An Ontology-Based Approach for Closed-Loop Product Lifecycle Management

THÈSE N^{O} 4823 (2010)

PRÉSENTÉE LE 15 OCTOBRE 2010

À LA FACULTÉ SCIENCES ET TECHNIQUES DE L'INGÉNIEUR
LABORATOIRE DES OUTILS INFORMATIQUES POUR LA CONCEPTION ET LA PRODUCTION
PROGRAMME DOCTORAL EN SYSTÈMES DE PRODUCTION ET ROBOTIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Aristeidis MATSOKIS

acceptée sur proposition du jury:

Prof. M.-O. Hongler, président du jury
Dr D. Kiritsis, directeur de thèse
Dr G. Hackenbroich, rapporteur
Prof. K.-D. Thoben, rapporteur
Dr M.-J. Yoo, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2010

Abstract

The main goal of the Product Lifecycle Management (PLM) is the management of all the data associated to a product during its lifecycle. Lifecycle data is being generated by events and actions (of various lifecycle agents which are humans and/or software systems) and it is distributed along the product's lifecycle phases: Beginning of Life (BOL) including design and manufacturing, Middle of Life (MOL) including usage and maintenance and End of Life (EOL) including recycling, disposal or other options. Closed-Loop PLM extends the meaning of PLM in order to close the loop of the information among the different lifecycle phases. The idea is that information of MOL could be used at the EOL stage to support deciding the most appropriate EOL option (especially to make decision for re-manufacturing and re-use) and combined with the EOL information it could be used as feedback in the BOL for improving the new generations of the product. Several PLM models have been developed utilising various technologies and methods towards providing aspects of the Closed-Loop PLM concept.

Ontologies are rapidly becoming popular in various research fields. There is a tendency both in converting existing models into ontology-based models, and in creating new ontology-based models from scratch. The aim of this dissertation is to include the advantages and features provided by the ontologies into PLM models towards achieving Closed-Loop PLM. Hence, an ontology model of a Product Data and Knowledge Management Semantic Object Model for PLM has been developed. The transformation process of the model into an ontology-based one, using Web Ontology Language-Description Logic (OWL-DL), is described in detail. The background and the motives for converting existing PLM models to ontologies are also provided. The new model facilitates several of the OWL-DL capabilities, while maintaining previously achieved characteristics. Furthermore, case studies based on various application scenarios, are presented. These case studies deal with data integration and interoperability problems, in which a significant number of reasoning capabilities is implemented, and highlight the utilisation of the developed model.

Moreover, in this work, a generic concept has been developed, tackling the time treatment in PLM models. Time is the only fundamental dimension which exists along the entire life of an artefact and it affects all artefacts and their qualities. Most commonly in PLM models, time is an attribute in parts such as “activities” and “events” or is a separate part of the model (“four dimensional models”). In this work the concept is that time should not be one part of the model, but it should be the basis of the model, and all other elements should be parts of it. Thus, we introduce the “Duration of Time concept”. According to this concept all aspects and elements of a model are parts of time. Case studies demonstrate the applicability and the advantages of the concept in comparison to existing methodologies.

Keywords: Product Lifecycle Management (PLM), Semantic web, Web Ontology Language (OWL), Interoperability, Reasoning, Mapping, Time Management

Résumé

L'objectif principal des méthodes de Gestion du Cycle de Vie d'un Produit (GCVP) est de traiter l'ensemble des processus et informations associés à un produit donné pendant son cycle de vie. Ces données sont générées par des événements et actions (eux même causés par différents agents pouvant être soit des êtres humains ou des logiciels) et leur apparition est distribuée le long des phases du cycle de vie du produit: début de la vie (conception, fabrication); milieu de vie (utilisation, entretien); fin de vie (recyclage, élimination ou autres options). La gestion d'un cycle de vie peut en outre être faite en boucle fermée. Dans ce cas, les informations disponibles sur la fabrication peuvent par exemple être prises en compte pour élaborer les processus de recyclage, de même qu'inversement les informations concernant le recyclage peuvent conduire à des choix de fabrication pour les générations futures d'un produit. Plusieurs systèmes de GCVP ont été développés, basées sur différentes technologies et méthodologies, avec pour but de d'intégrer le principe de la boucle fermée.

Le concept d'ontologie acquiert progressivement une certaine notoriété dans divers domaines de recherche. La tendance est à la conversion de modèles existants en modèles ontologiques d'une part, et la création de tels modèles à partir de zéro, d'autre part. Le travail présenté ici consiste à inclure les avantages et les possibilités offertes par les modèles ontologiques dans la gestion de modèle de GCVP en boucle fermée. Un modèle ontologique permettant de représenter un modèle de système de gestion de données et de connaissances de produit a ainsi été développé. Par ailleurs, le processus de transformation du modèle de représentation existant en modèle ontologique utilisant le langage de description Web Ontology Language-Description Logic (OWL-DL), est également détaillé. Les connaissances de fond nécessaires, ainsi que les motivations poussant à entreprendre une telle transformation, sont aussi fournies au lecteur dans ce travail. Le nouveau modèle obtenu utilise plusieurs des capacités du langage OWL-DL, tout en conservant les caractéristiques précédemment réalisées. Des études de cas se basant sur différent scenarios d'application et traitant de problème d'intégration de données et de problèmes

interopérabilité, dans lesquels un nombre importants de capacités de raisonnement sont mises en œuvre, sont présentées, afin de démontrer l'utilité du méta-modèle développé.

En outre, dans ce travail, un concept générique de traitement du temps dans les modèles GCVP a été mis au point. Le temps est la seule dimension fondamentale qui existe tout au long du cycle de vie d'un produit, et il affecte tous les produits et leurs qualités. De manière générale, dans les modèles de GCVP, le temps est un attribut lié à des«activités» et « événements » ou une partie distincte du modèle (modèles « quadridimensionnels »). Le nouveau concept mis au point consiste à considérer le temps non comme une simple partie du modèle, mais comme la base de celui-ci, tous les autres éléments devant faire partie de ce dernier. Ainsi, nous introduisons le concept de « Durée du Temps ». Selon ce concept, tous les aspects et les éléments d'un modèle sont des parties du temps. Deux études de cas, dont l'une mentionnée précédemment, montrent l'applicabilité et les avantages de ce concept par rapport aux méthodes existantes.

Mots-clés: Product Lifecycle Management (PLM), le Web sémantique, Web Ontology Language (OWL), l'interopérabilité, de raisonnement, de cartographie, Durée du Temps

Acknowledgments

I would like to express my greatest appreciation for Dr. Dimitris Kiritsis for accepting me as his PhD student and for guiding me during my four year thesis. It has been a pleasure working together and achieving a very good level of collaboration and mutual understanding. My special thanks go also to Prof. Paul Xirouchakis for accepting me as a PhD student in his lab.

I would also like to express my special thanks to the two British of our lab: Dr. Ian Stroud for all the time he devoted in advising me in various matters and for accepting me being his assistant in E-gpr course; our marvellous secretary Carol for her valuable administrative support and humour, as well as for her great support for our lab's special social events.

Furthermore, I would like to thank my colleagues: Nenad for the long discussions we had on scientific and social matters; Ali for the long discussions and activities we had in my early years in the lab; Sandeep and Saurabh who shared with me their company and friendship during my stay in EPFL; and all the other newer and older colleagues for all the time we spent together.

My Friends Nirav, Ben, Torsten, Leonidas, Carla, Gabi, Giannis and so many others for organising a number of dinners, trips, hikings, ski weekends and cycling tours together, while trying to make our life in Lausanne more pleasant.

My Parents Nikos and Roni who struggled to provide me with the best education, who are missing me so much, and they are supporting me from home for this very long period. Moreover, I would like to thank my Parents for their patience with me, for listening to my complaints about my life abroad and for always doing their best to help me whenever required…

My Brother Petros who has visited me several times and who has missed me so much from his life. Also, I would like to thank my Brother for the long, interesting and useful discussions we always have when we meet about the least and the most important things of life…

Finally, I would like to thank Yadira for having very big patience with my ups and downs during the writing of this dissertation and despite all this, still accepted to continue her life with me…

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ALM	Asset Lifecycle Management
BOL	Beginning Of Life
BOM	Bill Of Material
DL(s)	Description Logic(s)
EOL	End Of Life
GCI	General Concept Inclusion
IT	Information Technology
MOL	Middle Of Life
OEM	Original Equipment Manufacturer
OWA	Open World Assumption
OWL	Web Ontology Language
PDKM	Product Data and Knowledge Management
PLM	Product Lifecycle Management
RDF	Resource Description Framework
SOM	Semantic Object Model
SQWRL	Semantic Query-enhanced Web Rule Language
SWRL	Semantic Web Rule Language
UML	Unified Modeling Language
URI	Uniform Resource Identifier
XML	Extensible Markup Language

1
Introduction

The main players of this dissertation are the Product Lifecycle Management (PLM), its extension the Closed-Loop PLM, the Semantic Web methods and tools, as well as original ideas. This work describes an attempt of implementing semantic web methods and tools on PLM models. The definition of PLM is quite vague. Let's first define one by one the words of this phrase. Product could be something tangible (i.e. car, food, etc.) or intangible (i.e. software, algorithm, etc.) [1]. It could be defined as something which could be consumed and used by a customer, which could be sold and bought, which provides entertainment, which provides functionality or service, which could be maintained or a combination of the above, etc. Product in our case is something tangible or intangible and we focus on the functions that it provides. Lifecycle is the cycle of the life of the product. This starts when the idea to create a product appears, then, it passes from several phases (design, realisation, possible multiple usage phases, etc.) and it ends up on the disposal field as it is described by Stark *"from cradle to grave"* [2]. The exact definition of lifecycle varies depending if one is the manufacturer, who sees the big picture of the lifecycle, or the user, who sees the product mainly at its usage phase. In this dissertation with the term lifecycle we mainly mean the full picture from cradle to grave. Management is the method, theory or pattern to be followed during the lifecycle of the product in order to arrange important elements in a certain order (i.e. locate data in specific place of the information model) with the aim of achieving the desired performance and results. PLM is the combination of all the above meaning a system which manages the data and information generated from the product, during the product's lifecycle.

The main goal of the PLM is the management of all the business processes and of all the associated lifecycle data. Lifecycle data is being generated by events and actions (of

various lifecycle agents which are humans and/or software systems) and it is distributed along the product's lifecycle phases: Beginning of Life (BOL) including design and manufacturing, Middle of Life (MOL) including usage and maintenance and End of Life (EOL) including recycling, disposal or other options [3].

Closed-Loop PLM extends the meaning of PLM in order to close the loop of the information among the different lifecycle phases. The idea is that information of MOL could be used at the EOL stage to support deciding the most appropriate EOL option (especially to make decision for re-manufacturing and re-use) and together with the EOL information it could be used as feedback in the BOL for improving the new generations of the product. The interest of the different actors for Closed-Loop PLM arises from the need for measuring and controlling the total cost of the product as well as from the growing interest for sustainable development and production, for providing better service and for managing the EOL treatment of the products.

The concept of the Closed-Loop-PLM, in practice, has several requirements in order to be realised, a very important of which is the development of an information system which supports the continuity and retrieval of the lifecycle data and information. This requires a system or systems which achieve and maintain information integration and system interoperability across the entire lifecycle of a product. Interoperability gaps among main commercial PLM systems exist and are causing problems for the products overall. Typical example of such problems in the BOL is the delay on the production of Airbus A-380. Interoperability gaps exist even when the PLM systems are made by the same vendor, but the systems are focusing on different phases of the lifecycle. Data is the lowest level of abstraction and carries no useful meaning (i.e. in a passenger vehicle a sensor measures the engine temperature and sends measurements to the system). Then, on this data a pattern (with criteria) is imposed by a human or a machine in order to transform data into information which is the next level of abstraction (i.e. make the graph of temperature along time containing thresholds for temperature). Thus, data is interpreted and takes a meaning. Finally, information is processed more and it is transformed into the highest level of abstraction, into knowledge (i.e. how to deal with high temperature of the engine: stop engine to cool down, check the cooling system etc.).

An important element originating from Information Technology (IT) and might provide major or minor solutions is the use of ontologies combined with the use of the related IT methods and tools. Ontologies are rapidly becoming popular in academia. There is a tendency for both converting existing models into ontologies and creating new ontology models. Ontology models support several useful features, main of which are: to share common understanding of the structure of information among human or/and software agents; to enable re-use of domain knowledge; to make domain assumptions explicit; to separate domain knowledge from the operational knowledge; to provide formal analysis of terms; and based on them, analyse the domain knowledge [4]. Formal analysis of terms is extremely valuable when attempting both to re-use and to extend ontologies [5].

In this work we have developed both an ontology information model, to represent data and information, and a concept for utilising time as the universal reference-basis of the information systems. The developed time concept claims to be a lean method for providing the systems with a first level of data integration through synchronisation. The concept by-passes the burden of using different semantics in different models and therefore, models implementing the concept are able to be synchronised, although they might be having different semantics.

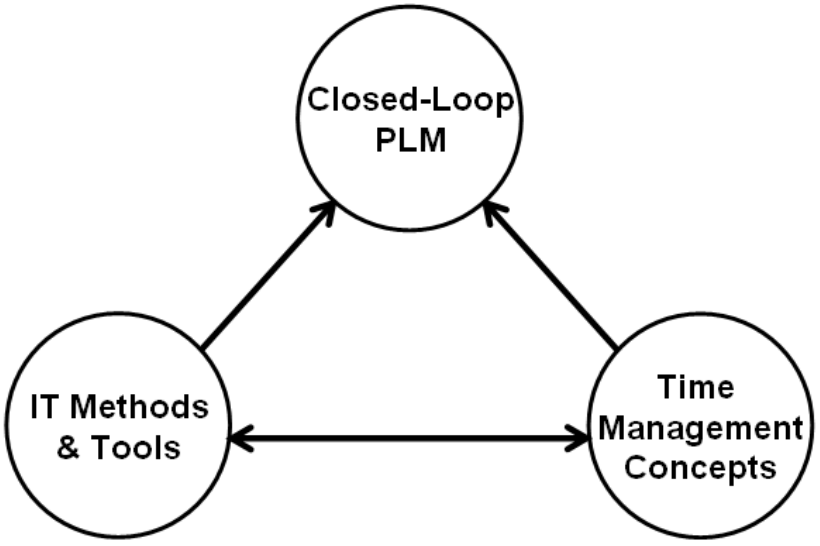

Figure 1: Main players of the dissertation.

This work aims in providing new functionalities and solutions to PLM systems towards data integration and system interoperability. The combination of the different players of this dissertation towards this aim is illustrated in Figure 1. The two arrows pointing to PLM models demonstrate the implementation of the advantages of the IT methods and tools, and of the time concept in PLM models in an efficient and simple manner. The double arrow between the IT methods and tools and the time management concept demonstrates the technology exchanged between them in order to show the physibility of the concept. Furthermore, this work aims at presenting a number of benefits and opportunities created for the future PLM systems through a number of case studies.

1.1 Motivation

The motivation of this work lies on three main pillars: the prior works on the PLM, the works on new IT methods and tools as well as original ideas on managing time in PLM data and information systems. Advancements of IT methods and tools are combined with the concepts developed in this work in an attempt to deal with the gaps in the PLM coverage arising from previous literature. Ontology-based semantics prove to be efficient for developing machine-understandable models which are able to understand the meaning of the data and information they contain. Thus, PLM systems developed using semantic web methods and tools would support system interoperability and data integration as well as lead a way towards using the information contained in the systems for extracting useful knowledge. This knowledge would provide feedback for the different PLM phases i.e. for improving the design of future generations of the product.

PLM Perspective

In today's systems the Closed-Loop PLM is not yet supported in an efficient and practical manner. Although information flow is well tracked during the BOL, this is not the case for the MOL and the EOL phases. In general, the information reaching EOL (from MOL) and BOL (from MOL and EOL) is incomplete and often inappropriate and/or insufficient to support decision. This, results in preventing the feedback of the product related information, generated during the MOL, to the EOL and to the BOL. Furthermore, the EOL information, i.e. high costs to disassemble the product, is prevented from reaching the BOL i.e. for improving design of the new generation in order to aid disassembly. One reason for this

situation is the limited view of the current systems on PLM. Each one of the existing models or standards, which deal with PLM to some extent, is focusing on a different level of abstraction of the PLM. This makes them not being able to see the entire lifecycle and therefore there is the need for interoperability between the different systems of the different levels of abstraction. However, this interoperability is missing even in cases that the PLM systems are made by the same vendor, which leads to information integration and system interoperability gaps. The different levels of abstraction which may be managed by different information systems isolated to each other are shown in Figure 2. It should be noted that one may arrange the structure of the levels or add new levels of abstraction according to his requirements and expertise. In Figure 2 the different levels of abstraction are illustrated in a pyramid, meaning that each system of a level controls or manages one or more systems of the level below. Therefore, in Figure 2 we have that: each PLM system may be controlling or managing one or more Original Equipment Manufacturer (OEM); each OEM may be controlling or managing one or more Decision Making System, for example its own structure and/or that of subcontracting or sister entities, that correspond to the Organisation (or Execution) level of the system (which includes humans or equipment utilised for taking decisions and/or for executing activities, processes, etc.); each Decision Making System's action may be controlling more than one Component, and it considers all the artefacts involved in decision making. This Decision Making System level links the Organisation level with the Component/Part level; each (type of) Component is utilised in more than one products. A typical example of this structure would be the PLM system of VW Group, which manages several OEMs (VW, SEAT, Skoda, etc.) and each OEM has several suppliers, decision groups, etc. Then, on product level, one or more types of vehicles share common components like i.e. VW Polo, Skoda Fabia and Seat Ibiza.

Several of the current systems focus on managing the maintenance activities and processes, other focus on the design or the manufacturing process, other on disassembly, reverse logistics, etc. and very few are trying to cover the whole lifecycle. In these cases there is a lack of vertical information visibility [6] of the systems, from the highest to the lowest level of abstraction and vice versa. Even for systems which are on the same level of abstraction, problems appear when attempting to share the data and information of each other (i.e. manual mapping). In this case there is a lack of horizontal visibility [6] of the information

of the systems of the same level of abstraction. In fact, current approaches have left uncovered areas in the area of PLM. This problem appears to be more crucial while engineers attempt to make the systems to interoperate with other systems in the same or other levels of abstraction of the PLM or even with systems of the same level, but are developed to cover different requirements. Solutions for improving the PLM systems towards Closed-Loop by suggesting and testing alternations on the existing PLM systems are necessary.

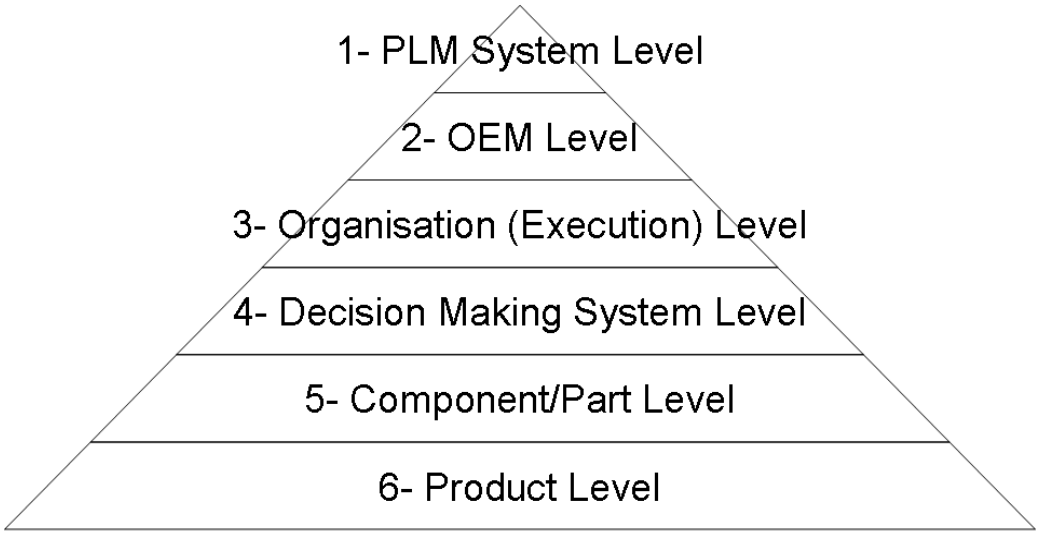

Figure 2: Levels of Abstraction.

Technological Perspective

There are several IT advancements which are not yet sufficiently implemented in the systems of the domain and could support the Closed-Loop PLM. On the product or product component level this is translated as the use of new generation of product embedded information devices [3], [7] which also have process power i.e. sensors which provide filtered data instead of raw data for the monitoring system. On the PLM system level the idea is to develop the model of the system using semantic web methods and tools. Nowadays interoperation and collaboration is an essential requirement for an increasing number of actors of extended enterprises, manufacturers and suppliers. Collaborative engineering even within the same enterprise has many barriers to overcome. Although a lot of data is being collected by various systems, there is no efficient and productive method to

map, to process and to make the data useful. Consequently, there is poor data management and several barriers for the Closed-Loop PLM appear i.e. the input data for improving future products, activities and actions is incomplete. The development of systems capable of understanding the data they contain and capable of generating knowledge out of their data is required. Ontology-based semantics ensure flexibility and a common understanding of terms for both human beings and computer systems. In this case the basis of the data and information systems is "concept composition and knowledge generation". The initial concepts are always simple (i.e. humans are mammals, mammals are animals, etc.). The composition of many simple concepts leads to complex concepts and thus, the system may compose new concepts.

According to Noy et al. [4] ontology is *"a formal explicit description of concepts in a domain of discourse (classes (sometimes called concepts)), properties of each concept describing various features and attributes of the concept (slots (sometimes called roles or properties)), and restrictions on slots (facets (sometimes called role restrictions))"* and *"An ontology together with a set of individual instances of classes constitutes a knowledge base"*. Therefore, the systems using ontology-based semantics claim to be concept-based and to be able to combine existing simple and complex concepts with data, in order to generate "new" concepts and hence new knowledge. All concepts of the system will be semantically defined. This could be performed using description logics and other types of rules in order to provide machine-understandable meaning to the concepts. Each Data loaded into the system has a meaning and hence, belongs to a concept. Relevant inference engines could be used to support reasoning on the concepts and data of the model. The importance and the usefulness of the "new" concepts would be evaluated according to the data loaded into the system. Moreover, the data will be used to validate the "new" concepts against any logical inconsistencies. Thus, the data will be fully exploited and new concepts would be validated. The validated "new" concepts are the new knowledge generated. Finally, semantic web methods and tools also allow ontology merging which is supported by the combination of the rules with the inference-engine. It should be noted that in case of ontology merging, several complementary restrictions might be necessary in the model in order to maintain consistency.

Although there are several works taking advantage of such methods and tools in other research domains, in PLM still, very little work has been performed. Implementing such advances could be beneficial towards developing systems supporting the concept of the Closed-Loop PLM.

Time Management perspective

Time has some qualities which make it special among all the attributes of the information systems: it is a universal and an objective element. Time is the only fundamental dimension which exists along the entire life of an individual (including materials and physical products) and it affects all individuals and their qualities. Individuals existed in the past and will exist in the future no matter if they only currently exist in our model. Furthermore, time is simple and comprehensive and therefore, application independent. In this way time may be used as the connecting element of various systems and models.

At the same time Closed-Loop PLM is naturally a system describing the timeline of the lifecycle of the products. Of course on this timeline many events, activities, processes, etc. of the various lifecycle phases take place. However, time is one of the very few elements that all the different parts of the systems have in common, time is objective and the parts of the systems can be described through it. In today's systems although time attributes exist in various parts of the systems, there are no systems which are based on time. Time is always considered in Asset Lifecycle Management (ALM) and PLM models either as a part of the model or as an attribute in parts of the model. It should be noted that "Asset" in our context has the meaning provided by the International Society of Engineering Asset Management (ISEAM). According to ISEAM the Engineering Asset Management focuses on "*life-cycle management of the physical assets required by a private or public firm, for the purpose of making products, and/or for providing services in a manner that satisfies various business performance rationales*" [8]. Thus, assets are types of physical products utilised to produce products or services and therefore, require different lifecycle management procedure than in PLM.

