is better to differentiate the meaning of the two terms, since in a *software perspective* they can differ one from another. Since the semantic model presented here is focused on *objects*, their *attributes*,

and the *relationships* among them (compare to the description of task TR9.2 in the DoW [1]), *operations* were not explicitly modelled.
Properties of the *objects* can be of two types: *associations* and *attributes*. Actually, they have exactly the same meaning in the *conceptual perspective*, though their related notation is very different. Here, a brief and informal explanation of what they are used for is provided. In a *software perspective* however, they are very different also in the meaning: for the interested reader, please refer to [18], [19], or [20].

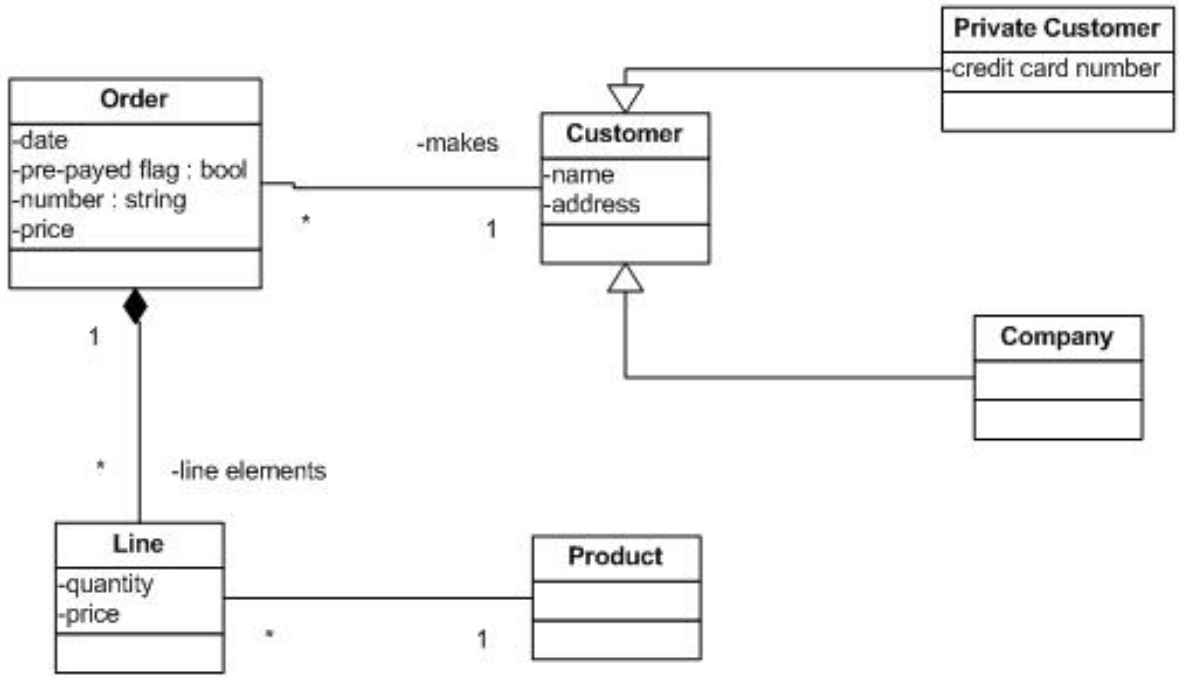

Figure 46: A typical class diagram for the management of incoming orders

Following an intuitive explanation, most of the syntax constructs of UML class diagrams used in the semantic model are outlined in figure 46. Each box represents a class, i.e. a type of objects belonging to the domain of interest. Here, the domain of interest is the management of incoming orders in a common company. Thus, the objects are represented by the Order class, the Line class (an order is composed of one or more lines), the Product class (each line refer to a specific product type), the Customer class (customers make orders), and the Private Customer and Company classes (different types of customers).
Each class can have (but this is not mandatory) one or more attributes, which represent a sort of "atomic" information embedded into the class, i.e. something typical for each *instance* of the class. This means that each instance, i.e. each object, created for that class during the existence of the system "can" have that kind of information associated to it. It is written "can" instead of "must", because the effective presence of each attribute into the description of each single object is dependent to the kind of multiplicity associated to the respective attribute. The multiplicity states the minimum and the maximum number of attributes of that type that each single object can have associated to itself. In the figure, each attribute, e.g. quantity, price, credit card number, address, etc., has one and only one value. In general, there can be from zero to one value, from zero to many values, from one to many values, etc., respectively indicated with 0..1, 0..*, 1..*, etc. For each attribute a (data) type can be specified, e.g. "string", "Boolean", "integer", etc.
Two classes can be linked via an association, represented by a line connecting the two classes. Each association says that among objects of the first class and objects of the second class a static link exists for some reason (in the *conceptual perspective*). This means that the specific objects at the two ends of the association, which are linked together, can differ from one time instant to another, but the classes are always linked together, also according to the multiplicity eventually