1.2 Objectives and Research Questions

In PLM systems as it is shown in paragraph 1.1 there are still several open issues which seek solutions and improvements. Firstly, our aim is to study currently used systems in PLM or in parts/phases of PLM in order to figure out gaps in the coverage of the PLM. Then, our aim is to select the most important and promising gaps according to our criteria, to figure out the requirements to address them and to develop ideas of ways of covering these requirements. Furthermore, we study the status and the capabilities of IT advancements, and develop a methodology for implementing them in PLM systems. The combination of all the above is used to develop alternations of the models of the systems, which could make the systems provide novel or extended services and solutions. Then, the solutions are validated through case studies and the ontology model is validated through ontology evaluation patterns considering how much generic and applicable it is. The research questions to be answered in this work are:

1. *How to develop ontology-based models in order to improve aspects of the Closed-Loop PLM systems?*
2. *How to use the IT methods and tools efficiently?*
3. *Which are the benefits and opportunities created for the PLM systems?*
4. *Can "time" be the sticking (glue) element of the various PLM systems?*
5. *Can "time" aid in providing vertical and horizontal integration of the systems?*
6. *Can "time" aid in providing interoperability among different systems?*
7. *How can the combination of "time" with IT methods and tools aid business applications towards Closed-Loop PLM?*

In order to address questions 1, 2 and 3 this work provides the background knowledge, the system architecture and an implementation methodology for developing an Ontology-Based approach. Furthermore, it provides solutions described in case studies, for improving the PLM systems towards Closed-Loop. It also introduces an implementation methodology of IT methods and tools, as well as of semantic web advantages in the existing PLM systems. It should be noted that in order to follow the suggested implementation method, alternations might be necessary to be performed on the existing systems. The result after

implementing the method is that the developed ontology systems have both the functionalities they had before the implementation and the new functionalities provided by the used technology.

Moreover, research questions stem from the gaps which were spotted during the study of the current systems regarding the possible capabilities which appear through the time implementation. These are questions 4, 5 and 6, and they are addressed by a developed original concept on managing time in PLM systems. The qualities of time characteristics (time is objective and exists naturally in the whole model) were the initiative to select time as the basis of our methodology for model development. Therefore, we introduce a concept, the "*Duration of Time*" concept, which utilises these unique advantages of time. Moreover, we provide a step by step method for implementing the concept in current systems and models. Time in this context is used with its generic meaning. The concept introduces the idea of seeing all aspects and elements of a model as parts of time and it provides flexibility, application independence and simplicity. In this way time exists naturally in every part of the system as it does in real life. Thus, time could be used to achieve a first level of integration among different systems.

Finally, to address question 7 a case study has been developed to test and validate the suggested concept. The key element in this case is to show how time could be used as the basis for securing the continuity of the multi-level system information along time.

1.3 Methodology

The methodology followed in this dissertation is shown in Figure 3. It consists of five logical steps describing the study of the background works and technologies, the development of new concepts and methods, the implementation and testing of the developed concepts and methods in case studies, the overall evaluation of the results, and the possible future extensions of this work.

In step 1 firstly, we studied the background works in current PLM models. The aims were: to acquire a good background and knowledge of the domain; to define which are the requirements of the domain from new methods and tools; to identify possible gaps of the current models in the domain coverage; and to define possible improvements we could

suggest towards Closed-Loop PLM. In the latter we realised that time is an under-developed and under-exploited element in current models, and it could be beneficial to introduce a method for developing models having an architecture for exploiting and using the characteristics of time (time is both an objective dimension and universal dimension). Moreover, we studied the theory, methods and tools of the IT which we considered as useful for ontology development. Thus, we obtained good knowledge of the capabilities and the functionalities of this technology. Furthermore, we studied applications of the ontology-based IT methods and tools in other research fields which have implemented them excessively. In this way we obtained a good overview of how they work; how they are implemented; and what possible opportunities they might provide for Closed-Loop PLM. Finally, we studied background works dealing with time management in various different sectors to obtain ideas for developing a method for exploiting better time in PLM models. The proposed method has to be feasible and easily implemented in current models, by using the IT advancements.

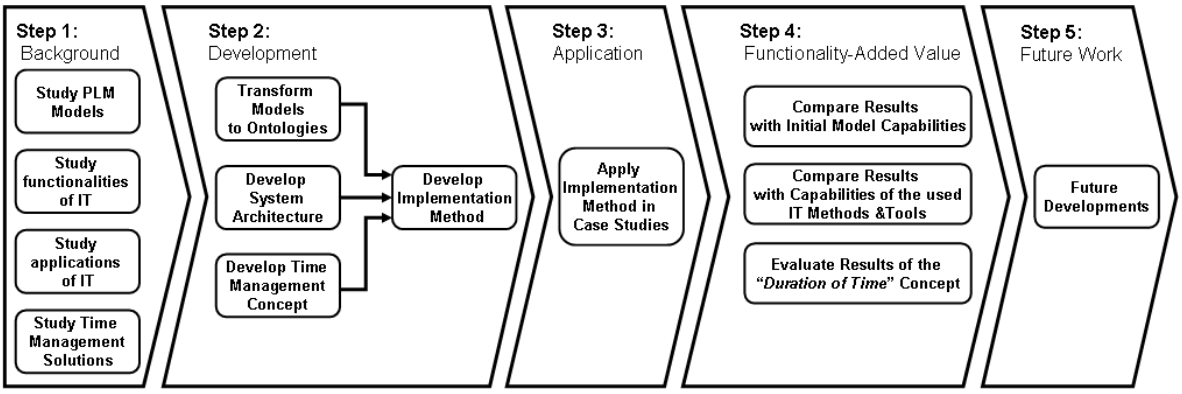

Figure 3: Methodology Process Overview.

In the second step, we used the knowledge of the first step in order: to transform current models into executable ontology models implementing a number of the IT methods and tools; to develop a system architecture for demonstrating how to exploit and apply the IT capabilities on the ontology PLM models; and to develop a concept for exploiting the time characteristics in favour of providing new capabilities for PLM systems. Then, we combined our knowledge and experience acquired in this step in order to propose a step by step implementation method of the IT advancements and of the time concept in current and future PLM models.

In the third step, we applied all the models developed, the proposed advancements and the implementation method in a number of applications, in case studies. The applications illustrate in detail the logic and the method used in practice for exploiting the potential of the proposed tools, methods and concepts. This part of the dissertation demonstrates the capabilities of the models implementing the proposed tools, methods and concepts; and the opportunities appearing towards realising the Closed-Loop PLM.

In the fourth step, we evaluate the functionality and the added value of the proposed concepts, methods and tools. Firstly, results are compared with the capabilities of the initial model with the aim of checking that all the initial functionalities and capabilities are also valid in the ontology models. Secondly, the capabilities of the ontology models are compared with the theoretical capabilities of the IT methods and tools as they are described in the background literature by the experts and the developers of these technologies. Finally, we evaluate to which extent the "*Duration of Time*" concept could be used as the basis of the PLM systems for providing advantages such as data integration.

In the fifth and final step, we define possible future extensions of this work as well as requirements for the capabilities of the next generation IT methods and tools which we believe would be useful to be provided for the PLM systems. All these are towards developing models supporting the full potential of the Closed-Loop PLM.

1.4 Contributions

The contributions of this dissertation are divided into two main parts: the development of an ontology approach for PLM with the use of the relevant methods and tools; and the introduction of the original "*Duration of Time*" concept. These parts are very well connected and related to each other, but they are presented as two parts in order to aid understanding of the benefits, functionalities and capabilities added to PLM models by each part.

Development of an ontology approach for PLM and the use of the relevant methods and tools in PLM models

This contribution provides in detail the process of how to use an ontology-based approach on PLM models in a lean manner which makes the models inherit new functionalities and

capabilities deriving from the utilised IT methods and tools. Firstly, this dissertation demonstrates how one may transform efficiently UML models into executable OWL-DL models. Also, it demonstrates a number of possible alternations which might be necessary to be performed on the models in order to make them capable of using the capabilities deriving from IT methods and tools. Moreover, through the description of the system architecture it demonstrates how to use the combined capabilities of the available tools on the developed OWL-DL models. Then, it provides a generic implementation method of the architecture which may be applied on a big number of today's models to broaden their capabilities and functionalities. Finally, in the first case study it is demonstrated how to use rules in order to obtain the benefits of the utilised IT methods and tools.

The "Duration of Time" concept

The second contribution of this work is an original concept on how to manage and use time efficiently in PLM models, the "*Duration of Time*" concept. The aim of the concept is to exploit the objectivity and universal status of time by using time as a reference-basis to integrate different models through synchronisation. It should be noted that time in this context is used with its generic meaning and it could be date, time, duration, etc. depending on the application. Moreover, it has been demonstrated how to implement the concept in existing models and how to use it. Furthermore, the second case study of this dissertation demonstrates a part of the benefits provided by the use of the concept in PLM models. Finally, in the third case study we have a demonstration of the combination of the capabilities of the "*Duration of Time*" concept and the IT methods and tools.

1.5 Thesis Structure Outline

In Chapter 2 background knowledge on semantic modelling methods, tools and ontology technologies is presented. This is a chapter which provides the reader with the basic knowledge of the technologies utilised. Its aim is to support the reader understanding the other parts of this dissertation.

In Chapter 3 the state of the art is presented. This chapter is divided in three parts: PLM, ontology applications and the time management. The aim is to demonstrate: the current status of the models used; and the possibilities appearing with the use of new ontology-

based IT methods and tools, and concepts. Moreover, it provides the basic overview of how time is managed in the current models and systems.

In Chapter 4 the developed models and methodologies are presented. This includes a detailed description of the system architecture and its functionality. Furthermore, the translation of the initial UML model into OWL-DL is described in detail as well as its extension to provide a better coverage of maintenance. Moreover, the time management concept is introduced. Finally, a methodology for implementing efficiently the utilised architecture and technologies is proposed.

In Chapter 5 the methodology, the architecture, the "*Duration of Time*" concept and the models developed in chapter 4 are implemented in three case studies. The case studies demonstrate in detail the results of the proposed implementation and the opportunities created in the domain for improving aspects of the current systems.

In Chapter 6 the evaluation of this work is presented. The evaluation is performed firstly, in checking whether the developed ontology models maintain the functionalities of the initial models; secondly, in checking to which extent the developed ontology models implement the functionalities of the IT tools and methods; and thirdly, in evaluating the results of the implementation of the "*Duration of Time*" concept.

Chapter 7 contains the conclusions of this work and the future perspectives for extending this work. The future perspectives mainly focus in the use of future IT technologies as well as in the applying the proposed tools, methods and concepts in complex and multi-system industrial environments.

2
Background in Ontology Development Technologies

In this chapter we present a brief description of the existing technologies, methods and tools for developing ontologies. The aim is to demonstrate the capabilities of the existing technology and the opportunities they would provide in PLM systems, when implemented, as well as to provide a basic knowledge for the reader to aid comprehension of the rest of this dissertation.

The aim of using IT methods and tools for developing ontologies is to represent data in both a machine-understandable and a human understandable manner. Moreover, the use of the methods and tools may also be used in order to transform data into information and then into knowledge.

Ontology models support several useful features, main of which are: to share common understanding of the structure of information among human or/and software agents, to enable re-use of domain knowledge, to make domain assumptions explicit, to separate domain knowledge from the operational knowledge, to provide formal analysis of terms and based on them, analyse the domain knowledge [4]. Formal analysis of terms is extremely valuable when attempting both to re-use and to extend ontologies [5].

2.1 Semantic Web Languages for Representing Ontologies-Data-Knowledge

Several languages have been created to represent data. The standardised and mostly used are the eXtensible Markup Language (XML) with the XML-Schema, the Resource Description Framework (RDF) with the RDF-Schema and the Web Ontology Language (OWL). The logical structure of these languages was first presented by Tim Berners Lee [9]

and since then this structure has been extended. Figure 4 shows a later version of the structure developed by Signore [10]. In this figure, the ontology layer includes OWL, the Rules include the Semantic Web Rule Language (SWRL) [11] and the Query layer includes SPARQL [12].

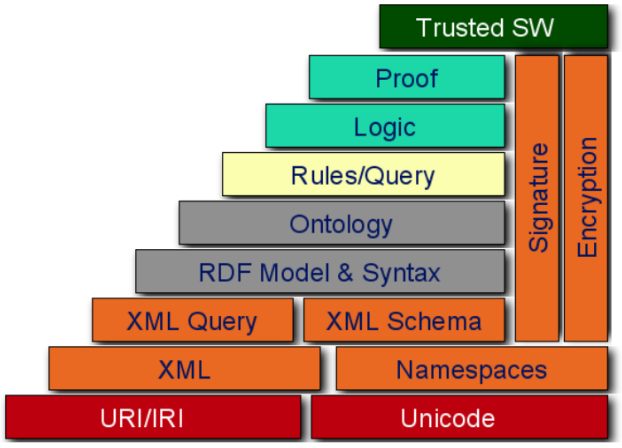

Figure 4: The Semantic Web Stack as it is described by W3C.

XML provides a basic syntax for structuring the content of documents [13] and in this way it structures the data contained in XML documents. XML as a language supports the feature that each user is capable of defining his own syntax formats (extensible), which is an advantage in comparison to older descriptions where this was not possible i.e. HTML. However, XML doesn't associate semantics with the meaning of the contained text of the documents and hence, incompatibility of syntax and, more importantly, of semantics is being created (i.e. in one format $50 might be labelled as "price" and at another format as "cost") [14]. This results into burdens while integrating data. There are several previous works which have used the XML for representing information in various applications in PLM. For example, Zeid and Gupta [15] addressed an XML-based knowledge representation model for disassembly planning. To solve this drawback of XML, XML Schema was created which provides the structure for characterising the content of the elements of the XML documents and in this way restricts them [16].

RDF is a language for expressing data models. Many previous works [17], [18], [19], [20], [21] introduced the basic concept, definition, and syntax of the RDF. It should be noted that models developed in RDF can be represented in XML syntax. RDF provides a mechanism

for allowing any developer to make a basic statement about anything and then, to layer these statements into a single model. It has a formalism of a triple-syntax consisting of a) resource: subject, b) resource's properties: predicate and c) property values: object, which makes it be similar to a human language's syntax. This can also be assumed as an object *O* which has an attribute *A* with the value *V* [18]. RDF allows objects and values to be flexible and interchanged, and thus, any object can play the role of a value of a triple which is used for nesting and chaining graphs. Furthermore, RDF provides the tools to indicate that a given object is of a certain type [18]. Based on RDF several application works have been done. For example, Klyne [22] described some experimental works for modelling complex systems with RDF. He built higher-level constructs in RDF that allow complex systems to be modelled incrementally, without necessarily having full knowledge of the detailed ontological structure of a system. Although RDF provides several advantages, it does not provide any mechanisms for declaring property names that are to be used for further data modelling. To provide a solution to this, RDF Schema was developed which supports basic elements for the description of an ontology such as Class, subPropertyOf, and subClassOf. It provides a basic type system for RDF models which lets developers define a particular vocabulary for RDF data and specify the kinds of object to which these attributes can be applied [23]. This mechanism allows defining a common vocabulary for researchers and engineers, who are collaborating and need to share information on a domain. Nevertheless, still human effort is involved and the human is required to understand the way of thinking of the machine which costs and is time consuming.

A new tool to be used is the web ontology language (OWL). OWL introduces the expressivity of logic into Semantic Web and it allows developers to express detailed constraints among classes, instances and properties. OWL was designed to provide a common way to process the semantic content of web information. It was developed to augment the tools for expressing semantics provided by XML and RDF. OWL-based models can as well be represented in XML syntax. OWL provides more vocabulary for describing properties and classes (including relations between classes, enumerated classes, cardinality, equality and characteristics of properties). Thus, it supports greater machine interpretability of Web content than supported by the previous languages by providing

additional vocabulary along with formal semantics. It supports machine semantic interpretation which makes the machine to think more like the human brain.

2.2 OWL Characteristics

OWL (in the version OWL 1) exists in three sublanguages or species: OWL Lite, OWL-DL and OWL Full. All species have the status of "recommendation" of W3C and they have different level of expressiveness [24]. Each sublanguage was created in a way to cover different needs of applications and requirements of developers. In brief the description of the sublanguages is:

- OWL Lite was developed to support those users primarily needing a classification hierarchy and simple constraints. For example, it supports cardinality constraints but it only permits cardinality values of 0 or 1, it also does not include owl: oneOf and owl: hasValue constructs. OWL Lite provides a quick migration path for taxonomies and it has a lower formal complexity than OWL-DL. It was developed with the aim of being simpler to provide tool support for OWL Lite than the more expressive OWL-DL. However, later it was criticised as being complex to compute. This is because this language requires reasoning with equality, which significantly increases computational complexity. Moreover, cardinality restrictions introduce quality in a non-intuitive manner and there is no notion of constraints. All these combined with undecidability issues described in [25] made it difficult to extend OWL Lite with a rule language.

- OWL-DL was developed to provide the maximum expressiveness in tandem with guaranteeing both computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL-DL includes all OWL language constructs (such as transitive properties, which allow more of the semantics of sequences to be represented explicitly than in RDF or OWL Lite) and it allows modelling at multiple levels of abstraction (and thus, sequences of classes can be characterized by their general or more specific properties). However, the usage of the constructs is limited under certain restrictions. For example, a class may of an OWL-DL ontology not be both a class and an instance. The term "DL" in the name OWL-DL derives from the fact that it uses

description logics, a field of research that has studied the logics that form the formal foundation of OWL and they are described below.

- OWL Full which is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full some resource can be both a class and a member of a class (individual). OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. According to its developers, it is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

Each one of these sublanguages is a syntactic extension of its simpler predecessor. It should be noted that the inverses of these relations do not hold.

- Every legal OWL Lite ontology is a legal OWL-DL ontology.
- Every legal OWL-DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL-DL conclusion.
- Every valid OWL-DL conclusion is a valid OWL Full conclusion.

Furthermore, OWL provides the capability of creating classes, properties, defining instances and its operations.

- Classes are sub-classes of the root class which is owl:Thing. A class may contain individuals, which are instances of the class, and other sub-classes. For example, Resource could be the sub-class of class owl:Thing while Personnel Resource, Document Resource, and Equipment Resource are sub-classes of Resource.
- Properties are binary relations that specify class characteristics. There are two types of simple properties: datatype and object properties. Datatype properties of the classes are attributes of instances which have as input (and may contain) data values (i.e. string, integer, etc.). Object properties are relations between classes and they are used to link instances of the classes to each other.
- Instances are individuals that belong to the classes of the OWL ontology and they are the elements that make real use of the properties defined for the classes. A class may have any number of instances. Instances are used to define the relationship

among different classes (object properties) and contain the actual values for the datatype properties.

- OWL supports various operations on classes such as union, intersection, complement, class enumeration, cardinality, and disjointness.

2.2.1 Description Logics

The logic on which OWL is developed, is a model theory based on Description Logic (DL) [26]. Logic provides a framework for defining all the inferences that a modelling language needs. On a specific OWL-DL ontology model DLs provide the developer with the ability to describe concepts formally and to use the description of the concepts in order to query the model about its concepts and instances. DL knowledge bases consist of two parts: the T-box and A-box. T-box contains the terminology defined for each concept (class) i.e. definition of what a product is, and A-Box contains the actual data i.e. the instance Car_1 is a product since it fulfils the criteria defined in the T-box of what a product is [26]. A brief introduction to DLs is provided by Horrocks et al. [27].

2.2.2 Inference Engines-Reasoners

Based on DLs, DL-reasoners have been developed to extract valuable information from the OWL-DL models. DL-reasoners (which are also called inference engines) can be used to check consistency of the model, figure out equivalencies among concepts and infer subsumption of concepts. Furthermore, DL-reasoners can categorise instances under the concepts they belong. This provides reasoning power supporting decision and can answer database-like queries. For instance, typical questions which are answered with such reasoning are: is a particular instance (member of an A-box) a member of a given concept? which is a query to perform instance categorisation; does a relation/role hold between two instances, in other words does A have property B? which is a query to perform relation checking; is a class a logical sub-class of another class? which queries about subsumption and re-classifies the class-hierarchy; and is there contradiction among definitions? check the consistency of the concepts in the model.

A very interesting survey on several semantic web technologies including DL-reasoners was carried out by Cardoso [28] in the period from Dec 2006 to Jan 2007. In the survey

participated 627 ontology developers from various sectors of academic and industrial research. According to this survey, actual DL-reasoners ordered according to the number of users are: Jena [29], Racer [30], Pellet [31], Fact++ [32]. The full list and the results of the survey can be found at [28]. Although Jena is the most popular and the mostly used reasoner (it has a very powerful RDF-Schema reasoner), it has several limitations on its OWL reasoner which lead us not to select it [33]. Even in the manual of the Jena inference engine it is recommended to use an external DL-reasoner in order to have a complete OWL-DL reasoning. Racer is a very powerful inference engine which implements the W3C standards of RDF and OWL. However, still (in version 1.8.1) it does not support reasoning on SWRL rules and in its latest release it provides limited support. Fact++ is also very powerful, but it does not provide any support for SWRL rules. The basic architecture and characteristics of Pellet are described by Sirin et al. [34]. Its advantages and its disadvantages in comparison to Racer and Fact++ are described by Sirin et al. [35] (at least for the period that the paper was written 2004). Pellet at that period was not as powerful as the other two reasoners for very big ontologies, but it implemented a more complete reasoning for OWL-DL and supported semantic web capabilities [35]. Pellet in our use cases proved to be as efficient as Racer and Fact++ and it also has the advantage (version 1.5.2) of being able to reason on SWRL rules.

2.2.3 Open World Assumption

Another OWL characteristic is that it uses Open World Assumption (OWA), in contrast to databases (i.e. SQL databases), which adopt the Closed World Assumption. According to OWA, if a statement cannot be proved to be true using current knowledge, the system cannot conclude that the statement is false [36]. Systems using OWA assume that there may always be more information (classes, DLs, etc.) to be added to the ontology model at later stages and thus, developers are able to extend other developers' models. Under OWA the DL-reasoner cannot determine that something is true or false unless it is explicitly stated in the model. This practically means that if we query a database to retrieve some data which it cannot find, it will return a negative answer whereas the DL-reasoner applied on an OWL model makes no conclusion. A good practical example for this is provided by Drummond et al. [36] on slide 18. We assume that the system is a doctor and wants to treat

a patient with a painkiller that is not an anticoagulant. The given information to the system is that a) "Aspirin" has the effect "Painkiller", b) "Wharfarin" has the effect "Anticoagulant" and c) "Paracetemol" has the effect "Painkiller". When we query the system to tell us which drugs we can use (in other words to tell us which drugs are painkillers but are not anticoagulants), a database would return "Aspirin" and "Paracetemol", where as OWL (due to OWA) cannot say this and will not return anything (i.e. in this case the fact that "Paracetemol" and "Aspirin" have the effect "Painkiller", does not imply that they are not anticoagulant). If we add the constraint that "Paracetemol is not an anticoagulant" and "Aspirin is not an anticoagulant" then it will return "Aspirin" and "Paracetemol". OWA should be understood well by the developers before they start modelling.

2.2.4 Rule Language for the Semantic Web

Semantic Web Rule Language (SWRL) is a proposal submitted to W3C in order to add rules to the Semantic Web that go beyond OWL [11]. This language combines OWL (the sublanguages of OWL-DL and Lite) with the Rule Markup Language (with the sublanguages Unary/Binary Datalog) [37]. One of the most interesting objectives of SWRL is to be a language for sharing rules and therefore, to support interoperability of the rule systems on the Semantic Web [38].

On the other hand, another proposed language for introducing rules in the Semantic Web is the Description Logic Programs (DLP) [39]. The basis of the logic of the DLP is the intersection of Horn logic and OWL. Since the logic on the background of these languages is different, research has been carried out (Horrocks et al. [40]) on which is the most appropriate and the long-term dangers of using both.

Protégé-OWL [41] comes with a built-in tab for a SWRL Editor [42] which is interactive and fully-featured. Protégé also supports a plug-in mechanism for integrating third party rule engines such as the Jess rule engine [43]. SWRL combined with Jess can provide a rich rule-based reasoning facility for the Semantic Web. After integrating Jess into Protégé-OWL, SWRL Factory mechanism is used to integrate the Jess rule engine with the SWRL Editor. The utilisation of Jess into the SWRL is performed by the SWRL Jess tab [44]. By using Jess, users are able to run SWRL rules interactively in order to create new OWL

concepts and then insert them into the OWL model. The interaction of SWRL with Jess is a starting point for further rule integration efforts [43]. Based on SWRL, engineers have also developed a query language, the SQWRL (Semantic Query-Enhanced Web Rule Language) [45] which also supports queries with complex closure requirements [46].