added to the association (refer to figure 46 again). If useful, the end of an association can be referenced with a "role" name, which explains the role of the object belonging to that class in the link with the related objects of the other class, e.g. each customer can make one or more orders.
Associations can be very general (simple lines) or more complicated if pieces of information have to be added for matters of clarity. E.g. the association between Order and Line is called *composition* and says that each single object of the Order class is composed of many objects of the Line class. This construct is capable of modelling the concepts of a whole and the parts of this whole. In this case, naturally, the whole is the order and the parts are the lines of the order. Each part can belong to one and only one whole. If this is not the case, the association is often called *aggregation* and the rhombus is painted white instead of black. Since the community of system modellers does not agree on the effective meaning of the aggregation construct, it is often preferred to leave it as a simple association. This approach was also followed in the development of the semantic model, thus, some relationships between couples of classes depicted as associations are effectively aggregations. Since this, however, is not a crucial point for a class diagram designed in a conceptual perspective, the problem is not treated further here. For the interested reader, please refer again to [20].
Finally, the links between the Customer class and both the Private Customer and the Company classes are called *generalizations* and state that a customer can be either a private customer or a company. From a conceptual viewpoint, the generalization construct say that e.g. Company is a subtype of Customer if and only if each instance of the Company class is also an instance of the Customer class, or anyway, if all that is valid for a generic object of the Customer class is also valid for each object of the Company class (which should be true, if the company is modelled as a particular type of customer).

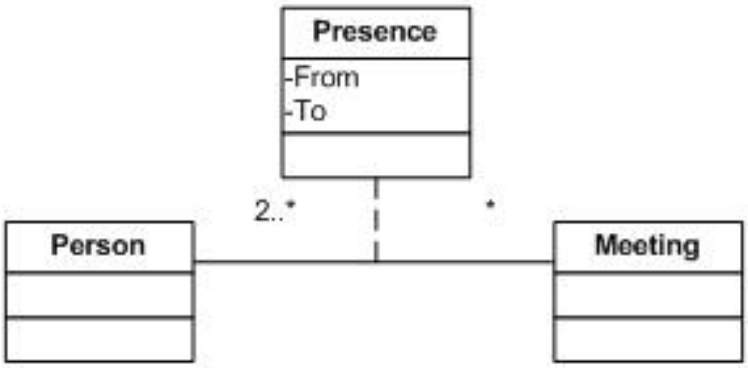

Figure 47: Use of the association class construct in a UML class diagram

Another important UML construct, which turned out to be useful for the semantic model, is the *association class* construct. This modelling element can be used to add attributes, operations, and other features to an association as indicated in figure 47. From the diagram, one can see that a person can participate to more than one meeting. Therefore, it may be useful for some reason to record the time instant when the person joins the meeting and the time instant when the person leaves the same meeting.

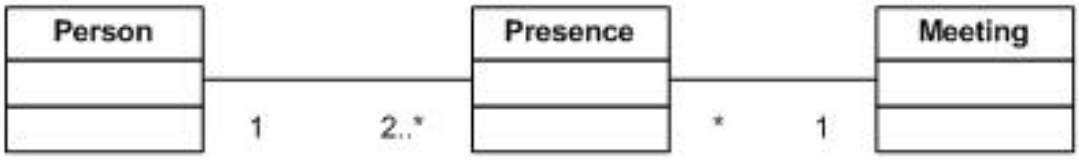

Figure 48: Alternative way to represent the information contained in figure 47

Another way to represent the same kind of information is outlined in figure 48. The two diagrams have exactly the same meaning, but the former is more compact.

Object Diagram

The UML Object Diagram represents a snapshot, at a particular point in time, of the objects composing a given system. This diagram only shows instances of classes and the existing links among them (at the instance level, not only at the class level), thus, it is often called “Instances Diagram”. To be more precise, the elements of an Object Diagram are not really instances of the related classes, but only specifications of instances of those classes. In fact, many attributes, though mandatory in principle for a certain class, can be omitted if not interesting for the purpose of the diagram.

In the semantic model, it is used to show self-explaining configurations of the objects. Thus, the focus is not on giving a complete list of all the attributes for each object represented in the diagram, but, accordingly to what is written above, on showing the attributes that are really needed for the explanation of the concept that the diagram was built for. Examples of object diagrams are given in chapter 6, where the semantic model is applied to some of the PROMISE application scenarios. The notation is self-explaining. The only major difference to class diagrams is that in an Object Diagram objects have their own name, which is underlined. Usually, the class, to which an object belongs, is indicated as well, though none of these pieces of information is mandatory.