In SWRL, rules are consisting of two parts: the first part which is an antecedent and the second part which is a consequent. The logic performed during execution is: if the conditions defined in the antecedent are true (hold), then (consequently), the conditions specified in the consequent must also be true (hold). For instance, we have instances of a class called Field_Data and each instance has a datatype property called Date which takes values of the type xsd:dateTime. We want to query this class and see which of these instances have a value Date which is in June of 2008. Then, we create a class called Field_Data_in_200806 under which we want to categorise the instances that fulfil our query. (It should be noted that in OWL there is no a direct way to make this type of query). The SWRL rule to perform this query is:

$$\text{Field\_Data}(?fdx) \wedge \text{Date}(?fdx, ?zx) \wedge \text{temporal:after}(?zx, \text{"2008-05-31T23:59:999"}) \wedge \text{temporal:before}(?zx, \text{"2008-06-30T23:59:999"}) \rightarrow \text{Field\_Data\_in\_200806}(?fdx)$$

This practically means if there is an instance (?fdx) of the class Field_Data AND if this instance has a Date datatype property (?zx) AND if this date has the quality of being after 2008-05-31T(=time)23:59:999 and before 2008-06-30T(=time)23:59:999 (which actually means that 2008-05-31T(=time)23:59:999<(?zx)< 2008-06-30T(=time)23:59:999), then the instance (?fdx) must also be an instance of the class Field_Data_in_200806. Still, this "knowledge" (that (?fdx) must also be an instance of the class Field_Data_in_200806) is not transferred in the OWL model. To achieve this the Jess rule engine is used, and the (?fdx) is made an instance of the class Field_Data_in_200806. For more details about the utilisation of these tools in this work see also Figure 11 and the description of the system architecture in section 4.1.

2.2.5 Evolution of OWL: The OWL 2

OWL has become a W3C recommendation since 2004 and since then, several users have sent their feedback which has provided the ground for improvements [47]. This was understood by the language developers and they continued developing a new version of

OWL which was initially called OWL 1.1 and finally, renamed to OWL 2 [48] and became a W3C recommendation in October of 2009. OWL 2 introduced several new features in OWL [64]. In OWL 2, there are three sublanguages (profiles) [65]: OWL 2 EL which supports polynomial time reasoning but does not support so much expressiveness and generates reasoning with priority in speed (performance). It is more suitable for very large ontologies; OWL 2 QL which is designed to enable easier access and query to data stored on a database using standard relational database technology. It is more suitable for light ontologies which contain a very large number of instances; and OWL 2 RL which enables the implementation of polynomial time reasoning algorithms using rule-extended database technologies operating directly on RDF triples; it is more suitable for cases that require high expressiveness and scalable reasoning.

It should be noted that OWL 2 supports backwards compatibility with OWL 1 and hence, all OWL 1 ontologies remain valid OWL 2 ontologies [48]. However, to date the available editors, reasoners and tools for OWL 2 are not as much developed as for OWL 1.

2.2.6 Merging Ontologies

A very useful achievement is to collect the distributed knowledge and easily merge it together for future use. This requirement arises since a lot of data and information is stored in stand-alone PLM systems which cannot work together without great efforts of manual mapping. This burden may be solved by efficient ontology merging. In our case it is very important to be able to merge ontologies. The outcome of the merging should be checked for its consistency and tools should be capable to reason on a group of ontologies.

Merging two ontology models is an equivalent process with being able to merge two or more PLM models together. In the case of having machine-understandable models, users are able to apply inference engines in order to figure out automatically the logical similarities and differences between the models. Furthermore, the DL-reasoner is able to reason on the new ontology model which derived from the merged ontologies and this could create new knowledge since the DL-reasoner logically puts all the parts together. This finds practical application for example in collaborative environment during the MOL when the maintenance teams are working remotely and at the end the OEM brings their works together to process the data and to extract knowledge about its products.

Up to date a number of methods and tools have been developed towards ontology merging and alignment. Stumme et al. [49] developed the FCA-Merge method for merging ontologies. This method follows a bottom-up approach. The generated result of the method always requires human interaction to be explored and transformed into the merged ontology. Noy et al. [50] developed the Prompt suite for merging ontologies. However, after merging there were a number of difficulties and problems on inconsistencies and other problems (naming conflicts on classes, dangling relationships and attributes limits or types, etc.). Kotis et al. [51] developed the HCONE-merge approach which is aiming towards automating the merging process. The approach makes use of the intended informal meaning of concepts by mapping them to WordNet senses using the Latent Semantic Indexing (LSI) method. Firstly, HCONE-merge automatically aligns and then merges ontologies based on these mappings and using the reasoning of DLs. Moreover, HCONE-merge method is tested for ontology mapping with varying degrees of human involvement and it is evaluated experimentally. The authors conclude that by using this method they reach a point where ontology merging can be carried out efficiently with minimum human involvement.

From the computer science point of view there is also another similar issue which is defined as modular use of ontologies and it has the aim to allow ontology re-use. One may find which part of an already developed ontology is useful for his own project and re-use it. In this way the developer avoids reinventing the wheel. Relevant work has been carried out by Grau et al. [52]. The authors have developed a theory to provide methods and tools for extracting the right parts of already developed ontologies in order to re-use it. The method is implemented and applied on a number of very well known ontologies. Based on previous works, Jiménez-Ruiz et al. [53] proposed a methodology for safe and economic re-use of ontologies. Also, another work deals with the safety of actually doing in practice re-use because after using parts the new model should maintain consistency [54]. Finally, Grau et al. [55] provides the background for making the modular re-use practically possible. The main issue which remains open is that the import or re-use of parts in a model might create logical conflicts with the already existing concepts in the model.

The conclusion is that more research still should be performed in this field in order to provide methodologies for ontology and rule development in order support automated

ontology merging [56]. The research could also be towards providing model-developing rules for safe extension and consistency after merging.

2.2.7 Ontology Editors

Several ontology building editors have been developed since the emergence of the idea of ontologies and the semantic web. The history about them is similar to all newly introduced products: in the beginning there are many producers and eventually only few survive and nominate the market i.e. in the beginning of personal computers there were many companies developing computer processors for PCs and today only 2-3 producers nominate the market. A big list of 94 editors which were available for use in 2004 is found in [57]. Only a handful of those tools is still supported and updated to facilitate new languages such as OWL. The most well known are the Protégé editor [41] which was developed by the University of Standford, the SWOOP editor [58] developed by the University of Manchester and the OntoStudio editor [59] which is marketed by Ontoprise. The ancestor of OntoStudio was called OntoEdit and was developed by the University of Karlsruhe. According to the survey carried out by Cardoso [28] involving 627 ontology developers from various sectors in the period from Dec 2006 to Jan 2007 the usage of ontology editors was: Protégé (68,2%), OntoStudio (17,7%), SWOOP (13,6%). It should be noted that a number of developers is using more than one editor. More details about the most popular editors can be found in [60].

OntoStudio is marketed by the company Ontoprise and it uses the OntoBroker as a reasoner. It supports among other languages: RDF-Schema, OWL and F-Logic [62]. There are several limitations on its OWL editor (i.e. it cannot represent enumeration of classes). Most commonly this editor is used to build ontologies using F-Logic. It also provides a method for extending the tool with plug-ins. It should be noted that the fact that it is not open source is the major drawback for not selecting to use this editor.

SWOOP is a very interesting lightweight OWL editor [61]. It contains its own reasoner; it is developed as a separate Java application; it provides a browser-like environment and it is extensible via a plug-in architecture. The latest version available was developed in 2006.

Protégé has more than 100 000 registered users and a significant number of ontology projects have been developed using this tool. It has the Protégé-OWL editor for building

ontologies in all three species of OWL and has frequent updates and good support. The significant number of users provides feedback for the Protégé developers which have lead to very frequent software updates to deal with issues, plug-ins and efforts to implement the latest versions of OWL. Projects developed in Protégé may also be processed in Eclipse software editor [63] which provides a familiar software development environment. Protégé is easily accessible (open source), it allows software developers to develop their own plug-ins, it provides UML- and Database- back-ends with which a project developed in Protégé may be saved as UML and database respectively, etc. Protégé goes beyond OWL with the use of SWRL and the Jess rule engine. It also contains a number of useful plug-ins to import, export and present data: DataMaster, Queries Tab and Jambalaya respectively. Finally, the Protégé-OWL (in version 3.4) comes with an integrated Pellet DL-reasoner.

2.3 Conclusion

The aim of this chapter is to present the basic knowledge of the IT methods and tools used in this dissertation. The content of this chapter does not claim to be exhaustive and the reader should always refer to the sources in case more details are required. The presented methods and tools theoretically are very powerful and could provide many benefits if implemented in PLM systems.

Description Logics combined with a DL-reasoner may be used for:

- Checking the class-hierarchy for its consistency.
- Defining/figuring out if there are equivalent classes in the class-hierarchy.
- Re-classifying classes in the class-hierarchy according to the concept that they describe.
- Inferring/categorising instances under the classes that they logically belong.

Moreover, the sublanguage of OWL, the OWL-DL, provides a good level of expressivity (i.e. transitive properties), it allows modelling at multiple levels of abstraction, and at the same time the models developed in this language are decidable.

Our challenge is to find ways of implementing these methods and tools in PLM systems to provide new opportunities and functionalities towards Closed-Loop PLM such as data integration, system interoperability and data continuity along time. In chapter 4 of this

dissertation is provided an implementation method to implement these advantages in a PLM model in an efficient and simple manner, and in chapter 5 applications are presented.

3
State of the art

This chapter contains the related works and literature to this dissertation. It is divided in three parts presenting related works and literature in: PLM systems, ontology applications in various domains as well as in PLM, and works which deal with the time management. The aim of the PLM systems review is to show the strong points of the currently used systems, to show the parts of these systems which we are using, and to emphasise in the existing gaps of the systems regarding Closed-Loop PLM. Furthermore, we present a brief description of existing applications using them in various domains. The aim is to demonstrate the capabilities of the existing technology and the opportunities they would provide in PLM systems when implemented. Finally, we present literature dealing with time management. It is divided in two parts: one describing time concepts and the other one describing time management approaches. The aim is to demonstrate the different approaches dealing with time elements in the different models of various domains. We study how time is treated in order to demonstrate how we concluded in proposing the methodology presented in chapter 4. Although time is naturally a common element of the different PLM systems and it has the characteristic of being objective, in our opinion, it is one underexploited element in the current models. Therefore, we have selected time in order to propose improvements in the current models.

3.1 State of the art in PLM

The main goal of Product Lifecycle Management (PLM) is the management of all the business processes and associated data. Data is generated by events and actions of various lifecycle agents (both human and software systems) and it is distributed along the product's lifecycle phases: Beginning of Life (BOL) including design and manufacturing, Middle of Life (MOL) including usage and maintenance and End of Life (EOL) including recycling,

disposal or other options [3]. A major requirement for efficient PLM is the traceability of the product which means to acquire information along the product's lifecycle about the product. Furthermore, making this information "smart" instead of "dump" is a key aspect of future systems aiming to boost performance in data management and in the transformation of the data into information and into knowledge. A big amount of this information-knowledge is being lost, due to lack of reasoning capabilities as well as lack of interoperability and integration of elements of today's PLM systems and models. Therefore, a new generation of intelligent models is required. Extracting knowledge in order to improve features of products and of future products is a very promising target field of using this information.

Most of the current ontology information models in PLM are developed using class diagrams in the Unified Modelling Language (UML) [66]. This language is human understandable and class diagrams are used to represent the domain. However, the language provides a loose interpretation of the meaning of the diagrams which creates problems for the machines. UML has many limitations when coming to object oriented modelling which include lack of precise semantics and of practical analysis techniques [67]. Since the development of the language, several efforts dealing with these limitations have been performed [68], [69].

A first step towards achieving interoperability and therefore, into allowing data exchange between different platforms used by various lifecycle agents' platforms, is the definition of a common-hierarchy data structure. Towards this direction are aiming models developed within standards covering parts of this domain. The thorough control and distribution of information between different lifecycle agents and phases is the underlying goal for the PLM approach. Moreover, using new tools with additional reasoning capabilities prove to be very promising for facilitating future PLM systems.

In this section a brief description of several models and standards is presented, demonstrating the important elements of the different standards and models which lead us to use them. Also a number of previous works is presented, in which these standards are implemented, extended or interrelated.

3.1.1 PLM Models

Emphasis is given on MIMOSA and on ISO-15926 standards. They are models used on their sector and on the specific phases of the PLM. Currently, there are various lifecycle information management systems developed by different vendors and they are covering a limited part of the lifecycle (i.e. design, maintenance activities). Many of these systems have their own unique data exchange interfaces which is a big burden to integrate the data contained in them. In the extended enterprise a number of suppliers might be using different information systems from each other. This creates problems of data integration at the Original Equipment Manufacturer (OEM) level where data from all these systems must be combined and utilised. The solution to this situation is not an easy task since different integration techniques bring their own advantages and disadvantages. One solution is to use systems from a single vendor, however the vendor may not provide a total information management solution, suppliers might have obligations to use other systems due to their cooperation with other OEMs and the dependence on one vendor can prove dangerous in various ways. Another solution is to purchase a commercial custom bridge that integrates different systems or build one internally. The first might be more cost effective and does not require own resources but the latter can be customised better to the specific task needed. The wider use of standardised information systems could lead to a solution to this important burden of data integration.

MIMOSA

Machinery Information Management Open Systems Alliance (MIMOSA) is an alliance focused on developing consensus-driven open data standards to enable interoperability of "operations and maintenance" (O&M) processes, systems and actors [70]. Standards developed in the framework of MIMOSA are aiming to be widely accepted and to be used in facilitating seamless asset management data exchange through integration. For the enterprises adopting such standards the result will be to eliminate the information gaps which exist among the different systems such as real-time control systems and business information systems. The aim of MIMOSA is to encourage the adoption of open information standards by introducing the MIMOSA OSA-EAI (Open System Architecture for Enterprise Application Integration). This is a standard for data exchange of engineering

asset management data about all aspects of equipment, including the physical configuration of platforms, the reliability, condition, and maintenance of platforms, systems, and subsystems. To this end MIMOSA provides a series of interrelated information standards:

- The Common Conceptual Object Model (CCOM) which provides the basic conceptual model basis for OSA-EAI.
- The Common Relational Information Schema (CRIS) which provides a common implementation schema and allows information from many systems to be communicated and integrated (vertical and horizontal integration)
- Metadata reference libraries and a series of information exchange standards which use XML and SQL.

The structure of the standards is briefly described in the Technical Architecture Summary document [71] of MIMOSA OSA-EAI and it is shown in Figure 5 (adapted from [71]). As shown in Figure 5, on top of the CRIS there is a reference data library which contains reference data compiled by MIMOSA. This library facilitates the communication between MIMOSA-compliant systems. Thus, OSA-EAI provides open data exchange standards in several asset management areas including work management, diagnostic and prognostic assessment, vibration and sound data, oil, fluid and gas data, and reliability information. Advantages of using OSA-EAI as a basis to build databases for asset management data include software re-use and data interoperability [72].

Another MIMOSA standard is the OSA-CBM (Open System Architecture for Condition Based Maintenance) which provides the architecture for moving information in a condition-based maintenance system and also provides the tools for implementing the architecture. Its architecture is based on ISO-13374 and comprises of six blocks: data acquisition, data manipulation, state detection, health assessment, prognostic assessment, and advisory generation [73]. The first three involve only devices which collect and process data to detect abnormalities. The rest combine devices and human agents to define the health status of the equipment, to predict future faults and to support decision. OSA-CBM uses many of the data elements that are defined by the OSA-EAI and in the future the aim is OSA-CBM to be mapped into OSA-EAI.

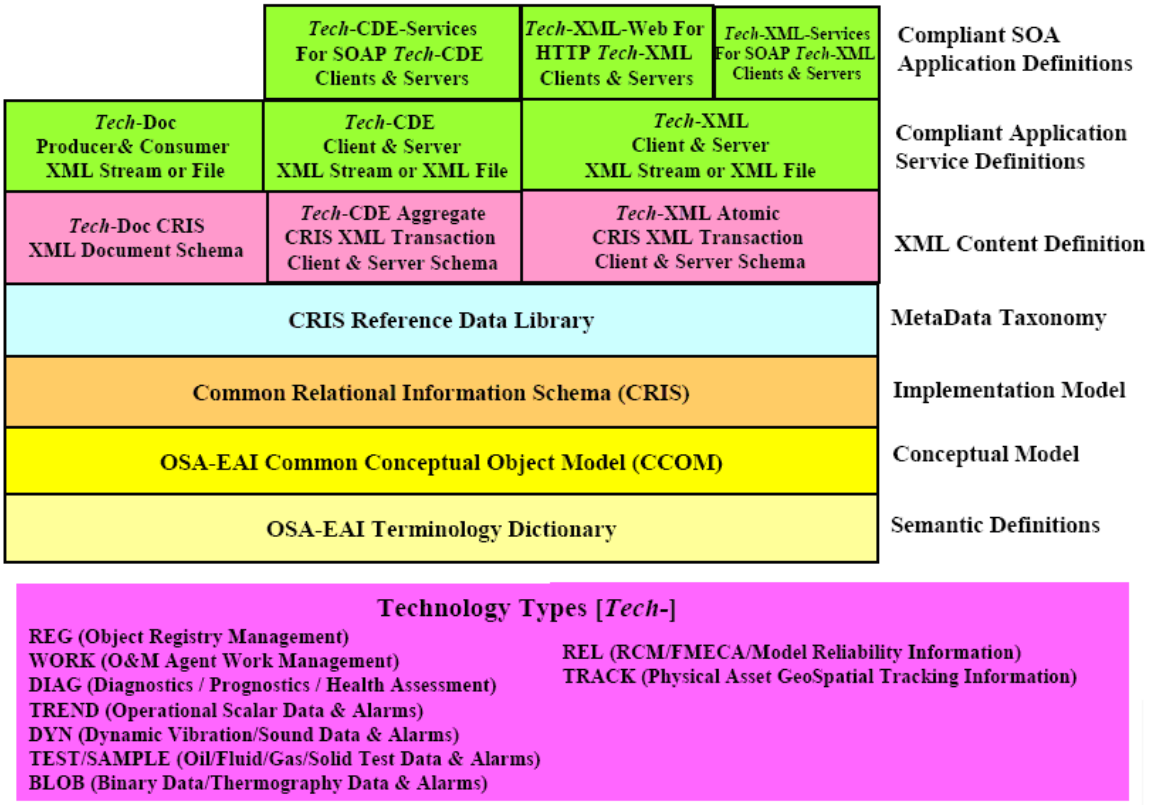

Figure 5: MIMOSA OSA-EAI v3.2 Architecture Diagram.

There have been several implementations of the OSA-CBM which have highlighted both advantages and weaknesses of the system. Firstly, we present the works which demonstrate the functionalities and advantages of the OSA-CBM, and then the works which propose changes and extensions to fulfil various requirements. Keller et al. [74] developed a vehicle health management system which is based on the OSA-CBM. The developed architecture is flexible and extensible towards supporting prognostics and decision support for maintenance. Byington et al. [75] developed an OSA-CBM-based system which was used for the diagnostics and the prognostics of the health management of avionics. The authors support the use of open data architectures and it has been demonstrated through a paradigm that such architectures enable information continuity and knowledge transportability between on-board and off-board systems or maintenance aids. Chidambaram et al. [76] used the OSA-CBM to monitor an electro-hydraulic test rig. The OSA-CBM architecture proved to be effective into collecting, processing and displaying sensor data as well as displaying the process results. Furthermore, the flexibility and extensibility that the OSA-

CBM brings to the maintenance system was demonstrated by using a variety of commercial and proprietary software tools (Fast Fourier Transforms, Neural Networks, Regression Analysis, etc.). Lebold et al. [77] have developed skeleton code to implement a functional communication system for the different layers of the OSA-CBM system.

Moreover, there are some works which propose changes and extensions of the model to make it more generic and to cover a wider area of the domain. Voisin et al. [78] in the framework of FP6 project DYNAMITE: Dynamic Decisions in Maintenance developed an e-maintenance platform to include prognosis in their model. The authors suggested extensions in several parts of the OSA-CBM model in order to formalize the prognosis objects and data. Mathew et al. [72] in their work developed a condition monitoring system called BUDS which claims to support advanced diagnosis and prognosis models not available in commercial systems. BUDS database is based on OSA-EAI and the authors describe several issues of the OSA-EAI which they defined during the development of BUDS including the lack of documentation and excessive normalisation.

ISO 15926

Another significant standard is the ISO 15926 which is called “Industrial automation systems and integration -- Integration of life-cycle data for process plants including oil and gas production facilities”. Initially the coverage of ISO 15926 was focused on the process industry. However, its coverage has increased and hence, has become more generic and less specific to a particular industry domain.

ISO 15926 consists of 7 parts. Each part has a unique function:

- ISO 15926-1 provides an overview of ISO 15926.
- ISO 15926-2 specifies a generic, conceptual data model that supports representation of all lifecycle aspects of a process plant. In this part an interesting method for managing time is presented which is discussed in section 3.3.
- ISO 15926-4 defines a reference data library that can be periodically updated by a competent body, designated by ISO as a registration authority, which has the requisite infrastructure to ensure the effective use of the reference data library.
- ISO 15926-5 specifies the procedures to be followed by a registration authority for reference data.

- ISO 15926-6 specifies the information required when defining additions to the reference data specified in ISO 15926-4.
- ISO 15926-7 (old) provides implementation methods for the integration of distributed systems (currently not available at the ISO website)

ISO 15926-7 (old) has been revised and will be split into 4 parts [79]:

- ISO 15926-7 Template Methodology
- ISO 15926-8 OWL
- ISO 15926-9 Façade Implementation
- ISO 15926-10 Abstract Test Methods

It should be noted that the developers of the standard have developed an ontology model which is discussed in section 3.2.1.

There are several significant applications implementing this standard. Teijgeler [80] points out the importance of a uniformly structured information chain for data across all lifecycle of the parts of a system. ISO 15926 combined with semantic web technologies may provide a solution towards this goal. However, Semantic Web technologies have weaknesses which are burdens for implementing them and therefore, they have affected the work of ISO 15926 community. According to the author these are: scarcity of semantically annotated information sources, performance and scalability and the lack of a standard rule language**.** The latter makes it impossible to write sets of rules that can be used in different implementations. Batres et al. [81] present a method for the identification of hazard scenarios. The proposed method is based on the concept of "hazard scenario graphs" or HSG. HSGs are visual representations of the sequences or networks of events and activities in a hazard scenario. HSGs are based on concepts defined in the ISO 15926. This standard includes the definition of kinds and structures of objects, properties, events, processes and relations which can be used in the integration of material property data, equipment information, maintenance activities, etc. and provides the background for recording how the plant changes as a result of normal or abnormal activities. The latter is critical during the analysis of contributing causes. Elements of the ISO 15926 used are: activities, events, physical objects, participating entities, causal relations, temporal relations and participation relations.

Furthermore, an Integrated Information Platform was developed using this standard and it was implemented in several applications. Sandsmark et al. [82] describe the framework of the project "Integrated Information Platform for reservoir and subsea production systems" (IIP) that is supported by the Norwegian Research council. In this project the concept is to develop an information platform combined with the use of ontologies to overcome proprietary and system dependent data definition that prohibits effective exchange, sharing and integration of information. As a basis ISO 15926 is used, since its generic concept model makes it ideal as an integration platform for other standards. Gulla et al. [83] describes the work done within the IPP towards transforming and extending existing standards into OWL ontology for reservoir and subsea production systems. This ontology used for analysing data and interpreting user needs, may allow data to be related across phases and disciplines, helping people collaborate and reducing costs and risks. Tomassen et al. [84] based on the work done in IIP project propose a method to improve information retrieval quality by using ontologies. The ontology used is the one developed in IIP, which is based on ISO 13628 and it will be modelled in ISO 15926. Strasunskas [85] presents research in IIP on development of rule-based notification in subsea production systems to monitor and analyse production data. The author concludes that the full expressive power of OWL (OWL Full) is needed in order to represent ISO 15926-2/4 which is a burden for reasoning (reasoning is incomplete) and inference (undecidability). Moreover, a certain future work will be the alignment of the method developed in IIP with MIMOSA's open systems architecture for condition based maintenance.

There are also works describing other applications of the standard. Klüwer et al. [86] describe how OWL can be used with ISO 15926 to represent common industry classes and relations. The authors note the need to provide an interface to the modelling patterns that is familiar to professionals. For this reason they combine the ISO with rules and provide simple templates for user interface. Price et al. [87] describes the implementation of OWL into OASIS Product Life Cycle Support (PLCS) [88]. First step is the use of Semantic Web technology for developing Reference Data which includes the re-use of Reference Data of ISO 15926. Stell et al. [89] use aspects of ISO 15926 in their work for developing a four-dimensional ontology to show the spatio-temporal dimension of entities. Mun et al. [79] demonstrate an application of ISO 15926 using part 7. The tools used are mainly RDF and

SPARQL (see section 2.1). The implementation is about a nuclear power plant in Korea and the goal is to support sharing of data among interested parties in a semantic web environment.

Through this brief description one may realise the significance of standards and the way that researchers are trying to interrelate them in order to obtain models containing the combination of their benefits. Still, work should be performed in order to develop common terminology, define the generic needs for the standards and develop the standards using tools which will make them flexible, transferable and extensible.

3.1.2 Closed-Loop PLM-Semantic Object Model of PROMISE

In this section the Semantic Object Model (SOM), developed in the PROMISE FP6 project [90], is briefly presented. One of the aims of PROMISE was to develop a Closed-Loop PLM system which uses smart embedded IT systems and allows the seamless flow of data and information in order to close the product lifecycle information loops. The developed SOM was applied, tested and validated in eleven application scenarios developed in cooperation with industrial partners. The SOM was developed using UML. It is a product item oriented model achieving both an efficient description of the product as it is designed from the manufacturer and a functional structure for storing data of the product's lifecycle. The schema presenting classes, attributes and relationships of the SOM is shown in Figure 6. The SOM schema as well as more details about the SOM can be found at [91] and [92]. The SOM has been used and tested in a number of application scenarios covering all phases of PLM and a wide range of different industrial sectors. For each application scenario only a small part of the SOM was used, necessary for the scenario and was extended with more detailed classes. The SOM provided a commonly accepted schema to support interoperability when adopted by different industrial partners. Although generic and extensible, the model inherited several limitations due to UML. These include the fact that models developed with this language are not well defined in lean and high extend in order to be machine understandable and therefore, in case of model extension the final models lose interoperability and data integration.

In the rest of this work, the following naming conventions are used: names of classes are written in boldface and capitalized/lower case Arial (i.e. **Product_EOL,** etc). Names of

attributes and associations (also called relationships) are capitalised /lower case Courier New (i.e. `isDesigned`) while names of instances are in italics Arial (i.e. *Passenger_Vehicle_1*).

The SOM consists of 26 classes and has two main parts focusing in different fields of information about the product. The first part of the model contains the information needed to describe the product instance and its characteristics. Architecture for categorising information about the product's type, conditions, properties, the product's serial number, data from BOL of the product, etc. is included here. The most important class is the **Physical_Product** class. This part is shown in Figure 6, bounded by the continuous line. The second part of the model is focusing on the life cycle phases of the product. The necessary architecture for managing and categorising valuable information about the main events such as breakdowns, and activities such as maintenance of the product is included here. This information is in a later phase used to support decision of life cycle agents of all PLM phases such as maintenance crew, the designer, the production manager, etc. Moreover, the architecture for storing field data (i.e. repetitive field data from sensors) for further analysis is included here. The most important classes are the **Field_Data**, **Event** and **Activity** class. This part is shown in Figure 6, bounded by the dotted line.

The functionality of the SOM is quite simple. Firstly, the list of the physical products is stored in the **Physical_Product** class. The physical products may be complex products which consist of many parts such as vehicles or simple which consist of only one part such as a screw. This is described through the **Part_Of** class which also contains the duration of the time that a specific part is part of a more complex product. In this way the model preserves continuity of the information about the physical product. (The complexity of the product and its parts in the OWL ontology model developed in section 4.2 is described through a "physical product to physical product" object property `hasParent` and its inverse `isParentOf`.) Depending on the requirements of the application the level of detail which is considered as "simple" may vary. Even for the same product, in different cases, one might have different levels of detail: i.e. the level of detail is different for products of a fleet management company and different for a single user who might be interested to have more detailed model for the one product he is using. The properties of the

products are stored in the **Property** class, the **URI** class, **Information_Provider** class and **ID_Info** class. Furthermore, each physical product is related to the **Life_Cycle_Phase** class which enables each product to be related to one or more instances of a lifecycle phase i.e. to multiple instances of the MOL. (In the model developed in section 4.2 this relationship combined with the object properties `hasParent`/`isParentOf` allows the information system to track information about the product through its different phases as well as types of usage and therefore, preserve continuity of information about the physical product). Thus, the model stores information about which data is related to the product for each of its use. During its lifecycle the product is monitored with sensors which collect valuable data of different types such as temperature, pressure, velocity, viscosity, etc. in various measurement units such as Celsius, bar, m/sec and Pascal-second respectively. The different sensors related to the product are stored in the **Field_Data_Source** class and the types of the data collected are stored in the **Valid_Field_Data_Type** class. The collected data from the sensors is stored in the **Field_Data** class and in documents if necessary. In the **Condition** class it is stored a list of the required or recommended conditions for the well-functioning of the products. These conditions may vary depending on the product and are adjusted according to various criteria. Then, the data of the **Field_Data** class is compared with the conditions. If one or more conditions are not met, one or more events are created and stored in the **Event** class. Events depending on their severity may trigger activities such as maintenance, part replacement, etc. which are stored in the **Activity** class. To perform activities several resources are used. The available resources are in the **Resource** class. Finally, activities may cause events (i.e. start, finish, etc.). This part of the model combining activities, events and resources is the part which supports the actual maintenance. Many more details may be found in the PROMISE Research Deliverable 9.2 [91].

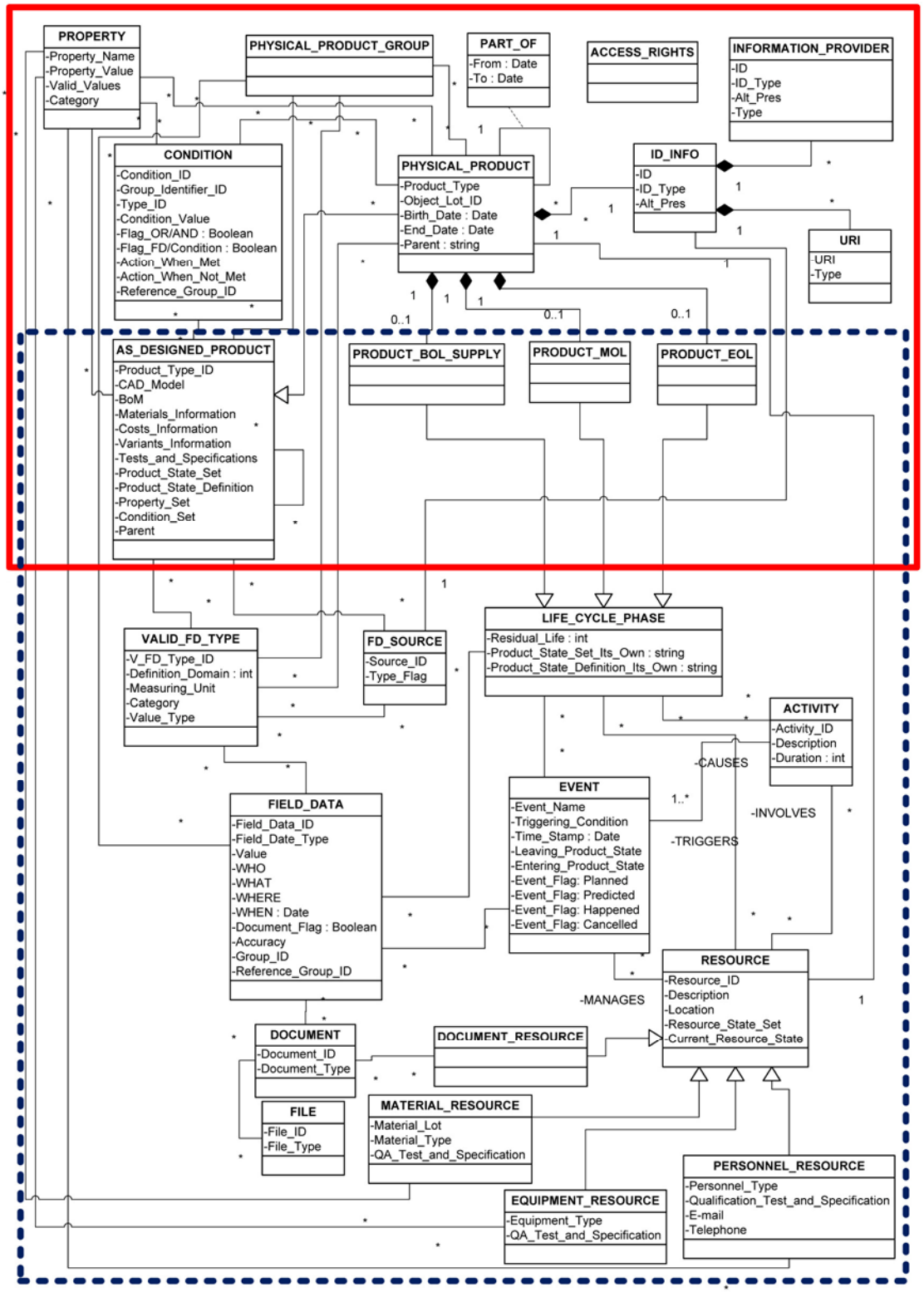

Figure 6: Complete schema of the PROMISE SOM.

3.2 State of the art in Current Ontology Models

In the ontology development process, the starting point is the definition of the terminology to be used. Then, domain ontologies describing these definitions are developed. Domain ontologies will take advantage of the shared common terms and definitions, and therefore, they will support data interoperability among software and database applications. The concept is that such ontologies will be re-used and used as the basis for developing application specific ontologies which will facilitate semantic interoperation between applications. Still, research on ontology re-use is limited and there are no widely accepted techniques to follow during the ontology development in order to support it.

In this section ontologies developed in various scientific domains are presented. Firstly, the main ontologies developed in the section of bio-informatics are presented. This is done because in this sector the most significant applications of ontologies have been performed. The requirements and the tests of these applications provided the most important initiatives for later improvements of the ontology tools. It should be noted that Semantic Web experts have chosen this domain to perform the widest applications of these tools due to its credibility and objectivity of terms. Medical terminology is well-defined and widely accepted by the related scientific society. For example, the term *poliomyelitis* is understood the same by all physicians in the world. This is an element missing in the engineering sector. For example, the term *product* one may define it as a thing which can be traded and another may define it as a thing which can be maintained. Moreover, the main works in the engineering and the PLM sectors are presented. The main characteristic of these works is that the notion of ontology varies. Usually it is used to express a UML model which is only human understandable, but in practice they are limited to represent the structure of a database. In very few works advantages of DLs are actually used and they are focused on inferring instances, without exploiting the full potential of DLs. Furthermore, applications of ontologies in the PLM are mainly focusing in the BOL, there are very few in the MOL and even fewer considering the whole lifecycle. Finally, there are no major works in the field of the whole PLM achieving the full implementation of ontology based IT methods and tools which leaves a significant open field for research and innovation.

3.2.1 Ontology Models

Already, ontologies have been implemented in various scientific fields. In medicine efforts for categorising all the terminology and development of structured vocabularies for health care into an ontology are in process in the SNOMED project [93] and the semantic network of the Unified Medical Language System [94]. The 2008 release of SNOMED [95] contained over 311 000 active concepts (classes) portrayed by almost 800 000 active descriptions and associated to each other by more than 1 360 000 relationships. SNOMED has provided the field for several tests and suggestions for improvements of OWL ontology capabilities as well as the related tools. Bodenreider et al. [96] have developed methods making subsumptions for the over 200 000 classes (at the time) of SNOMED. Horrocks et al. [97] in the "Instance Store" have developed a method dealing with problems arising when ontologies have large number of individuals. Brandt [98] in his work shows that using general concept inclusion (GCI) axioms and role hierarchies in EL terminologies preserves the polynomial time upper bound for subsumption and therefore, he claims that reasoning over SNOMED is possible in polynomial time. Even in the official W3C document describing the specifications and the details of the sublanguage OWL-EL, SNOMED is used as an example for applying this language [65]. Other significant ontologies in this field include the National Cancer Institute (NCI) Ontology [99], the Gene Ontology (GO) [100] and the GALEN ontology [101].

In the field of engineering there are several works developing general purpose upper ontologies. The two most referenced ontologies are the Suggested Upper Merged Ontology (SUMO) [102] and the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [103]. The SUMO is developed by the IEEE Standard Upper Ontology Working Group and it consists of approximately 4 000 assertions and 1 000 concepts. Its aim is to "*provide a structure and a set of general concepts upon which domain ontologies (i.e. medical, financial, engineering, etc.) could be constructed*". Domain ontologies based on SUMO will take advantage of the shared common terms and definitions and therefore, they will support data interoperability among software and database applications as well as interpreting natural language. SUMO will also support automated reasoning and inferencing. Another significant ontology is the DOLCE ontology developed in the framework of FP5 WonderWeb project in order to support understanding of the

information contributed by the different project partners. One of the main difference between DOLCE and SUMO is that DOLCE is a more complex ontology since it uses many OWL-DL constructors. The aim of DOLCE is to capture the ontological categories of the natural language and the human common-sense [104].

Ontology models developed in PLM are focusing in both translating existing models and developing new models into ontologies. Batres et al. [105] describe their effort to develop an ontology based on ISO 15926. They are based upon the concept of supporting the development of domain ontologies. These are upper ontologies which define top-level concepts such as physical objects, activities, mereological and topological relations from which more specific classes and relations can be defined. Smith [106] criticises the effort concerning its ontological applicability from the philosophical point of view. The author supports the idea that the way of developing the model should change in order to be developed into an ontology. Leal [107] explains the reasons why ISO 15926 has been developed, its relationship to the STEP (ISO 10303) standard and provides an overview of its functionalities including the "4D approach". The author also explains how this ISO is described through first order logic and its ability to be converted to an ontology. Hakkarainen [108] carried out a study on mapping ISO 15926-2 with OWL-DL. Three alternative semantic transformation approaches were developed and two were tested and analysed in their ability to preserve semantics. Transformation Method one results in a seemingly direct representation of ISO 15926 in OWL, and enables full specifications. Transformation Method two takes more advantage of the language constructs in OWL and is most appropriate if the transformation is performed in order to take advantage of the reasoning provided by OWL and therefore, providing functionality not natively present in ISO 15926.

Furthermore, Fiorentini et al. [109] translated the NIST's core product model and proposed an ontology for the Open Assembly Model (OAM) implementing several OWL capabilities. Also, Fiorentini et al. [110] based on the work developed for the OAM demonstrated how to implement ontologies into existing product models. Tektonidis et al. [111] with project ONAR developed Semantic Web technologies for application integration. Lee et al. [112] developed a model for sharing product knowledge of the Beginning Of Life (BOL) on the web. Brandt et al. [113] apply ontologies on to knowledge management in design processes

with the aim of making knowledge of the design processes understandable and accessible to all engineers. Zhang and Yin [114] make an attempt of applying ontologies in a multi-agent distributed design environment. Suh et al. [115] use ontologies for interoperability and present a model for using data of the entire life of the products as an input for the design and production of new products. Chang et al. [116] are focusing in design and therefore, in the BOL. Their model is developed in order to guide designers in the design process of metal parts. Its aim is to make recommendations to the designer towards making parts which will be developed using friction stir welding, a solid-state joining technique. Still, in this work the implementation of ontology advantages remains a future perspective. Jun et al. [117] have developed an ontology model for product lifecycle metadata to Closed-Loop PLM. Aziz et al. [118] (Open standard, open source and peer-to-peer tools and methods for collaborative product development) have developed an ontological management methodology to overcome limitations of current PLM implementations.

The main characteristic of these works is that the notion of ontology varies. Usually it is used to express a UML model which is only human understandable, but in practice the models are limited to represent the structure of a database. In very few works ([109], [110]) DLs are used, which make the computer understand the meaning of each class, attribute and relationship. However, still there is limited use of the advantages they provide in favour of improving current PLM systems. The vast majority of the ontology applications and models mainly focus on product models in BOL and from the ontology perspective they are limited on inferring instances, without exploiting the full potential of the DLs.

3.3 State of the art in Time Management

Time is the only fundamental dimension which exists along the entire life of an individual (including materials and physical products) and it affects all individuals and their qualities. Individuals existed in the past and will exist in the future no matter if they only currently exist in our model. Time is considered as the fourth dimension in several sciences and Sider in his work “Four Dimensionalism” [119] provides a good description of the 4D paradigm. Individuals exist in a manifold of 4 dimensions, three space and one time and therefore, they have both temporal parts and spatial parts. Time in this context is used with its generic meaning as a dimension.

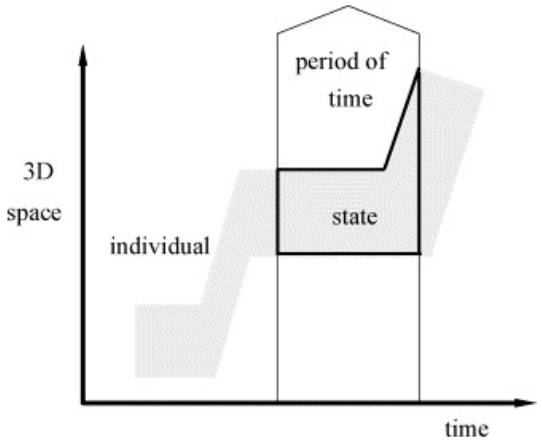

Figure 7: An object (possible individual) and it temporal part (state) according to ISO-15926.

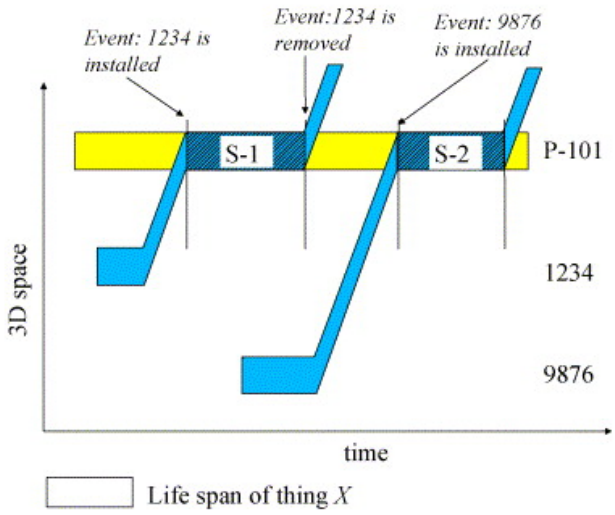

Figure 8: A pump and its temporal parts 1234 and 9876, according to ISO-15926.

3.3.1 Time Concepts

The importance of time in the field of engineering has been noted in several works. In part 2 of the ISO 15926 [120] there is a use of time as the fourth dimension. It is used to describe: actual individuals (including physical objects) which actually exist, or have actually existed in the past; possible individuals which possibly have existed in the past, and may possibly exist in the future; and individuals which are hypothetical having no existence in the past or future. West [121] describes the need for tracking the state and status of an individual along time (including to which physical product the individual

belongs or is part of). The author also describes how this need inspired the development of ISO 15926-2. As a solution the author recommends the use of International Standards combined with ontologies. Batres et al. [105] describe their effort to develop an ontology based on ISO 15926, analyse part 2 and briefly show how time is used to demonstrate the continuity of functionality of the parts. This is shown in detail in Figure 7 and in Figure 8. Roddick et al. [122] discuss the significance of time in spatio-temporal data mining systems and describe the need for future research that has to be carried out. Zhang et al. [123] suggest a model for the lifecycle of the infrastructure system facilitating the spatio-temporal data. Roddick et al. [124] on their bibliography research point out the value of investigating temporal, spatial and spatio-temporal data for future knowledge generation. In PROMISE [91] semantic object model the continuity of the history of each part over time is also considered important and it is stored in the "part of" class. Jun et al. [117] developed a time-centric ontology model for product lifecycle meta-data for supporting the concept of Closed-Loop PLM. Finally, very important work towards describing how to deal with time handling and synchronisation issues in computer distributed systems has been performed by Tanenbaum et al. [125] in the book "*Distibuted Systems: Principles and Paradigms*". The authors among other issues provide detailed approaches on achieving system synchronisation on distributed object-based systems, distributed file systems, distributed web-based systems and distributed coordination-based systems.

3.3.2 Time Management Approaches

In the "four dimensional models", time attributes are included in a separate part of the model (Date_Time class) to which other parts (not necessarily all parts) are associated through relationships as shown in Figure 9. Such systems become complex due to the large number of relationships between Date_Time class and the other parts of the model. Furthermore, time data is not being collected about the whole system for the whole life cycle. The latter occurs either in cases where not all parts are connected to the Date_Time class or in cases where the architecture of the system changes along the life cycle and the relationships to the Date_Time class are changed.

In a significant number of models which do not claim to be four dimensional time attributes exist in the parts of the model where time was considered necessary by the model designer.

Most commonly time attributes are in the parts of the model describing the "process", the "activity" (having starting time, finishing time and duration) and the "event" (having points in time or time stamps). An example is shown in Figure 10. These types of models face data integration and interoperability issues and are mostly developed to describe specific applications. Moreover, time data do not cover the whole system which has consequences in later stages, when time elements are required (i.e. feedback from maintenance to design) but they were not collected and therefore, are not available.

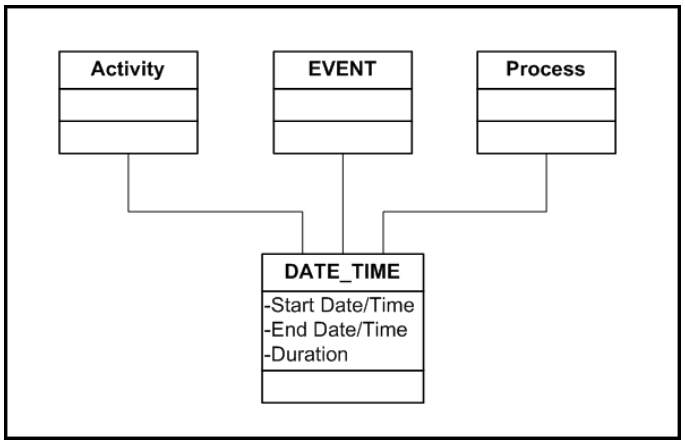

Figure 9: Schematic representation of a four dimensional model.

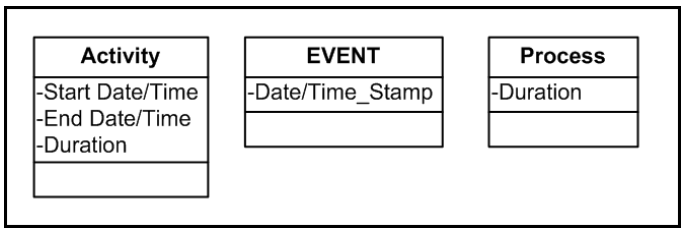

Figure 10: Schematic representation of a model with time/date attributes distributed in various classes.

In today's systems although time attributes exist in various parts of the systems, there are no systems which are based on time. Time has some qualities which make it special among all the attributes. It is the sole fundamental element which exists along the entire life cycle of all individuals. Furthermore, time is simple, comprehensive and objective and therefore, application independent. In this way, time may be used to be the connecting element of various systems and models. These qualities of time characteristics were the initiative to select time as the basis for developing a methodology for managing time in PLM/ALM

systems (the "*Duration of Time*" concept, see section 4.5). This methodology introduces the idea of seeing all aspects and elements of a model as parts of time and it provides flexibility, application independence and simplicity. In this way time exists naturally in everything. This holds also in our everyday life but sometimes we do not really realise it since our view is too "narrow" to see the big picture and we focus only on the small part which affects us directly considering time with its generic meaning as stable.

3.4 Conclusion

In this chapter, firstly, are presented the MIMOSA and ISO-15926 standards as well as the Semantic Object Model of PROMISE. In this way the reader may acquire an overview of the current models which are used in various sectors of PLM. Secondly, we have presented the utilisation of ontology based IT methods and tools in bio-informatics and other research fields. We also have presented the scope of these implementations towards improving or adding new solutions and fulfilling requirements of these scientific fields. Furthermore, we have presented implementations of IT methods and tools in models focusing in parts of PLM. Our conclusion is that the implementation level of the new methods and tools in PLM, still, is less than in other research fields. One of the possible causes for this situation is the lack of a methodology of using these technologies in the field PLM efficiently.

The importance of time in PLM has been pointed out by several developers and in this section we have presented a number of ways that time is treated in today's models. There exist two types of model architectures regarding time: the four-dimensional models in which all parts that need time properties are related with a class that contains the time properties; and the models in which there are time properties in each class that is required. Our claim is that time in its generic meaning is under-exploited in both types of models. This claim is based on the fact that time is a fundamental element which exists naturally in all the parts of the PLM systems and that the notion of time is objective and is easily understandable since it exists in our everyday life. Therefore, innovative ideas of time treatment are necessary in order to change the philosophy of the model architecture and to provide new services for the Closed-Loop PLM. To this end it would be of significant value to develop a method for system modelling which is lean and utilises the advantages of time.

4

Ontology Development for Closed-Loop PLM

This chapter describes a number of developments performed in this work. Firstly, the architecture of the system is presented in order to describe in detail how we combined and utilised a number of IT methods and tools. For representing data we have chosen to use OWL-DL and as an editor for developing the ontology we selected Protégé [41], which provides the Protégé-OWL plug-in. Secondly, the step-by-step development of the ontology model is presented. The model that is our basis is the SOM developed in PROMISE which is described in section 3.1.2 (Figure 6). The SOM was developed using the UML class diagrams which was static and it did not facilitate functionalities such as loading data on to the model and performing reasoning on the model. Therefore, in this part is presented the transition from the initial UML model to an OWL-DL model and the new opportunities created due to this transition. In order to make the model capable of exploiting these opportunities, several changes were performed on the classes as well as on the object and datatype properties of the model. The third part describes what actually happens in the model during ontology merging and provides a methodology for making the ontology model ready to be merged with variations of the initial ontology model. The fourth part, describes the extension of the model to support semantic maintenance. To achieve this, the model developed in the second part (ontology model derived from the SOM) was extended with several classes representing concepts and properties. The fifth part describes the new developed "*Duration of Time*" concept. Our research aim is to provide a concept based on time which may be implemented and function efficiently using current technologies in the current PLM/ALM systems. Finally, we provide a generic

implementation methodology for implementing the system architecture and the "*Duration of Time*" concept in existing PLM systems.

4.1 System Architecture Description and Functionality

In accordance with the ontology tools analysis in chapter 2 and the modelling requirements described in chapter 3 we have selected to work with OWL-DL. The decision of using OWL-DL was inspired by the reasoning capabilities of the DL which provide consistency checking, subsumption, realisation and retrieval [26]. According to Ian Horrocks [126], *"DLs are a family of logic-based knowledge representation formalisms creating an object oriented model"*. Instances, classes (representing human concepts) and relationships among classes (representing roles of the concepts in real life) are the building blocks used by the ontology to describe the domain. An ontology consisting of these terms and being developed in DL is extensible since DL allows class descriptions to be composed from classes and relationships, expressing that a class is a sub-class of or equivalent to another class. In addition, DL supports reasoning by supporting the designer of the model with information about inconsistencies, synonyms and classification relationships implied from the rules. The latter are used by the DL-reasoner to update the class-hierarchy.

The selected editor to build the ontologies is Protégé-OWL. It fully supports the OWL-DL and it contains a number of useful plug-ins to treat data as well as a built-in version of the Pellet DL-reasoner. Moreover, it is well supported and open source and therefore, our partners may easily process and use our work.

The DL-reasoner is very important part of the system architecture since it provides the reasoning on the model. In our work we selected to use Pellet for mainly two reasons: Pellet in our use cases proved to be as efficient as Racer and Fact++; and it also has the advantage (version 1.5.2) of being able to reason on SWRL rules. Therefore, we selected Pellet as our DL-reasoner.

4.1.1 System Description

The different IT methods and tools utilised in this work are shown in Figure 11. The big rectangle represents the Protégé-OWL editor software. The circles in this rectangle represent the different tools which are implemented in the software and are used in this

work. The other two smaller rectangles show the data stored in spreadsheets (excel 2003) and CSV (Comma Separated Values) files (data collected from sensors and industrial partners was provided in these formats). The thin continuous line arrows inside the rectangle of the Protégé-OWL show the information flow inside the software. The thick dashed arrows connecting the Protégé-OWL with the spreadsheet and the CSV file show the information flow (export/import) between the plug-ins and external files. Also, data is imported from CSV files to spreadsheets. The arrow head (for all the arrows) shows the direction of the information flow: i.e. the class-hierarchy, properties and instances existing in the OWL-DL are input (and are read by) for the Queries Tab. It should be noted that the description of the elements contained in this figure represents how we used these tools and does not claim to be exhaustive about the capabilities of the tools. The tools contained in the circles are:

OWL-DL: in this part the model is built using OWL-DL. It contains all the classes, instances, properties and restrictions (DL- rules) of the model.

DataMaster tab**:** this part is used to read instances and their data from spreadsheets. Then, it is used to load the instances with their data to the OWL-DL model.

DL-Reasoner (Pellet): Pellet is included in the Protégé-OWL and reads/understands the DL rules (semantics) of the model. It is used to perform logical queries on the OWL-DL model such as: is the class-hierarchy consistent?; which is the right logical position of each class in the model?; or to which class(-es) each instance belongs?. It checks the class-hierarchy for its consistency; it re-classifies the classes according to their meaning; and finds equivalencies between them. Moreover, instances are inferred in their logical position under the classes.

Queries Tab: it is used to perform database-like queries. It may be used to make queries on the OWL but it cannot read the DL rules and therefore, the queries are limited i.e. it cannot understand that a property is transitive and hence, it cannot understand the representation of the sequence (also called "inheritance") of semantics. Its advantage is that it produces results very fast even for large ontologies (it returns results much faster than the SQWRL for the same query). It provides the possibility of exporting the results to excel spreadsheets.

SWRL: (which contains the SQWRL and the bridge with the Jess rule engine). The rules written in this language are used to extract knowledge about the OWL- model. For more details on SWRL see section 2.2.4. Still, it cannot infer all the knowledge as compared with the DL-reasoner.

SQWRL: (stands for Semantic Query-Enhanced Web Rule Language [45] and is contained in the SWRL tab [42]) it is used to perform database-like queries and has the limitation that it cannot read the DL rules and therefore, the queries are limited i.e. it also cannot understand inheritance. This tool is used when non-logical queries need to be performed i.e. check if values are within thresholds, calculate duration, sort the events according to when they occurred (before or after a certain date), etc. It also provides the possibility of exporting the results as CSV (comma separated values) file.

Jess Rule Engine: it runs in the SWRL tab through the SWRL Jess Bridge [42] and reads a number of the basic OWL axioms. Its main use in our case is to read the SWRL rules and the OWL model, infer knowledge according to the SWRL rules and return the knowledge back to the OWL model. This might create consistency problems. According to the protégé documentation (in the SWRLJess tab:) *"A significant limitation of the current bridge is that it does not represent all OWL axioms when transferring knowledge from an OWL ontology to Jess. The exceptions are the basic class, property and individual axioms. As a result, the Jess inferencing mechanisms do not know about the remaining OWL axioms. To ensure consistency, a reasoner should be run on an OWL knowledge base before SWRL rules and OWL knowledge are transferred to Jess. Also, if inferred knowledge from Jess is inserted back into an OWL ontology, a reasoner should again be executed to ensure that the new knowledge does not conflict with OWL axioms in that knowledge base"*.

4.1.2 System Functionality

The functionality of the system is as follows. Firstly, the model is developed in the Protégé-OWL editor as an OWL-DL ontology containing the classes, object properties (relationships between classes) and datatype properties (attributes). Then, DL rules are added to the classes as restrictions to define them according to requirements. These definitions of each class are machine understandable. In the next step instances are loaded into the model either manually or from spreadsheets through the DataMaster.

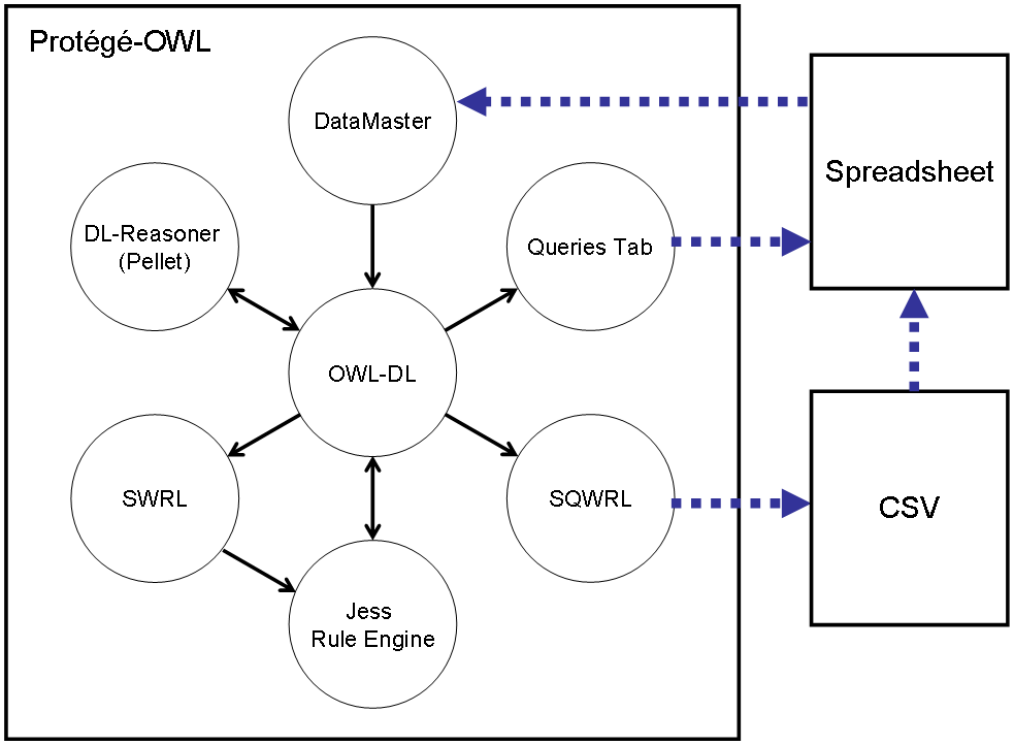

Figure 11: System Architecture

Furthermore, in the Figure 11 there is a triangle shaped loop of information flow from OWL-DL, to SWRL, from SWRL and OWL-DL to Jess Rule Engine, and finally back to OWL-DL. This loop describes a process which consists of the following steps: in the first step the SWRL rules are created; in the second step the SWRL rules and the OWL knowledge are transferred to Jess (two input arrows to Jess in Figure 11 one from SWRL and one OWL-DL); in the third step the Jess rule engine is executed to infer knowledge (according to SWRL rules and the OWL knowledge it has read); and finally the inferred knowledge from Jess is transferred into the OWL ontology. Since Jess does not read all OWL axioms, Pellet DL-reasoner should be executed at this point to ensure that the OWL model is still consistent.

This step by step process was used to infer instances of selected parts of the OWL model under the right classes and then return this knowledge back to OWL. This part specific return of instances is not possible to be performed by the DL-reasoner, since the DL-reasoner firstly, needs to read the whole model and check it for its consistency and then, to continue to perform the inference. The loop was also used when instances and data were

introduced using spreadsheets. In this case, the problem was that each row of the spreadsheet is translated into OWL as an instance; each column is translated as datatype property; and the values of all datatype properties per instance are contained in each cell corresponding to the instance row and datatype property column. The values are in a format of an xml schema datatype i.e. string, date, etc.

In many cases the form of data imported from spreadsheets needed to be treated in order to be in a certain required form i.e. some elements imported as datatype properties are required to be in the model as object properties. For example, the desired situation that a product (instance) "car1" is related with the physical product group "car", just after importing the instances from the spreadsheet was declared as: instance (of product class) "car1" has a datatype property physical product group with the value "car" as a string. Therefore, appropriate SWRL rules were created and applied to treat the imported data. Example of such rules can be found in Appendix D.

Different ways for querying the model apart from the above mentioned loop are: the DL-reasoner, the Queries tab and the SQWRL. Each one of these tools is used under different circumstances and it depends on the type and the nature of the query. This is briefly described in this section.

The DL-reasoner has the advantage that it can read all the OWL-DL axioms and rules. It is the only tool which checks the ontology model for its consistency. However, its drawback is that it is applied on the whole model which for large ontologies makes the answer process slow. The results of the queries regarding the classification of classes (including equivalencies) and the inference of instances may be saved as a separate OWL model. The possibility of selectively asserting/returning the knowledge one by one back to the OWL-DL model may be performed only for the classification results (class by class). In the case that we need to return instances selectively to the model we have to use the previously described loop with the SWRL and Jess.

The Queries tab is used to apply database-like queries which are applied on selected parts of the model. These queries are strictly non-numeric i.e. impossible to ask for finding instances that for a datatype property (i.e. salary) they have a value greater than or smaller than a certain value (i.e. 1 000 Euros); they are applied on classes and properties, and they

return the list of the instances which fulfil the queries. The results cannot be returned back to the OWL model but they are easily exported to excel spreadsheets. Then, if necessary the data from the spreadsheets may be imported to the OWL model through DataMaster.

The SQWRL is also used to apply database-like queries which are applied on selected parts of the model. These queries have the extra advantage of being also numeric (i.e. it is possible to ask for greater than or smaller than queries); they are applied on classes and properties, and they return the list of the instances as well as their datatype and object properties which fulfil the queries. The results cannot be returned back to the OWL model but are easily exported as CSV files. Then, if necessary the CSV files are imported to excel spreadsheets and they may be imported to the OWL model through DataMaster.

The OWL-model in the framework of this architecture is very flexible and changes may be performed whenever required: on the class-hierachy, on the classes, on the instances, on the properties and on the DL rules. In chapter 4 only a small part of the architecture was used since the ontology models are being developed in OWL-DL and they are loaded on the Protégé-OWL editor. Excessive use of the whole system architecture has been performed in the case studies in chapter 5.

4.2 Ontology-Based Model for Closed-Loop Product Lifecycle Management

This work describes the process and various details of developing the SOM described in section 3.1.2 into an ontology using OWL-DL. The model was slightly modified to facilitate several of the OWL-DL capabilities, always maintaining previously achieved characteristics. The tool we selected for developing the ontology is that of Protégé, which provides the protégé-OWL plug-in. In the rest of this work, the naming conventions used are the same as in section 3.1.2: names of classes are written in boldface and capitalized/lower case Arial (i.e. **Product_BOL_Supply, Product_MOL, Product_EOL,** etc). Names of attributes and associations (also called relationships) are capitalised /lower case Courier New (i.e. `isDesigned`) while names of instances are in italics Arial (i.e. *Passenger_Vehicle_1*).

The primary aim was to give to the ontology model the functionalities implemented in the SOM and at the same time to keep it lean. The initial model was slightly modified in order

to be transformed into an ontology. The ontology developing process is described in the following paragraphs.

4.2.1 General Alternations of the SOM of PROMISE

At the beginning we created an ontology containing all the classes of the SOM shown in Figure 1, with some alternations in the class-hierarchy. The alternations were:

- All classes were first added to the ontology as sub-class of owl:Thing.
- The structure of generalisations (class, sub-class) of the UML model was kept unchanged with only one exception which is the generalisation of **As_Designed_Product** to **Physical_Product**. This generalisation is transformed into an association and it is expressed through the functional object property `isDesigned` with domain **Physical_Product** and range **As_Designed_Product** and its inverse, which is inverse functional, `hasDefined`.
- The compositions between the **Physical_Product** class and the classes **Product_BOL_Supply, Product_MOL** and **Product_EOL**, do no longer exist. A new object property has been created associating **Life_Cycle_Phase** class with the class **Physical_Product** and it is the object property `Life_Cycle_Phase2Physical_Product`. Furthermore, the three classes of the composition are sub-classes of **Life_Cycle_Phase** class. Thus, the three classes are associated with the **Physical_Product** class indirectly through **Life_Cycle_Phase** class.
- The composition between the **Physical_Product** class and the class of **ID_Information** (**ID_Info** in the UML model) does no longer exist. They are associated through the object property `ID_Information2Physical_Product` and its inverse `Physical_Product2ID_Information`. Each instance of **Physical_Product** can be related only to one instance of **ID_Information** and vice versa. Thus, for example a **Physical_Product** instance A is related exactly to **ID_Information** instance B and **ID_Information** instance B is related exactly to **Physical_Product** instance A.

The compositions between the **ID_Information** class and the classes **Information_Provider** and **URI**, do no longer exist. They are associated through the functional object properties `ID_Information2Information_Provider` and `ID_Information2URI` respectively. Their inverse properties are `Information_Provider2ID_Information` and `URI2ID_Information` respectively and they are inverse functional.

At this stage all the structure of the classes has been loaded on the Protégé-OWL editor. This is the class-hierarchy of the ontology model.

4.2.2 Transformation of Attribute Properties

The next step is the definition of the attributes of the classes. When an instance of a class is created, it may have values for each attribute of the class. In OWL there are two types of properties; the object properties expressing relationships and the datatype properties expressing attributes. Datatype properties are equivalent to UML attribute properties. Most of the datatype properties of the new model are the same as described in SOM. However, several changes were performed due to mainly the use and the expressivity of the OWL-DL:

- Primitive datatype properties `*_Name` or `Name` have been eliminated wherever possible. Their functionality is being fulfilled by Protégé-OWL internal names (hidden property ':NAME') of the individuals. This has been preferred to rdfs:label since the individuals are named always in English and this property can be easier inserted to the restrictions widget. Moreover, rdfs:labels are let free to be used exclusively by the user for the requirements of each application. In OWL-DL each individual when created has to have a name which is stored in the property ':NAME' and it is unique for the entire ontology.
- Primitive datatype `Parent` of the class **Physical_Product** has been omitted. This functionality is acquired through the object property `hasParent` with domain **Physical_Product** class and range **Physical_Product** class and its inverse `isParentOf`.

- Primitive datatype Parent of the class **As_Designed_Product** has been omitted. This functionality is acquired through the object property `isDesigned` and `hasDefined` described in paragraph 4.2.1.
- Alternations have been conducted at the primitive datatype property pairs of `Product_State_Set` and of `Product_State_Set_Definition`, `Resource_State_Set` and `Resource_State_Set_Definition` expressing the allowed values for the state (state set) and the chosen value out of the set (state set definition) of the **Product** and **Resource** classes respectively. They have been translated into one datatype property per pair with defined allowed values (`Product_State` and `Resource_State`). The allowed values should be defined in advance, before populating the model, according to the requirements of each application.
- A datatype `Product_Complexity` with allowed values "simple" and "complex" has been added in **Physical_Product** class. A physical product is "complex" when it is composed of more than one part or sub-systems i.e. a passenger vehicle consists of an engine, wheels, gearbox, battery etc., and it is "simple" when the product is for the current model the highest level of detail and it is composed of one part. Hence, it does not have any sub-systems or sub-products.
- In **Life_Cycle_Phase** class we added the attributes `Starting_Date_Time` and `Finishing_Date_Time`. A product instance of the **Product** class is always related to one or more instances of the **Life_Cycle_Phase** class. This is done in order to define the value of "when" the product has entered or exited a lifecycle phase. The use of this is to show the duration that a product was a part of another product and was used in a certain way.

At this level all properties describing the properties of SOM have been added to the classes of the ontology developed in paragraph 4.2.1.

4.2.3 Transformation of Associations

In OWL there are no associations like in UML. Object properties of OWL are used in order to represent them. These properties are used to relate the different classes of the model. The process followed is:

- Un-named binary associations have been expressed through OWL object properties and they are named according to domain-range policy, domain2range i.e. `Field_Data2Document`.
- Named binary associations have been kept unchanged.
- A new object property has been created associating **Life_Cycle_Phase** class with the class **Physical_Product** and is the object property `Life_Cycle_Phase2Physical_Product` and its inverse.

In this manner, all the relationships between the classes of the ontology model have been created. The ontology model developed up to this stage is complete and it facilitates all the functionalities of the SOM (Figure 12).

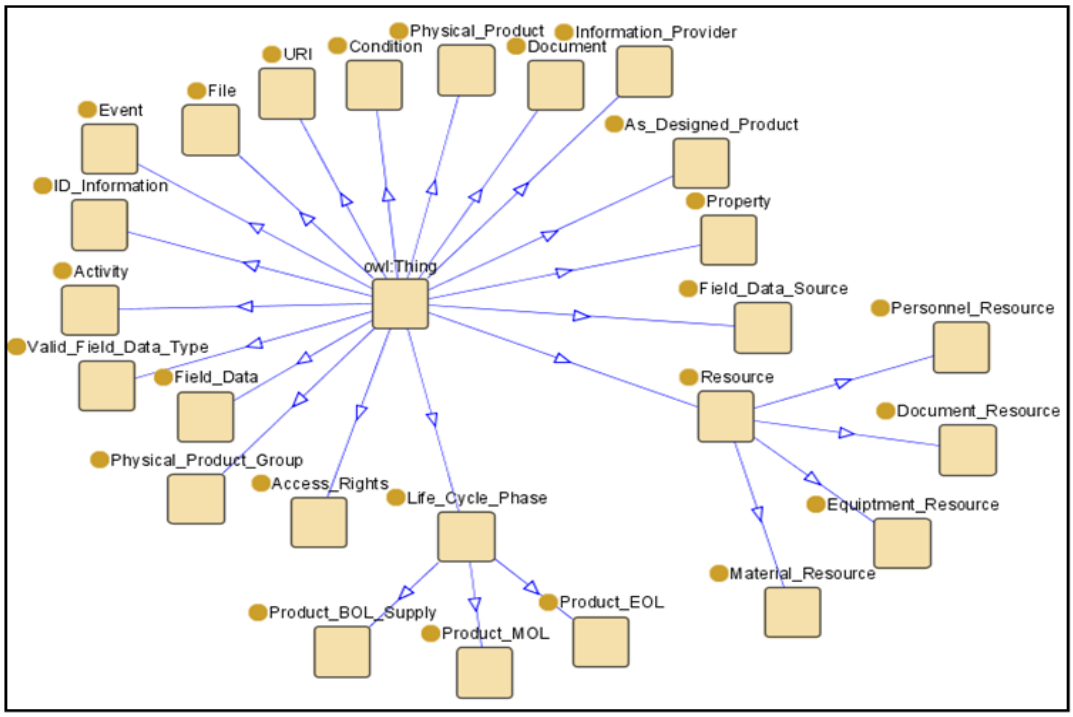

Figure 12: Structure of the class-hierarchy of the PROMISE PDKM SOM.

4.2.4 Alternations for Supporting Additional Functionalities

The SOM was designed to be a framework for meta-data and to be used for one product and its components. After using and testing the model we concluded that some more changes could be done in the OWL version to improve the model. The use of an ontology makes the model dynamic and allows to record, store and process data-information about a number of systems in a single source. Moreover, there are alternations based on the higher description ability of the new tools.

Alternations in classes and properties:

- In the UML model the class **Part_Of** describes the several parts (single physical products) a complex physical product may consist of and "for how long" each of these parts is part of the complex physical product. In order to express in OWL the **Part_Of** class and eliminate it, we studied the W3C recommendation "Simple part-whole relations in OWL ontologies" [127]. The suggested structure is not suitable for our model for mainly two reasons:
  - It is not possible to have a Physical Product, which is normally `partOf` a Physical Product, without being `partOf_directly` of a Physical Product. This means, for example, that this representation does not allow the model to contain a motor which does not belong to a car. Therefore, simple statements like "a motor is in stock waiting to be installed" cannot be described.
  - While adding existential restrictions, incorrect statements for our generic point of view such as all motors are car parts are inferred. However, "not all motors are for cars, some are for trains, boats, etc."

  Finally, the UML class **Part_Of** (Figure 13) is expressed through the object property `hasParent` and its inverse `isParentOf` as shown in Figure 14. The concept "for how long" is expressed through the multiple instances of **Product_MOL** class a physical product may be related to. For this reason the datatype properties `Starting_Date_Time` and `Finishing_Date_Time` were added to the **Life_Cycle_Phase** class. The `hasParent`, `isParentOf` object properties are transitive to cover the cases where we have more than one level of inheritance and

therefore complexity. Thus, they will function like a chain and relate all the related instances if necessary. For example if A `isParentOf` B and B `isParentOf` C, then A `isParentOf` C will be assumed.

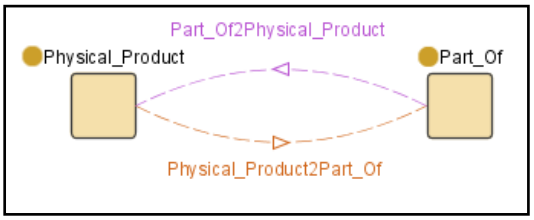

Figure 13: Physical Product and Part Of before changes.

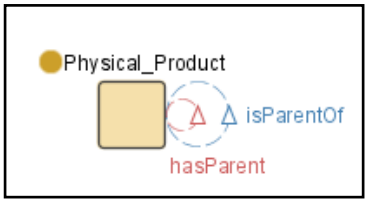

Figure 14: Physical Product after changes.

- The ontology model allows the recording of information about a number of systems in a single source. Thus, while populating the model with individuals representing several different products, a problem occurred with the non-existence of a relationship between **Physical_Product** and **Field_Data**:
  - With the existing structure we have a bottleneck effect when we have multiple physical products of the same type. The two classes will be related to the same **Valid_Field_Data_Type** instance. For instance *Motor_1* and *Motor_2* will be connected to **Valid_Field_Data_Type** instance *Motor_Temperature* with `Measuring_Unit` "Celsius" which is connected to several **Field_Data** instances representing different measurements at different times and different physical products such as *Motor_Temperature_1*, *Motor_Temperature_2*, *Motor_Temperature_3*, etc.
  - This causes loss of information because we cannot relate the **Field_Data** instances to a **Physical_Product** instance. Hence, in the previous example we

cannot know whether *Motor_Temperature_3* is referring to *Motor_1* or *Motor_2*.

For solving this problem we associated **Field_Data** class directly with the **Physical_Product** class with the object property `Field_Data2Physical_Product` and its inverse `Physical_Product2Field_Data` (Figure 15 and Figure 16).

- The same problem like the one mentioned above occurred between **Field_Data_Source** and **Physical_Product**. In this case, with the given structure it was not possible to identify which instance of **Physical_Product** was associated with each instance of **Field_Data_Source**.

  Solution chosen: We associated **Field_Data_Source** class directly with the **Physical_Product** class with the object property `Field_Data_Source2Physical_Product` and its inverse `Physical_Product2Field_Data_Source` (Figure 15 and Figure 16).

- Similar problem appeared between **Field_Data_Source** and **Field_Data**. In this case, with the given structure it was not possible to identify which instance of **Field_Data** was associated with each instance of **Field_Data_Source**. This problem is partly solved through the four new object properties created in the two previous paragraphs. The added properties provide solution only in the case that each **Physical_Product** instance is related to only one **Field_Data_Source** instance. However, this is a very rare case since it means that the product has only one sensor. Therefore, we were obliged to provide a solution.

  Solution chosen: We associated **Field_Data_Source** class directly with the **Field_Data** class with the functional object property `Field_Data_Source2Field_Data` and its inverse (inverse functional) `Field_Data2Field_Data_Source` (Figure 17).

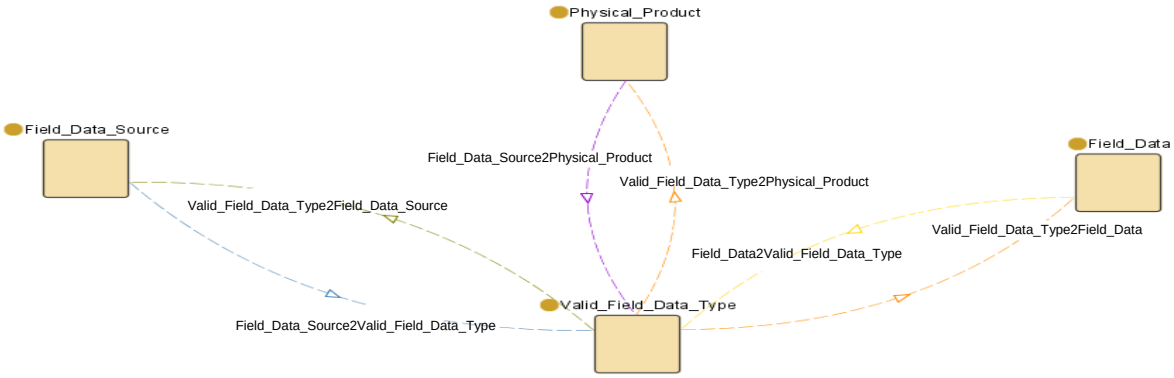

Figure 15: Relationship view for Physical Product class before alternations.

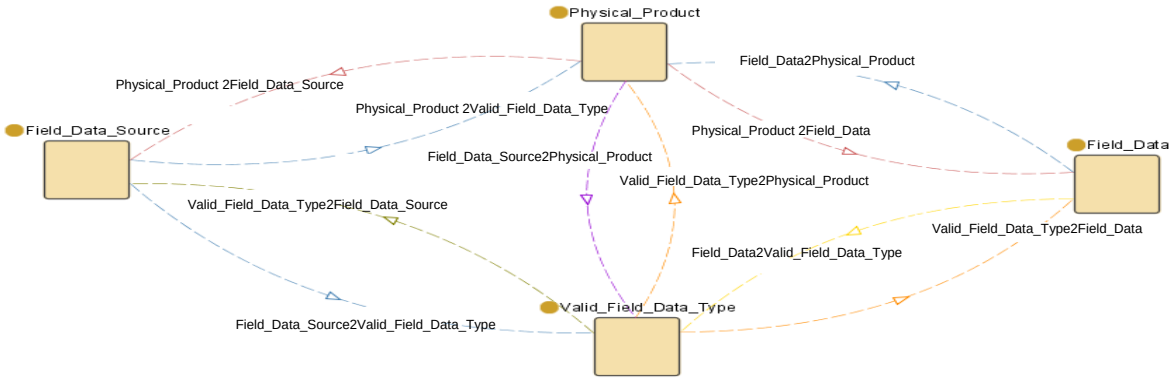

Figure 16: Relationship view for Physical Product class after alternations.

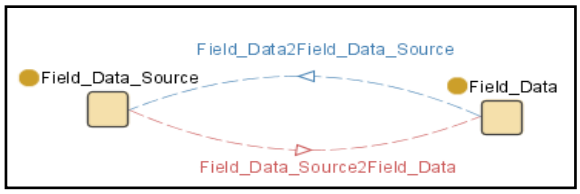

Figure 17: Relationship view for classes Field Data Source and Field Data after alternations.

The developed ontology model is dynamic (changes can be made on the fly), it can store data about multiple products on a single source (it allows to record, store and process data-information about a number of systems in a single ontology source) and it has higher description ability (allows the user to see the multiple levels of inheritance). The developed model is shown in Figure 18. See Appendix A for a full list of object and datatype properties.

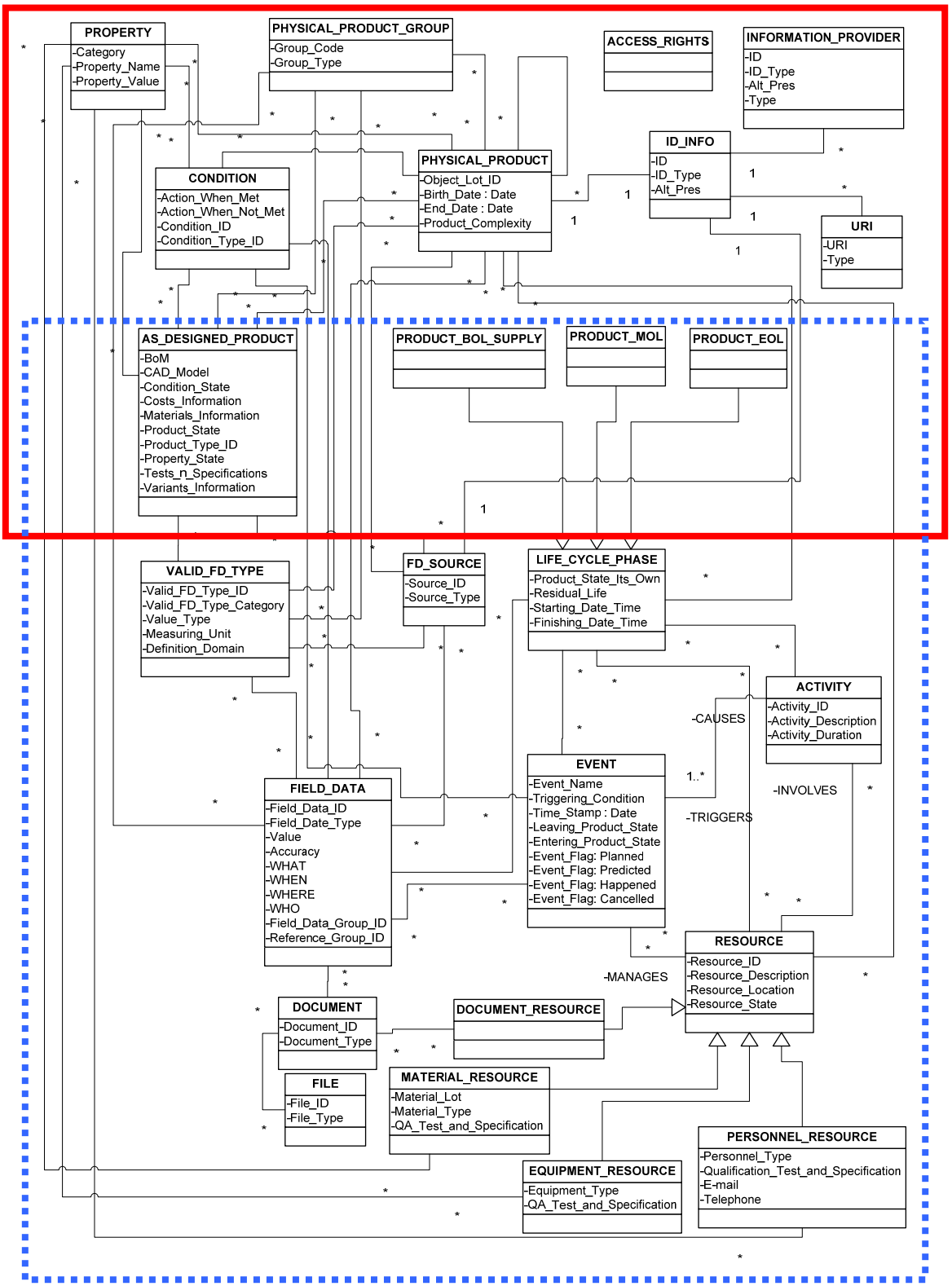

Figure 18: Complete UML schema of the ontology model.

4.3 Towards an Ontology Merging friendly system

Ontology models theoretically are developed to formally describe domains of knowledge. Their aim is to be re-used and to provide a common understanding among different partners. In practice, several ontologies are developed to describe the same domain using different semantics and therefore, there is a lack of interoperability and the creation of burdens for performing ontology re-use. The concept developed and described in this work is that such interoperability issues could be tackled with the appropriate utilisation of OWL, description logics and inference engines.

An ontology model consists of a hierarchy of classes which are related to each other with object properties. The classes also contain datatype properties. Then the classes are populated with instances which contain data loaded on the properties fields.

In OWL every developed ontology is related to one unique Uniform Resource Identifier (URI). Therefore, we have the axiom:

$$\forall \text{ontology } O \text{ there is a unique URI } U$$

In practice this means that if for example one ontology has a URI *U* where:

$$U = \text{http://www.owl-ontologies.com/Ontology1202459344.owl}$$

A class of this ontology named "Product" will have as a full name:

http://www.owl-ontologies.com/Ontology1202459344.owl#Product

If there is another ontology with a different URI *U'* where:

$$U' = \text{http://www.owl-ontologies.com/Ontology1202459355.owl}$$

And if this ontology has also a class named "Product" the full name of this class is:

http://www.owl-ontologies.com/Ontology1202459355.owl#Product

This full name policy is respected for all different elements of each ontology. In this example the two classes named "Product" have different full names and this is machine-understandable. Therefore, when we merge two or more ontologies together, this characteristic allows OWL (including the Ontology Editor and the inference engine) to assume these two elements are different and are parts of different ontologies. In this way

the ontology editor verifies which elements (classes, properties, instances, etc.) belong to a specific ontology.

4.3.1 Merging One or More Ontologies

The concept is that experts of the OEM develop one ontology model to facilitate the information model for the data generated during the lifecycle of a product or asset. This model is generic and is required to be flexible and extensible according to the user's needs. Moreover, the OEM develops a method for extending the model by using DL rules. Then, copies of the model together with the method for using DL rules are provided to the partners. The method of using DL rules allows the OEM not to lose interoperability and data integration among the different copies of the model. The partners are able to extend the model according to their needs (following the method provided) and populate it with data. In the next step the OEM collects the different copies and merges them under one single ontology model. Thus, the final single model contains all the different elements of the copies without duplicates. This includes classes, object and datatype properties, instances and DL rules.

Initially the OEM has an ontology *O* (defined by the URI *U*) with a set *A* of classes, DL rules, object properties, datatype properties and instances with data, hence *A* = {class_A1, class_A2, etc., DL rule_A1, DL rule_A2, etc., object property_A1, object property_A2, etc., datatype property_A1, datatype property_A2, etc., instance_A1, instance_A2, etc.}. All these elements of set *A* have the same URI *U* of the ontology *O* which define their full names.

Then, the OEM makes copies of this ontology and distributes them to its partners for use together with simple user's instructions (method) of how to extend the model using DL rules. The partners extend their copies according to their needs in all types of aspects: classes, DL rules, object properties, datatype properties. They also create instances to load the data generated during the lifecycle of the products. At some point the OEM collects all the distributed copies of the ontology. Each copy has the initial set of elements *A* plus the extra elements which were loaded to it. The total of the extra elements of all the copies are a set *B* where *B* = {class_B1, class_B2, etc., DL rule_B1, DL rule_B2, etc., object property_B1, object property_B2, etc., datatype property_B1, datatype property_B2, etc.,

instance_B1, instance_B2, etc.}. All these elements of the different copies of the set *B* have the same URI *U*. When the OEM merges all the distributed copies of the ontology the total set of elements is described by the equation:

$$A \cup (A \cup B) \tag{1}$$

Where for sets *A* and *B* we have:

$\forall$ontology *O* there is a set of elements $A \in O$
$\forall A$ there may exist sets of:
Classes $Cl_i \in A$,
Individuals $In_i \in A$,
Datatype Properties $Dp_i \in A$,
Object Properties $Op_i \in A$,
Description Logic Rules $DL_i \in A$,
Where $i \in \mathbb{N}$

$\forall$ontology *O* there is a set of elements $B \in O$
$\forall B$ there may exist sets of:
Classes $Cl_j \in B$,
Individuals $In_j \in B$,
Datatype Properties $Dp_j \in B$,
Object Properties $Op_j \in B$,
Description Logic Rules $DL_j \in B$,
Where $j \in \mathbb{N}$

But, from the set theory we have the axiom:

$$A \cup (A \cup B) = A \cup B \tag{2}$$

The total number of elements in the final ontology is described by the set *C* which is:

$$C = A \cup B \tag{3}$$

Where for set *C* we have:

$\forall$ontology *O* there is a set of elements $C \in O$
$\forall C$ we have:
Classes $Cl_i \cup Cl_j = Cl_k$, where $Cl_k \in C$
Individuals $In_i \cup In_j = In_k$, where $In_k \in C$
Datatype Properties $Dp_i \cup Dp_j = Dp_k$, where $Dp_k \in C$
Object Properties $Op_i \cup Op_j = Op_k$, where $Op_k \in C$
Description Logic Rules $DL_i \cup DL_j = DL_k$, where $DL_k \in C$
Where $i, j, k \in \mathbb{N}$

Thus, the OEM has a final ontology (set *C*) which contains all the classes, DL rules, object properties, datatype properties and instances with data from all the copies of the initial ontology model.

The important and challenging task is to define a method for developing a model which will exploit the capabilities of DL rules and the DL-reasoner for passing automatically from equation (1) to equation (3). At the final ontology (set *C*) the model may contain duplicates of the classes, since different partners might have created classes with different names for facilitating the same concept. This might be due to different vocabulary or different language used by each partner. For example, one partner might have named a class "car" and another one might have used the word "vehicle", but actually these two classes might have been created to represent the same concept. A solution to this problem is provided by the use of DL rules and the DL-reasoner. It should be noted that it is assumed that all new classes have been created using DL rules as it is described at the method of the OEM. Thus, the DL-reasoner may be used to support the OEM to figure out the logical duplicates of concepts described in the final ontology. The DL-reasoner is applied on all the DL rules of the final ontology. When the DL-reasoner is executed all the benefits of OWL-DL hold and therefore, the result is:

- The class-hierarchy of the final model is checked for its consistency
- The classes are re-classified on the class-hierarchy according to the concept they represent
- Equivalencies among the classes are found and reported to the OEM
- All instances are categorised under the classes following the DL rules

The reasoner understands the DL rules, relationships etc. such as inheritance of the final ontology and apply them on the total number of the instances and classes. This is very important since the OEM is able to understand the content of all the copies of the initial ontology without the need to do any type of ontology mapping. With the use of DL rules the system is able to know the data that it has. A valuable usage of re-categorisation is that even if only one partner has "created" an important new and beneficial element, the whole system will benefit from it (see case study 1 in section 5.1.5). It should be noted that the above are valid in the general case where copies of multiple different ontologies are imported into one model.

4.3.2 Achieving Ontology Merging

While attempting to achieve ontology merging with OWL we faced several difficulties which are well documented in Appendix B. The solution selected to overcome these difficulties in order to achieve merging is described in the following simple steps:

Step 1

The OEM develops an ontology *O* which has URI *U*.

Step 2

The OEM makes copies of the initial ontology *O* which will be distributed to the partners, in a later stage.

Step 3

The OEM changes the URI of each copy to a unique URI combined with an ascending three digit number xxx in the end i.e. *U'*=http://www.owl-ontologies.com/Ontology1202459344_ Copy_001.owl, where xxx=001. Thus, all initial elements of each copy (classes, object and datatype properties, instances) keep their original URI *U* of the initial ontology *O*.

Step 4

The OEM sets as the default namespace of each copy the URI *U* of the initial ontology model *O*. Therefore, all the new elements added to each copy after this point, have the namespace of the initial ontology *O*.

Step 5

The OEM distributes the copies to its partners. All the new elements which are added by the partners to each copy have the namespace of the initial ontology *O*.

Following these steps the result is that we import ontology *O'* into *O* and then all the elements of both *O'* and *O* have the same URI *U* of ontology *O*. It should be noted that when the OEM will collect all the copies from its partners, the OEM can import all of them under one single source no matter which ontology is loaded first or the loading order in the ontology editor. In all cases all elements will be loaded and read by the editor. For more

details on how to deal with the technical difficulties and why we concluded to this solution please see Appendix B.

4.4 Extending the Ontology Model to provide Semantic Maintenance

In the framework of SMAC project (Semantic-maintenance and life cycle), supported by Interreg IV programme between France and Switzerland we have developed an ontology model for Semantic Maintenance (Rasovska et al. [128]) focusing on the collection and the analysis of the maintenance data. The developed model is extending the functionalities of the model developed in section 4.2 and its aim is to provide advanced maintenance methods which are beneficial in many ways such as to provide new services, to improve customer satisfaction, to acquire compliance with environmental friendly legislation, and to achieve higher product quality, higher performance and reliability. In order to develop our model, we combined the advantages of two previous developed models. In this project the model is applied on a lathe machine of the manufacturer TORNOS [129].

The first model we are based on is the model developed in section 4.2 (Figure 18) which is based on the PROMISE SOM. This model was made for supporting Closed-Loop Product Lifecycle Management. In this way the data and the information produced from the asset during its Middle Of Life (MOL) is collected and processed to be used as input for improvement of Beginning Of Life (BOL) activities (design, production), and End Of Life (EOL) activities (recycling, re-manufacturing, re-use, etc.). Thus, this model allows closing the information loop between the different phases of the lifecycle.

The second model used is a modified version of the semantic model of PROTEUS project (Bangemann et al. [130] and Rasovska et al. [131]), developed by Karray et al. [132]. The modifications where judged necessary in order to cover a wider field of maintenance. The PROTEUS platform supports vertical integration of applications in providing maintenance to remote industrial installations (Bangemann et al. [130]). Moreover, it provides description of the equipment through an ontology description, a generic architecture based on the Web services and models of heterogeneous components. The main aim of this platform is to provide an environment for integrating the execution of distributed processes which run on heterogeneous hardware/software platforms. As a communication tool the technology of Web services is used. The final UML ontological model of maintenance

consists of twelve parts [132] corresponding to both the structure of the enterprise information system and the maintenance process. These are: the monitoring management model; the site management model; the equipment expertise management model; the resource management model; the intervention management model which focuses on the maintenance intervention to remedy the equipment failure and is described by an intervention report. It is composed by maintenance activities performed by maintenance actors which create reports for future use; the maintenance strategy management model which depends on technical and financial indicators of the maintenance contract for each equipment; the maintenance management model which manages the different types of maintenance (corrective, preventive, predictive); the equipment states model which has as possible states: Normal state, Degraded state, Failure state, Programmed stop state; the historic management model which contains the main data related to the equipment maintenance; the document management model; the functional management model which describes the function of the equipment or of the component; the dysfunctional management model which stores the characteristics of different failure states of the equipment.

The two models are combined to develop a model with the aim of providing semantic maintenance. The mapping of the major parts of the models is shown in Table 1 (N/A=Not Available). The next step after the mapping was to develop the SMAC-Model which is shown in Figure 19. The details about the alternations made in the model are presented in the following sections. It should be noted that the alternations described were made on the basis of the model developed in section 4.2. A brief description of this section has already been published by Matsokis et al. [133].

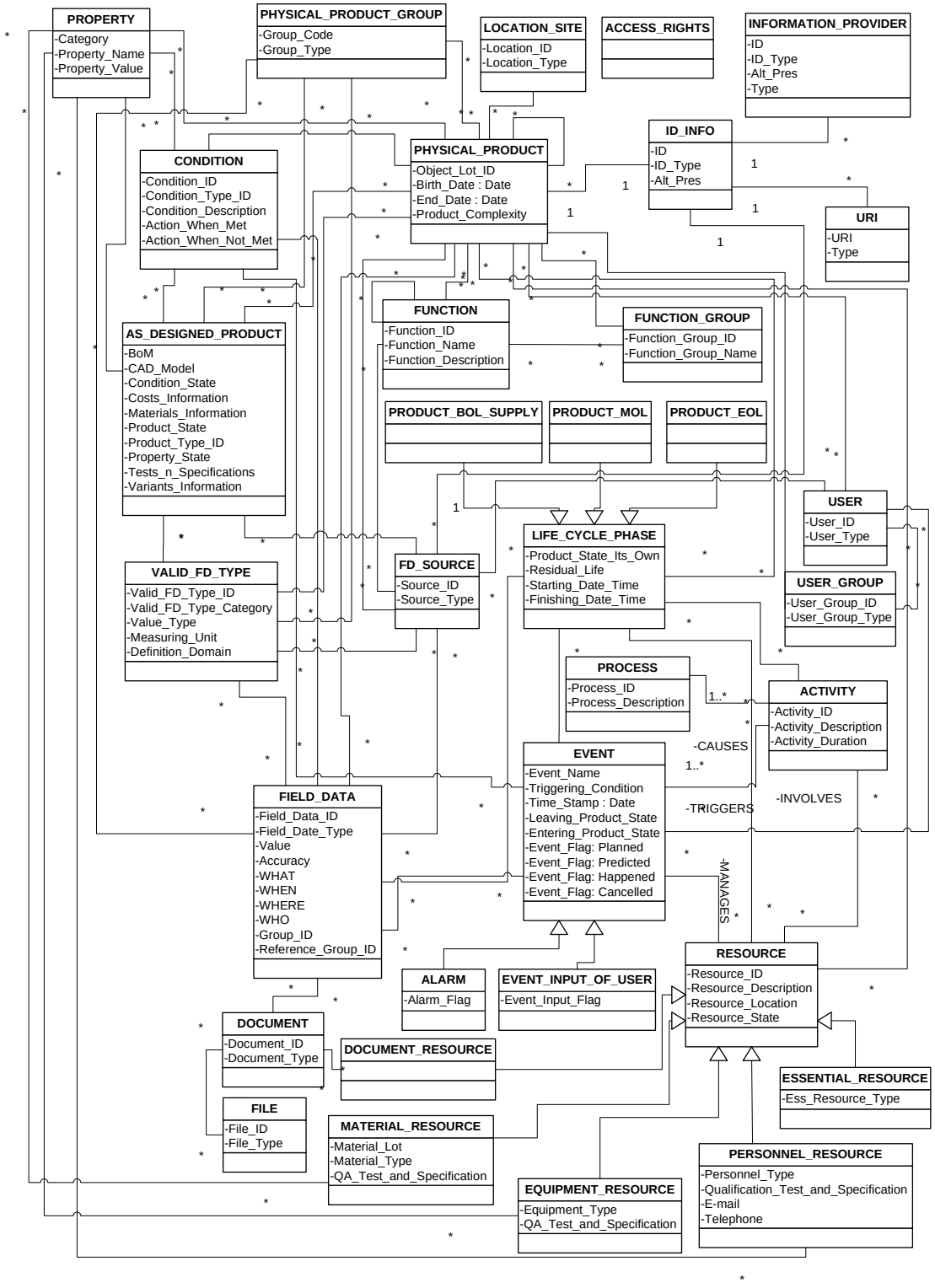

Figure 19: Complete UML schema of the SMAC ontology model.

The concept is to combine the Closed Loop-PLM SOM with the Proteus e-maintenance platform in order to provide more complete maintenance model applicable in the entire lifecycle. The SMAC-Model contains classes and relationships from both previous models. It should be noted that in this work the meanings of the main concepts of each model are translated into OWL-DL and they are described as such using the expressive power of the OWL-DL. For example, the dysfunctional management model of the PROTEUS model is described through the events, activities and processes and then documents are created. Similarly, during the future mapping with OSA-CBM, classes and parts of the OSA-CBM which deal with algorithms (calculations) would not be included in the new model since they do not describe concepts. Moreover, the SMAC-Model includes new classes, relationships (object properties) and attributes (datatype properties) in order to increase the capabilities of the model. These new elements derive from the fact that after combining the two models new opportunities were created, and hence, the model was extended to support them. The SMAC-Model developed is shown in Figure 19. The most important extensions in the model are described in the following paragraphs. The model is developed using OWL-DL which provides a number of functionalities. It should be noted that relationships have been expressed through OWL object properties and they are named according to domain-range policy, domain2range i.e. `Field_Data2Document`. Named associations of the initial model are unchanged.

Table 1: The mapping of the basic parts of the different systems.

Proteus	**Promise SOM**	**SMAC Model**
Equipment Expertise Model		
Physical Equipment	Physical Product	Physical Product
Equipment Model Group	Physical Product Group	Physical Product Group
Functional Component	Physical Product (Part of)-or-MOL	Function
N/A	N/A	Function Group
Additional Component	Field Data (FD) Source	Field Data Source
Monitoring System		
Sensor	Field Data Source	Field Data Source
Measure	Field Data	Field Data
Data Acquisition System		
Measure	Field Data	Field Data
Triggering Event	Event (after threshold filter)	Event

Sites Model	Resource (attr: Location)	Location Site, Resource (attr: Location)
Functional Model	Document Resource	Document Resource
Dysfunctional Model	Document Resource	Document Resource
Equipment States Model	Document Resource	Document Resource
Maintenance Types Model	Document Resource	Document Resource
Intervention Management System		
Intervention	Event	Event Input Of User
Activity	Activity	Activity
Actor	Resource	Resource
N/A	N/A	Process
Resource Management System		
Resource	Resource	Resource
Human Resource	Personnel Resource	Personnel Resource
Role	Personnel Resource (attr: Personnel Type)	Personnel Resource (attr: Personnel Type)
Material Resource	Material Resource	Material Resource
Material Resource	Equipment Resource	Equipment Resource
Maintenance Strategy	Document Resource	Document Resource
Documentation Management System		
Document	Document Resource	Document Resource
Historic Management System		
Life History	MOL_Phase	MOL_Phase
N/A	EOL	EOL
N/A	BOL	BOL
Need	N/A	Essential Resource
N/A	N/A	User
N/A	N/A	User Group
Alarm	Event	Alarm
ObservedEventByUser	Event	Event Input Of User
Intervention Order	Group of Activities	Process

4.4.1 Expansion in Classes

Each class in the model describes a concept of the real life. Therefore, the model was extended in classes in order to increase the described concepts. The classes added to the model were **Location_Site**, **Essential_Need**, **User**, **User_Group**, **Function**, **Function_Group**, **Alarm** (as critical event), **Event_Input_of_User**, **Process**. The classes added and the concept described by each class in detail, are:

- The **Location_Site** class was added which is related to **Physical_Product** via `Location_Site2Physical_Product` (and its inverse). Its datatype attributes are

`Location_ID` and `Location_Type`. The information stored in this class is for instance the geographical location of a manufacturing plant. This is important for the better management of resources while maintenance, since we know the location of the product and we can use the closest maintenance facilities possible. Furthermore, any maintenance activities will have to be compliant with local regulations.

- The **Essential_Resource** class as a sub-class of **Resource** class was added. Its datatype attribute is `Ess_Resource_Type`. This class describes the requirements of the physical product regarding infrastructure and its environment in order to be ready to perform its functions. Such needs could be power supply, water supply, gas supply, oil supply, sunlight, etc.
- **User** and **User_Group** were added. They are related `User2User_Group` (and its inverse), and to **Physical_Product** via `User2Physical_Product`, `User_Group2Physical_Product` (and their inverse relationships).
    - The concept described by the **User** class is that the user will be the "client" or "customer" who is buying the service of using the physical product/machine on contract: i.e. when one rents a car from a car rental provider, he is the user for a certain time period or/and a limit in km. Similarly, a company may “rent” a product for certain working hours with leasing and perhaps adjust it to the needs of the user.
    - The **User_Group** describes elements such as the type of maintenance performed by the user. Thus, we can have groups according to their maintenance contract type or the type of industry the machine is used in (for the use of the machine) i.e. form steel or aluminium parts. In the previous model this was only referred as a **Document_Resource** class and it was declared through a datatype attribute.
- **Function** and **Function_Group** were added. They are related to each other through the relationships `Function2Function_Group` (and its inverse), and to **Physical_Product** via `Function2Physical_Product`, `Function_Group2Physical_Product` (and their inverse relationships).

  - The idea behind the **Function** class is that each physical product may have one or more functions (i.e. rotation, linear movement, store coolant liquid, etc.). Therefore, whenever a physical product is degraded one or more of its functions are affected. Through the connection of this class with the **Physical_Product** we are able to know which functions are related to each individual physical product and they are or may be affected during its degradation.
  - The **Function_Group** is used to describe functions at a generic level i.e. group all material of heat isolation of the product, group all rotating parts of the product, etc.

- The **Alarm** class as a sub-class of **Event** class was added. This class contains only the critical events. The system issues events, some of which are evaluated as alarms. These events may cause the breakdown of a function of the physical product.
- The **Event_Input_of_User** class as a sub-class of **Event** class was added. This class contains only Events which are input by a user. These events may have been caused by:
  - The fact that there is place for improvement in the monitoring system i.e we don't have a sensor at a place where we should have it. In that case it gives us feedback for possible weaknesses of the system.
  - The slow response of the system in an abnormal situation.
  - An external factor i.e. in case of flood or fire in the building.
- The **Process** class was added. It is related to **Activity** via `Process2Activity` (and its inverse). The meaning of a process in this context is that a process consists of one or more activities and its task is to group together the maintenance activities. This was required in order to accumulate knowledge about which activities are performed per process and make the system capable of automatically listing the activities needed depending on the events.

All these classes were added to provide a wider support for the maintenance of the product, than the one provided by each one of the two initial models.

4.4.2 Expansion in Relationships

After extending the domain coverage of the model with the new classes, some more relationships were added to describe the links between the classes. These relationships compose the active network of communication of the model since they are used like verbs of the sentences in a structure "noun-verb-noun" which in the model is "class-relationship-class". During the usage of the model they may be used as the basis for introducing DL rules in the model. These were:

- The **Condition** class is related to **Event** via the relationships `Condition2Event` and its inverse `Event2Condition`. Thus, the model can describe the condition(s) that trigger an event and relate them directly.
- The **User** class is related to **Event** and to **Field_Data_Source** via `User2Event`, `User2FD_Source` (and their inverse relationships) respectively.
    - The relationship `User2Event` describes the case where a user notices some malfunction and makes an action. In this case an event instance is created and it is related to a user instance. This may provide feedback and may be a source for reporting bugs of the monitoring system.
    - The relationship `User2FD_Source` describes the case where a user notices some malfunction and acts as a field data source. In this case the user inputs data at the **Field_Data** class. Then this data will be evaluated by the system and an event instance may be created depending on the conditions.
- The **Function** class is related to **Field_Data_Source** via `Function2FD_Source` and its inverse relationship. Moreover, the **Function** class is related to itself with the relationship `FunctionIsComposedBy` and its inverse `FunctionComposes`.
    - The relationship `Function2FD_Source` is used to relate the function with the sensor of field data source. Thus, when an alarm is created from field data of a specific sensor, we know which function is affected.
    - The relationship `FunctionIsComposedBy` describes the case where a function is composed by a number of sub-functions. Similarly its inverse describes which sub-functions are composing more complicated functions.

These relationships create a more complete network of communication between the existing and the new classes.

4.4.3 Extension in Datatype Attributes

After making the changes in the classes and the relationships, some more datatype attributes were added to describe various requirements. These were:

- Attributes `Group_Code` to facilitate the No of Component and `Group_Type` to facilitate the description "designation objet" of the component were added to the **Physical_Product_Group** class.
- Attribute `Condition_Description` was added to the **Condition** class. This attribute contains a short description of the condition.
- In the **Alarm** class, for the better management of alarms, `Alarm_Flag` was added. There are two levels of alarm described through the datatype attribute `Alarm_Flag`:
    - Yellow alarm (the function is likely to fail ~50-75%).
    - Red alarm (the function is likely to fail >75%).

  Comment: These likelihoods are estimated on real time and could be coordinated with time. For example the likelihood of ~50% for the Axis X1 drive to fail in the next 5 working hours might be a red alarm, whereas a likelihood of ~50% for the Axis X1 drive to fail in the next 500 working hours might be a yellow alarm.
- In the **Event_Input_of_User** class the attribute `Event_Input_Flag` was added and it may have the following values:
    - Weakness Factor which describes the fact that there is place for improvement in the monitoring system i.e. we don't have a sensor at a place where we should have it. In that case the system doesn't "feel" the problem or has slow (less than satisfactory) response in an abnormal situation.

      The user is not an expert and therefore the exact weakness factor has to be verified by the cause/fault/data analysis.

- External Factor which describes an external factor which is coming from the environment of the product i.e. in case of flood or fire in the building, black-out etc. This is recorded since it may trigger activities.

  This case is out of the scope of the model since it cannot be predicted from the monitoring system.

- Attributes `Function_Group_ID` and `Function_Group_Name` were added to the **Function_Group** class in order to describe the elements of this class.
- The attributes of the **Function** class are `Function_ID`, `Function_Name` and `Function_Description`.
- Attributes `User_Group_ID` and `User_Group_Type` were added to the **User_Group** class in order to describe the elements of this class.
- The attributes added to the **User** class are `User_ID` and `User_Type`.
- The attributes added to the **Process** class are `Process_ID` and `Process_Description`.

The goal of all these new attributes is to describe better the various aspects of the maintenance of the product and like the relationships they may be used for introducing DL rules in the model.

The overall functionality of the developed system as well as the use of DLs is demonstrated in the next chapter in section 5.3. The model is used to facilitate the data about the MOL of a lathe machine. On the ontology model the data describing the complex machine which consists of ~1 770 parts has been loaded. This creates a complex environment of more than 240 classes, 3 000 instances and 20 000 triplets. Furthermore, in the case study also the time management concept described in the following section is implemented.

4.5 The Duration of Time Concept

The next step of this work is the development of a concept for better management and exploitation of time in PLM systems. In today's systems although time attributes exist is various parts of the systems, there are no systems which are based on time, although time is

objective and it naturally exists in all applications and parts of the models. The qualities of time characteristics were the initiative to select time as the basis for our methodology for model development, the "*Duration of Time*" concept. This concept introduces the idea of seeing all aspects and elements of a model as parts of time and it provides flexibility, application independence and simplicity. In this way time exists naturally in every part of the system like it actually exists in real life. Thus, time may be used to support a first level of data integration and system interoperability through system synchronisation (section 4.5.2).

4.5.1 Time implementation for Ontology based PLM

The aim of this work is to introduce a new methodology for improving today's Asset Lifecycle Management (ALM) and PLM systems in the aspects of data handling (visibility and integration) as well as system interoperability. Visibility of information between the different levels of abstraction in different information and data management systems is not always available and if achieved it requires a lot of effort due to the complexity of the systems (for the sake of simplicity in this document when we use the term "systems" we mean "Information and Data systems"). All these systems either are different to each other or are under the same commercial "ALM" system. In both cases it is very difficult to retrieve and synchronise the data of all phases (Beginning of Life (BOL), Middle of Life (MOL) and End of Life (EOL)) after the product exits its (BOL) phase (design and production). Furthermore, data is collected only for some pre-defined products-components. However, experience has shown that the requirements for the types of collected data change depending on the use of each part of the model and hence, essential parts of data are missing and are impossible to recover when needed in later stages. This leads into having stored data, for use as input in decision making, which is incomplete and therefore decision support is unsatisfactory.

The objective of the proposed methodology is to improve today's ALM and PLM systems by changing the use of time in the systems. The importance of time in ALM and PLM has been noted in section 3.3. Time has some qualities which make it special among all the attributes and in our opinion remain unexploited. Time is the only fundamental dimension which objectively exists along the entire life cycle of all individuals (including materials

and physical products) and it is an objective element. Time exists in our everyday life on different levels: duration of accomplishing a task, duration of coffee break, duration of a phone call, duration of studies, age of a human, roman era, duration of a trip, duration of a maintenance activity, working hours of a machine, etc. Time also has granularity in order to be easier comprehensible by humans depending on the application i.e. it is easier understandable to say that I signed a five year contract than to say that I signed a 43800 hours contract. In this way time is affecting all aspects of individuals and their qualities; people are getting older (changes in character due to experience, in health, etc.) and objects wear out. All have the need for some type of maintenance. Furthermore, time is simple, comprehensive and objective and therefore, application independent. For instance duration of 5 years is understood by all systems and humans. Of course it might have different meaning and importance when it is referring to the age of a human or of a machine. For instance if one is employed by company A for a duration of 5 years, it is not really important for him to know that the company has a history of 150 years. From the company point of view the individual exists only for a small fraction of its life, where as for the individual 5 years is an important part of his 35 years of work. Regarding assets, time has a meaning of useful life, working hours, maintenance intervals, etc. Similarly, a used component of a machine has its time in the previous machine and now it has a life in a current machine. In this way the component has more than one "middle of lives". Its lifetime history would be the following: duration *MOLa* of MOL A in machine A (during which it performs task A1, task A2, etc. with durations *a1*, *a2*, etc.), duration *r1* of re-manufacture, and duration *MOLb* of MOL B in machine B (during which it performs task B1, task B2, etc. with durations *b1*, *b2*, etc.). Of course the component might have unlimited number of future uses. In this way time describes the continuity of the functionality of the components.

4.5.2 Basis for The "Duration of Time" Concept

This work introduces the "*Duration of Time*" concept for improving today's ALM and PLM systems in the domains of data visibility, data integration and system interoperability. The main element of the concept, used for improving the systems performance is time. The concept is:

Since time exists naturally in all parts of these systems, it could be used as the universal common reference-basis for providing a first level of integration among the systems.

To fulfil this concept, time, should not be one part of the model, but it should be the basis of the model and all other elements should be parts of it. A schema of a possible model implementing the concept is shown in Figure 20.

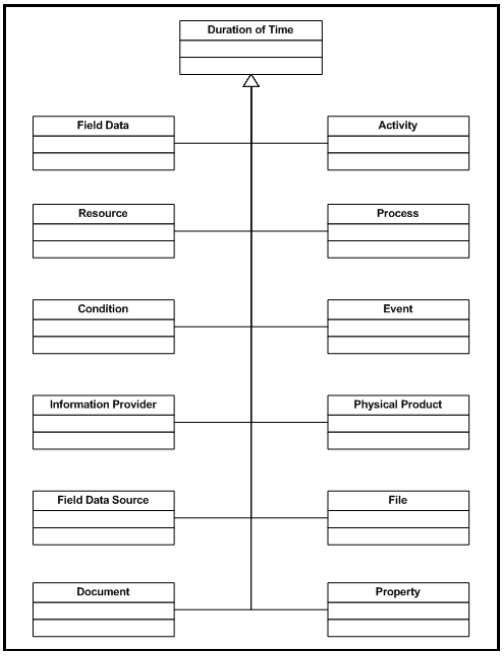

Figure 20: Schematic Duration of Time representation example.

The concept is easily applied on existing models by making a "duration of time" class as a super-class of all classes of the model. This class provides the unified time framework for the entire system (Figure 20). The concept is filed as a PCT (Patent Cooperation Treaty) application with serial number PCT/EP2010/053238 [134]. In general to implement the concept the following steps are necessary:

1. Set the **Duration of Time** class as a super-class of the model
2. Develop a time framework for the existing ontology PLM (i.e. start_date_time, end_date_time, duration), and introduce it in the **Duration of Time** class
3. The already existing data of the model are copied from the datatype properties of the pre-implementation classes to the new attributes of the **Duration of Time** class

4. All the time related datatype properties of the pre-implementation classes are deleted from the model. They are expressed by the datatype properties of the **Duration of Time** class
5. Select a central reference time for the model i.e. GMT or CET

Steps 3 and 4 are necessary only in the case of implementing the concept in already functioning models which contain data before the implementation. The technical details of the implementation in already functioning models which contain data depend on the tools used i.e. different step by step procedure is required to implement the concept in models developed in OWL-DL than in models developed in C++ or Java.

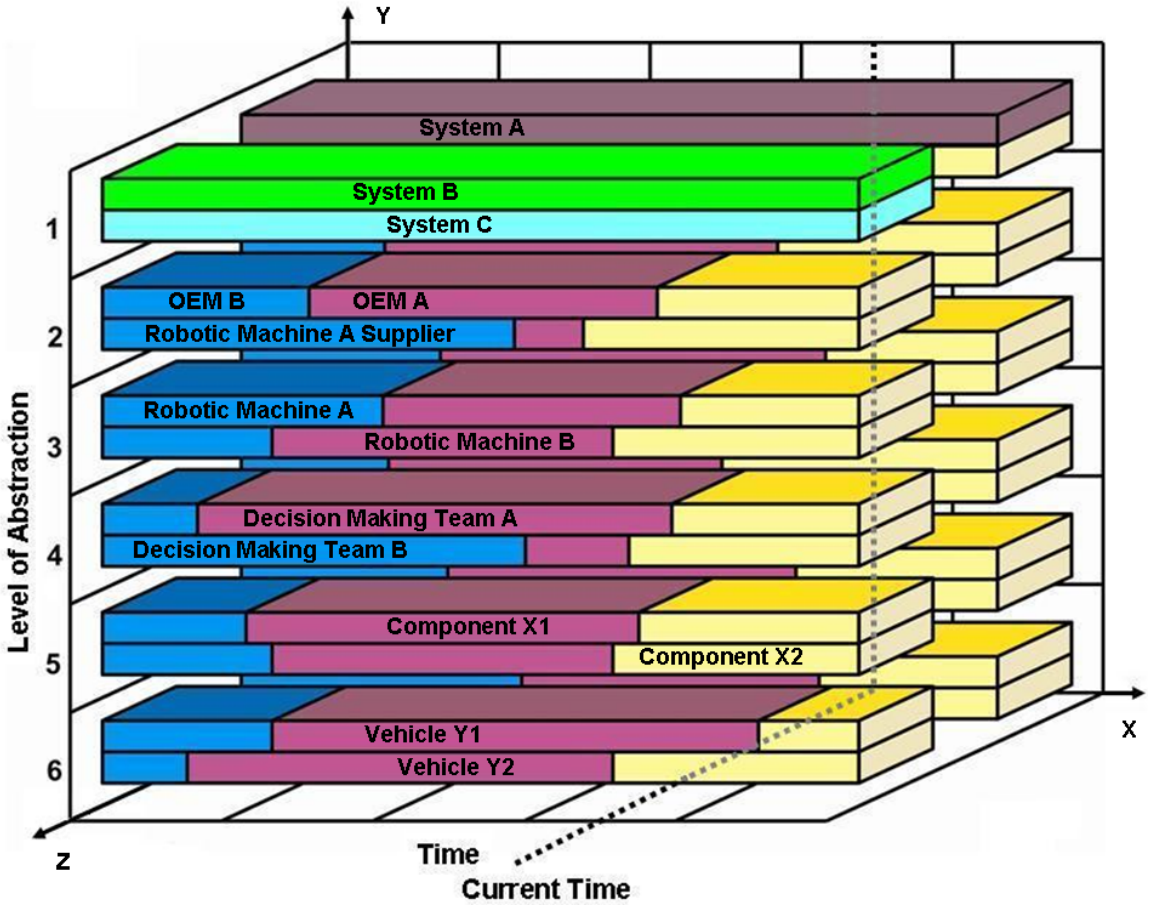

Figure 21: Multi-system architecture using the Duration of Time concept.

The "*Duration of Time*" concept has unique advantages over existing concepts, which stem from the qualities of time characteristics. Time is objective and it may be used as a guideline basis for achieving data integration and system interoperability. Therefore,

systems built on this concept take advantage of the time characteristics and when combined with semantics provide data visibility, data integration and system interoperability. Time is used as a reference-basis to provide a first step system to system visibility and common understanding no matter the different vocabulary, definitions, semantics or language used in the different systems. This allows for a better compatibility and portability of data from one system to another, since all of these elements are essentially defined with reference to time, which is common across all systems. Two different time based systems will certainly have in common their time attributes and therefore, they are synchronised even though they might have been extended and used differently. An example of how a group of systems using the "*Duration of Time*" concept would work and it would provide vertical and horizontal integration is shown in Figure 21. The description of this figure is:

- X-Axis: represents time; the length of the boxes represents the duration of the lifecycle of the element i.e. robotic machine, system A, etc.
- Y-Axis (Vertical integration): represents the different levels of abstraction (as it is shown also in Figure 2). These levels contain the different information systems for each level. The important information is whether two or more are on the same Y-Axis level. Therefore, System B and System C are in the same level. They seem to be on top of each other only for illustrative reasons.
- Z-Axis (Horizontal integration): represents the fact that more than one box can be at the same "Y-Axis" level on the same time of "X-Axis". Therefore, System A and System B are in the same level Y-Axis and X-Axis. They are parallel to each other on the Z-Axis only for illustrative reasons.
- "Current Time" line illustrates the vertical and horizontal visibility achieved by Duration of Time system. One may integrate all the systems on the time basis.

A comment on the figure is that it is clearly illustrated that a System may have longer lifecycle time of a decision making team. In this case the system collects the information and the knowledge of the other levels. Case studies implementing the concept are presented in sections 5.2 and 5.3.

4.6 Implementation Methodology of our Ontology-Based approach

This section describes a step-by-step methodology of implementing and using efficiently the system architecture described in section 4.1 in order to exploit DL capabilities in ontology-based PLM models. As it has been demonstrated in sections 4.2, 4.4 and 4.5, the initial model was slightly modified to facilitate several of the OWL-DL capabilities, always maintaining previously achieved characteristics. The developed ontology model is dynamic, it can store data about multiple products on a single source (it allows to record, store and process data-information about a number of systems in a single ontology source) and it has higher description ability (allows the user to see the multiple levels of inheritance).

Implementation Process

The proposed implementation methodology assumes that the initial OWL-DL model is developed by one team of developers of the manufacturer (or OEM: original equipment manufacturer) of the product, which has full administrator rights on the model (step 1). The model should be generic facilitating the most abstract concepts necessary to describe the domain. An example of such model is the model developed in 4.2. Then, the OEM team should populate the concepts with instances which are static (step 2) i.e. which parts of the car are being tracked (see **Physical_Product_Group** class in section 5.1). These instances cannot be changed by the users, but only by the OEM team. Therefore, they will be common for all the partners that will be using the model and they are the common vocabulary among the variations of the model. In the next step, the OEM team should develop a methodology for extending the model (step 3). The instructions to be followed by the users are: study the concepts of the model to find out according to what element (guideline) data will be categorised; select the most appropriate guideline; and then create sub-classes with DL rules using the guidelines you have selected. In the fourth step the OEM team develops a typical example following the instruction of the previous step. Then, the OEM team makes copies of the OWL-DL model (steps 5, 6 and 7) and distributes them to the partners (step 8). The partners populate their copies with data and extend them with classes following the instructions described in step 3. Furthermore, the OEM team collects copies from the partners in predefined time intervals i.e. the first working day of every month (step 10) and imports them into one model (step 11). Then, the DL-reasoner may be

used to check consistency, equivalencies and re-classification on classes and to categorise instances (step 12). Finally, SWRL or Jess rules might be added if necessary (step 13). The implementation steps briefly are:

1. Develop the ontology in OWL-DL (ontology *O* with the URI *U*)
2. Provide instances for the static parts of the model
3. Develop instructions for extending the model with DLs. A possible order is:
    a. Study the concepts included in the model
    b. Select guideline
    c. Develop sub-classes with rules
4. Provide a typical paradigm how to implement this methodology (of the previous step) according to the requirements
5. Make copies of the ontology model
6. Change the URI of each copy to *U'* (unique URI for each copy)
7. Set as its default namespace the URI *U* of ontology *O*
8. Distribute the models to the partners
9. The copies are populated and/or extended by the partners. In case of extension the partners have to follow the instructions of step 3
10. Collect the copies
11. Import the copies into one model
12. Execute the reasoner to check consistency, equivalencies and re-classification on classes and to categorise instances
13. Add SWRL and/or Jess rules if necessary

It should be noted that steps 6 and 7 may vary depending on the ontology editor used. In this work these steps have been developed to function correctly with the Protégé editor (version 3.4) and its current plug-ins as shown in Figure 11.

The most important part of the method consists of steps 2 and 3. Step 2 demonstrates how to use the developed ontology and step 3 provides the user with an application example on the domain. The latter is used to demonstrate the benefits obtained from implementing the ontology model.

In the case of implementing the time concept, the process described in section 4.5 is considered in different steps depending on the status of the model. If the model is new and does not contain data then the concept is implemented in step 1. The extra steps to be considered while developing the step 1 are (as described in section 4.5.2):

1. Set the **Duration of Time** as a super-class of the model
2. Develop a time framework for the existing ontology PLM (i.e. start_date_time, end_date_time, duration), and introduce it in the **Duration of Time** class
3. Select a central reference time for the model i.e. GMT or CET

If the model is already in use then the concept is implemented in step 3. The extra steps to be considered while developing the step 3 are (as described in section 4.5.2):

1. Set the **Duration of Time** class as a super-class of the model
2. Develop a time framework for the existing ontology PLM (i.e. start_date_time, end_date_time, duration), and introduce it in the **Duration of Time** class
3. The already existing data of the model are copied from the datatype properties of the pre-implementation classes to the new attributes of the **Duration of Time** class
4. All the time related datatype properties of the pre-implementation classes are deleted from the model. They are expressed by the datatype properties of the **Duration of Time** class
5. Select a central reference time for the model i.e. GMT or CET

The proposed implementation methodology makes the model being extensible while keeping compatibility with the other copies of the initial model. Each partner might use different terms describing the same concepts, which in other cases causes confusion, problems of interoperability and data integration, and still, no such problems are created since, efficient use of DL rules provides a solution. Moreover, the methodology is generic and therefore, applicable in a number of different domains. In chapter 5 applications of the proposed methodology in practice are presented.

4.7 Conclusion

In this chapter it has been presented: which methods, tools, models and theory we used; how we used them to create new opportunities for the PLM models; why we used them

(implement new functionalities in current models); what new opportunities are created for PLM models; the original "*Duration of Time*" concept developed; how to implement the combination of developed concepts and used methods and tools in PLM models.

To achieve the comprehensive demonstration of the system we presented the system architecture (Figure 11) and the functionality of all the different parts in the structure. Then, the goal was to make the current PLM models capable of utilising this architecture. Therefore, we transformed the PROMISE SOM model from UML to OWL-DL. This provided the new functionality of facilitating multiple data about multiple products under one single source. The transformation provided also the functionality of merging two or more models together. The question which arises on this is: "how to use the system architecture to automatically perform the mapping of the models during merging?". This question is discussed in chapter 5 section 5.1.5. Ontology merging has provided an extra capability which seems to be very promising: even if only in one model there is an important (one or more) new and beneficial element, after the merging the whole system will benefit from it. Moreover, the model was extended with elements of the PROTEUS model in order to facilitate more capabilities for maintenance.

Furthermore, the original "*Duration of Time*" concept is presented. The main aim of this concept is to exploit the characteristics of time (in its generic meaning) and to provide original solutions towards data integration and system interoperability.

Finally, a generic implementation method of the system architecture and the developed "*Duration of Time*" concept is proposed. Applications in case studies of this method, the system architecture and the concepts are presented in the next chapter.

5
Case Studies

In this chapter three case studies are described in detail. The aim of these case studies is to demonstrate the new opportunities existing for the Closed-Loop PLM. This includes the demonstration of the functionalities of the ontology models using DLs, the exploitation of the reasoning capabilities of the relevant architecture described in section 4.1 and the implementation of the "*Duration of Time*" concept. The first case study is an application of the model developed in section 4.2. In this case study the main benefits of using OWL-DL are demonstrated. Applications described in this case study show: the usage of the DLs and the DL-reasoner for the re-classification of the class-hierarchy; the check of equivalencies among the concepts of the model; the check of consistency of the model; the expressivity of the model; and the logical categorisation of the instances under the classes. These functionalities support data integration among the different variations of the system and therefore, data is located in the right place in the model. Actors of all the PLM phases may retrieve and use the data. The second case study is an application of the model developed in section 4.2 combined with the "*Duration of Time*" concept developed in section 4.5. Its aim is to demonstrate the applicability of the "*Duration of Time*" concept and the benefits it provides to the current model. The "*Duration of Time*" elements may be used as the common reference-basis among the different models that implement the architecture of the concept. The advantage towards Closed-Loop PLM is that the continuity of information is preserved through the common reference-basis during the lifecycle. The third case study is an application of the model developed in section 4.4 combined with the concept developed in section 4.5. This case study demonstrates the applicability of the "*Duration of Time*" concept and of the OWL-DL in a complex industrial environment.

5.1 Case Study 1

The aim of this case study is to demonstrate the functionalities of the ontology model developed in section 4.2 towards providing features for realising the Closed-Loop PLM. This is performed through extending the model using DLs and exploiting the reasoning capabilities of the relevant architecture shown in Figure 11. The machine-understandable model is used to make the information visible and ready for further use as input to all PLM phases, which means that information is categorised at its logical place in the model. The case study deals with information collected during the Middle of Life (MOL) in order to be used in the Beginning of Life (BOL) and in the End of Life (EOL).

This case study represents an example of using the developed ontology as a database for the MOL of passenger vehicles, the data of which can be later used as input to provide decision support for agents in both the BOL and the EOL. The specific dictionary (describing which values and parts need to be tracked through MOL) for the case study has been developed after combining requirements of application scenarios dealing with MOL and EOL cases in automotive industry [135]. In this case a part of the data stored in the model is to be used as input to a Decision Support System (DSS) for all PLM phases: in BOL for improving design and production, in MOL for improving maintenance and in EOL for supporting dismantling, recycling, re-manufacturing, re-use and disposal. The steps followed are: populate the ontology model with instances, extend the model according to requirements, provide guidelines for sorting data according to requirements to the extended model while supporting data integration.

This case study represents an implementation of the method described in section 4.6 on the ontology model developed in section 4.2 (Figure 18). Strictly following the steps of section 4.6, this case study goes as far as step 12 and overall provides a paradigm for implementation (step 4). It should be noted that, initially, in section 5.1.2 the model has been populated without following steps 3 and 4. The extended model is compared with both the initial model and the model before the extension as well as with a variation of the model. The implementations included in section 5.1.3 are actually step 3 and provide extension guidelines. Section 5.1.4 represents steps 4 and 12: testing the model. The contents of 5.1.3 and 5.1.4 can then be used as a paradigm for constructively extending the model.

The model is developed by the OEM and copies of the model are distributed to its maintenance providers to collect the maintenance data of the products. Each maintenance provider extends the model according to his needs, using its local language and terms, and uploads the data into the model. When the OEM collects the different models from the different maintenance providers, and loads them under the same Protégé-OWL project, the DL-reasoner categorises the information as well as the new classes at their logically correct place in the model. This is performed through efficient use of DL rules (section 5.1.4). The categorisation of the data-instances contained in the model under the new sub-classes and classes has been also proposed and demonstrated (section 5.1.3).

In section 5.1.2 it is shown how the model is developed and used initially as taxonomy without using the DLs and other tools. In sections 5.1.3 and 5.1.4 excessive use of the DLs and the relevant tools is performed in order to demonstrate the capabilities of the model.

5.1.1 Ontology Development Description

The OEM team has developed (step 1) the model shown in section 4.2 (Figure 18). Then, data describing three passenger vehicles has been added to the model. Therefore, data was loaded in the model without having the necessary detailed structure. The amount of data stored, grew significantly as we added data describing more physical products. The aim is to test its functionality using the provided reasoning capabilities. Hence, the classes of the ontology have been deliberately populated with instances in such a way as to create the most complicated possible "data management case".

This situation demonstrates the case of not having defined the concepts (classes and sub-classes) to the right level of detail in advance. This is due to either poor or due to incomplete design of the system for the implementation or/and later changes in requirements. The data model developed for an application has been at some stage considered as "complete" and is implemented by engineers in real-life use. It is like performing steps 1, 2, 5, 6, 7, 8 and 9 (without performing the steps 3 and 4) of section 4.6. However, the models initially considered as "complete" in a later stage are extended and improved according to experience collected in practice and according to new requirements. *In our case, the initial model is extended by adding sub-classes to the already existing classes* or even by creating new classes. This provides support to data categorisation

through semantics which is very useful for Closed-Loop PLM since it makes data both human- and machine-understandable. Thus, the data is ready for further use as input to all PLM phases.

5.1.2 Populating the Ontology Model

The existing classes of the model developed in section 4.2, are populated with instances containing the data of three passenger vehicles. Thus, there is no structure of sub-classes describing the specific application requirements for directing the user to store the right data to the right place. As a consequence, each class of the initial ontology model has a mixture of different instances. For instance, all physical products such as batteries, engines, etc. are instances of the **Physical_Product** and whenever a new physical product is added, it is added randomly to the list of instances of this class (Figure 22). The same applies for all the rest of the classes. On the other hand, this provides easy data integration and interoperability since the classes are the same for all the variations of the model of all the partners.

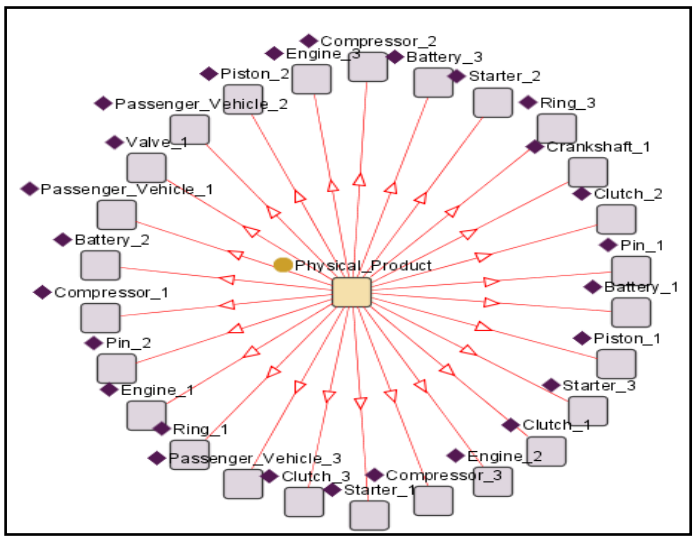

Figure 22: Physical Product class data of the initial model.

5.1.2.1 Populating Process

Firstly, we populated the **Physical_Product_Group** class and the **Valid_Field_Data_Type** class because they describe groups of instances of other classes. These instances are static and cannot be changed by the users, but only by the OEM team

(step 2). The instances of these two classes were named according to the requirements of PROMISE application scenarios, i.e. a passenger vehicle consists of a battery, a clutch, a crankshaft, an engine, pistons, pins, rings and valves. This is because these are the car parts that the manufacturer, engineering team, etc. is interested to track. The **Physical_Product_Group** class was populated containing instances describing physical products of the same type. These instances declare the types of physical products that can appear in the ontology. The same strategy was followed for the **Valid_Field_Data_Type** class instances, which define the measuring unit and other attributes of the **Field_Data** instances as well as **Field_Data_Source** instances. The instances of these two classes are shown in Table 2. Furthermore, the **Property** class contains the information provided by the manufacturer and was extended with the sub-classes **Material_Code**, **Product_Info**, **Serial_Number**, **Substitution_Mileage** and **Vehicle_Code**.

Table 2: List of instances for two selected classes

Class	Instance
Physical_Product_Group	*Physical_Product_Group_Battery*
	Physical_Product_Group_Clutch
	Physical_Product_Group_Compressor
	Physical_Product_Group_Crankshaft
	Physical_Product_Group_Engine
	Physical_Product_Group_Passenger_Vehicle
	Physical_Product_Group_Pin
	Physical_Product_Group_Piston
	Physical_Product_Group_Ring
	Physical_Product_Group_Starter
	Physical_Product_Group_Valve
Valid_Field_Data_Type	*Aging*
	Battery_Voltage_Data
	Car_Temp
	Car_Humidity
	Clutch_Pressed
	Compressor_Pressure
	Eng_Temp
	Mileage
	New_Substitution_Date
	New_Substitution_Mileage
	Out_Temp
	Starting

Then, copies of the model were made and distributed to partners (steps 5, 6, 7 and 8). Next step was the addition of the instances of the rest of the classes. This represents a part of step 9 since the model is not extended with sub-classes. These are the instances that normally are added by the users. While creating each instance, all values for the datatype properties were added. Furthermore, the new instance was either associated to already existing instances or new instances were created to be associated with them. For example when *Passenger_Vehicle_1* was created, it was associated to the already existing *Physical_Product_Group_Passenger_Vehicle*. Then, for the object property `isParentOf` the instance *Engine_1* was created. This process continued until we had all related instances. The lists of instances are dynamic and can be altered whenever necessary. The final state of *Passenger_Vehicle_1* is shown in Figure 23.

In this way all classes were populated containing all the data about attributes and associations in their instances. The main drawback about this process is that big amounts of data are stored in each class, referring to different real life artefacts. In **Physical_Product** we have all different kinds of physical products, while in **Field_Data** we have all instances collected by the field data sources and they are referring to both different physical products and different valid field datatypes. Similar is the situation for the **Field_Data_Source** where its instances are representing all sources of data (i.e. sensors) for all physical products and different valid field datatypes. In **Property** we also have all properties given by the manufacturer about all different physical products. While adding data describing more physical products, the amount of data in each class grows and data becomes very difficult to handle and to extract useful information from. This represents, for example, the situation that is faced by maintenance teams tracking data and willing to have an overview about the fleet of vehicles they are responsible for. On a higher level, the manufacturer can also collect the data from different maintenance teams, from different countries and merge them together under one source. At this stage a data repository on a common source has been developed, without having any ruled sorting and hence, and it is difficult to manage.

At this stage the system provides the following advantages:

- Data of multiple products are stored under one source

- The different copies of the model can be collected and merged. This is performed without any problems since all the copies have exactly the same class-hierarchy. Thus, data integration and interoperability between the different copies of the model is guaranteed.

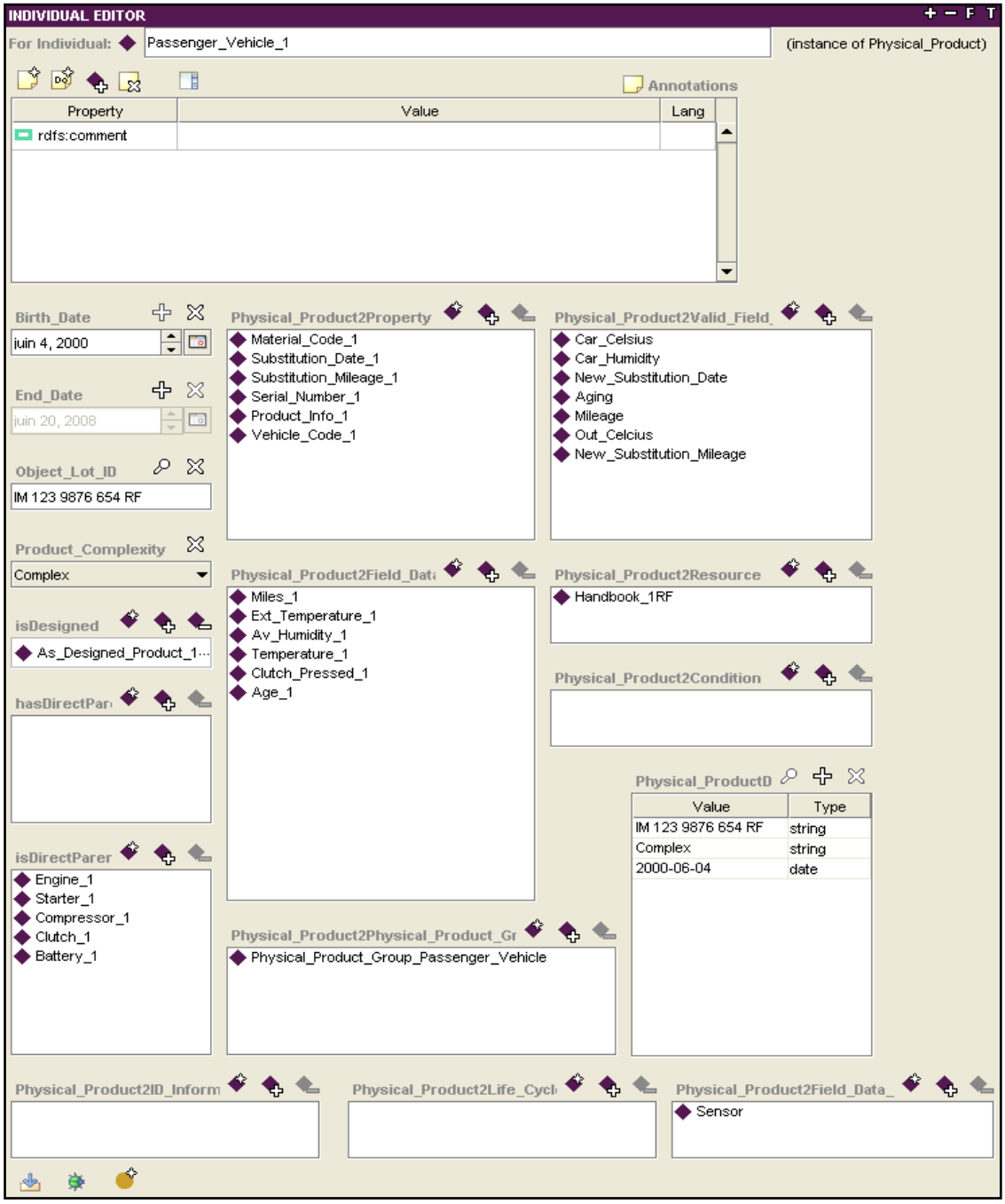

Figure 23: Instance editor of *Passenger_Vehicle_1* instance.

The model at this stage has some disadvantages such as that when the model is extended it loses the integration and interoperability with the other copies of the model and to solve this a manual mapping between the models should be performed; the data is not sorted according to various criteria i.e. to what type of data it is. These disadvantages are dealt with in the following paragraphs.

5.1.3 Inferring Instances

In this section, it is demonstrated how to create new sub-classes with rules for sorting data in any desired manner. Thus, all engineering teams may achieve auto-categorisation of data immediately after it is inserted in the model. In paragraph 5.1.2 we have described briefly the state of the populated model. Although having the advantages of interoperability and integration since all classes are the same for all models; these models have a lot of data allocated in their generic classes. The solution chosen is *the extension of the existing classes with new sub-classes by using DL rules*. The means that we have to implement step 3 and provide a representative paradigm (step 4) showing how to follow the guidelines for extension. Thus, each new sub-class should be defined with DL rules. This is similar to attempting to sort computer documents by creating new sub-folders of existing folders and sub-folders of them etc. as well as providing a smart "auto"-sorting method to sort the documents according to keywords. However, this leads to each different actor of the extended enterprise, having his own ideas for extending the model in order to facilitate better his needs. Differences appear in both naming policy of the new sub-classes and in criteria chosen for sorting data. Thus, in the end we have many different versions of the model.

Our mission is to solve the problem of how to preserve the advantages of data integration and interoperability in tandem with extending the model with new sub-classes. The answer to this is extension of the model with facilitating reasoning capabilities. All different actors will try to extend the model in order to facilitate better their needs of expressivity. Then, a new question is which the best guidelines for the rules are. The answer is the requirements. In this scenario after studying the model (step 3a) we decided to use (step 3b) the **Physical_Product_Group** class and the **Valid_Field_Data_Type** class as two major guidelines for developing rules for sorting the instances of the **Physical_Product** class.

Similarly the categorisation of the instances of the **Field_Data** class has been performed mainly according to the **Physical_Product** class. Other guidelines have also been used as shown in the following paragraphs.

A lean and easy to apply method has been developed to make the data manageable and extract useful information from it. To achieve this we added rules and we run them on the DL-reasoner Pellet 1.5.2. The DL-reasoner has been used to read the semantics of the sub-classes and to infer and distribute the instances to the sub-classes automatically following the applied rules. Typical examples are presented in paragraphs 5.1.3.1 and 5.1.3.2. The way the model is extended is not unique and whenever required it may be altered or extended further in order to query the ontology. The use of DL rules on the instances provides the advantages:

- It is no longer necessary for the user to know the exact detailed structure of the model, since new instances are located under the right class automatically.
- The system automatically avoids the creation of data miss-location or of data duplicates.
- In cases of importing variations of the model under one source the OEM is not required to know the detailed structure of the final model since the model is machine-understandable and the DL-reasoner is used.
- The method is very flexible: the way the model is extended is not unique and whenever required it may be altered or extended further in order to query the ontology.

All the implementations demonstrated in paragraphs 5.1.3.1 and 5.1.3.2 are implementations of step 3c developing sub-classes with rules. In 5.1.3.2 also an SWRL rule with the Jess rule engine are used.

5.1.3.1 Physical Product Instances

The **Physical_Product** class contains all the products that are being tracked. Sorting them according to their type is useful, giving engineers an overview of how many products of each product type are being tracked. First of all, eleven sub-classes (step 3c) of the **Physical_Product** class were created according to the eleven types of products. The instances of the **Physical_Product_Group** class are the dictionary (also called reference) of the **Physical_Product**, declaring the types of physical products. This is declared by adding rules to each new sub-class relating it to a specific instance of

Physical_Product_Group class. For example, to make the reasoner understand that batteries are those physical products that are related to the physical product group battery, we added to the sub-class **Battery** as Necessary and Sufficient the following restriction:

Battery ≡ Physical_Product∩ ∋
Physical_Product2Physical_Product_Group.(Physical_Product_Group_Battery)

This is translated in human language as: **Battery** is a **Physical_Product** whose object property `Physical_Product2Physical_Product_Group`, has the value of the instance of the **Physical_Product_Group** class, named *Physical_Product_Group_Battery*. Similar sub-classes and rules where used for the other ten types of products.

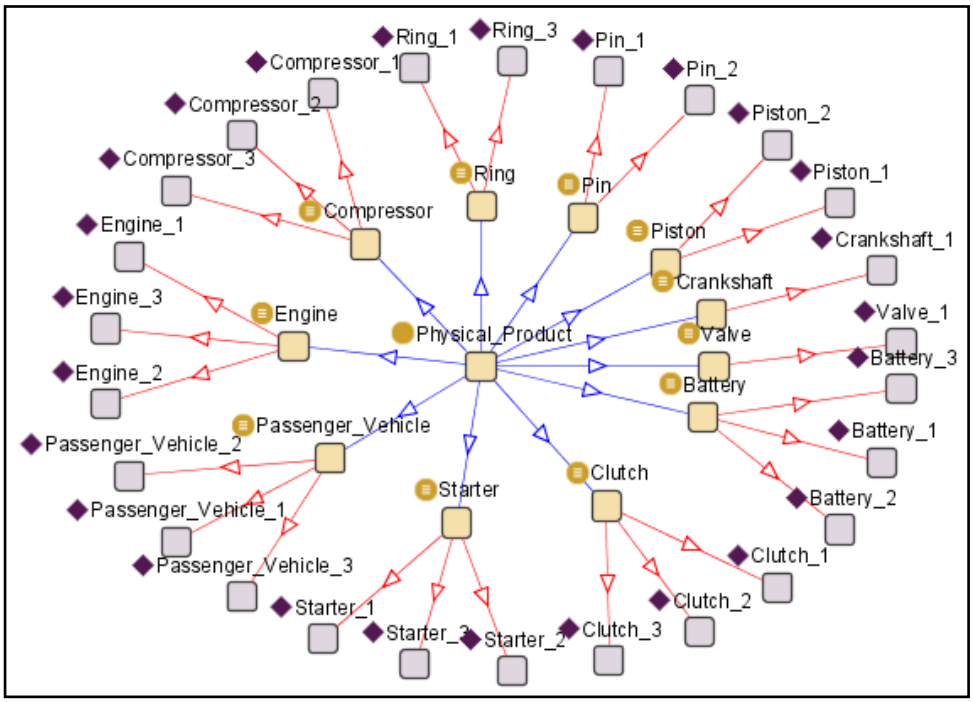

Figure 24: Physical Product class data after distribution.

Then we run the reasoner to achieve separation and sorting of data. The result for the data of the **Physical_Product** is shown in Figure 24. Specifically, this figure shows all the eleven sub-classes of the **Physical_Product** class and how the reasoner has distributed the instances among them according to the rules. Thus, in this case the instances have been distributed to the sub-classes according to which **Physical_Product_Group** they belong to. Comparing Figure 24, with the Figure 22 which shows the structure of all instances

stored under the **Physical_Product** class of the initial model, it is obvious that the mission of sorting data has been accomplished. Answers to questions like "which of the physical products are batteries?", "Which are engines?" etc., and in general: "which of the physical products of a certain, selected product type?" have been achieved.

Another way for categorising the instances of the **Physical_Product** class was their complexity. **Physical_Product** instances were categorised according to their complexity, described through the datatype property `Product_Complexity`. To achieve this we added **Complex_Physical_Product** as sub-class of **Physical_Product** with Necessary and Sufficient the following restriction:

$$Complex\_Physical\_Product \equiv Field\_Data \cap \ni Product\_Complexity.(\text{"Complex"})$$

This is translated in human language as: **Complex_Physical_Product** is a **Physical_Product** whose datatype property `Product_Complexity` has as value "Complex". The same was achieved for the "Simple" products. The result of the rule is shown in Figure 25.

The **Physical_Product** instances were also categorised according to which **Physical_Product** instance they are part of. The need for this categorisation arose in paragraph 5.1.3.2 when trying to sort the **Field_Data** instances of the complex physical products. The sub-classes containing these data are the key for "finding" the indirect **Field_Data** instances of the complex physical products. As a solution we added **Parts_of_Physical_Product_P_V_1** as sub-class of **Physical_Product** with Necessary and Sufficient the following restriction:

$$Parts\_of\_Physical\_Product\_P\_V\_1 \equiv Physical\_Product \cap \ni hasParent.(Passenger\_Vehicle\_1)$$

Similar categorisations were carried out for all the **Complex_Physical_Product** instances in order to sort their data.

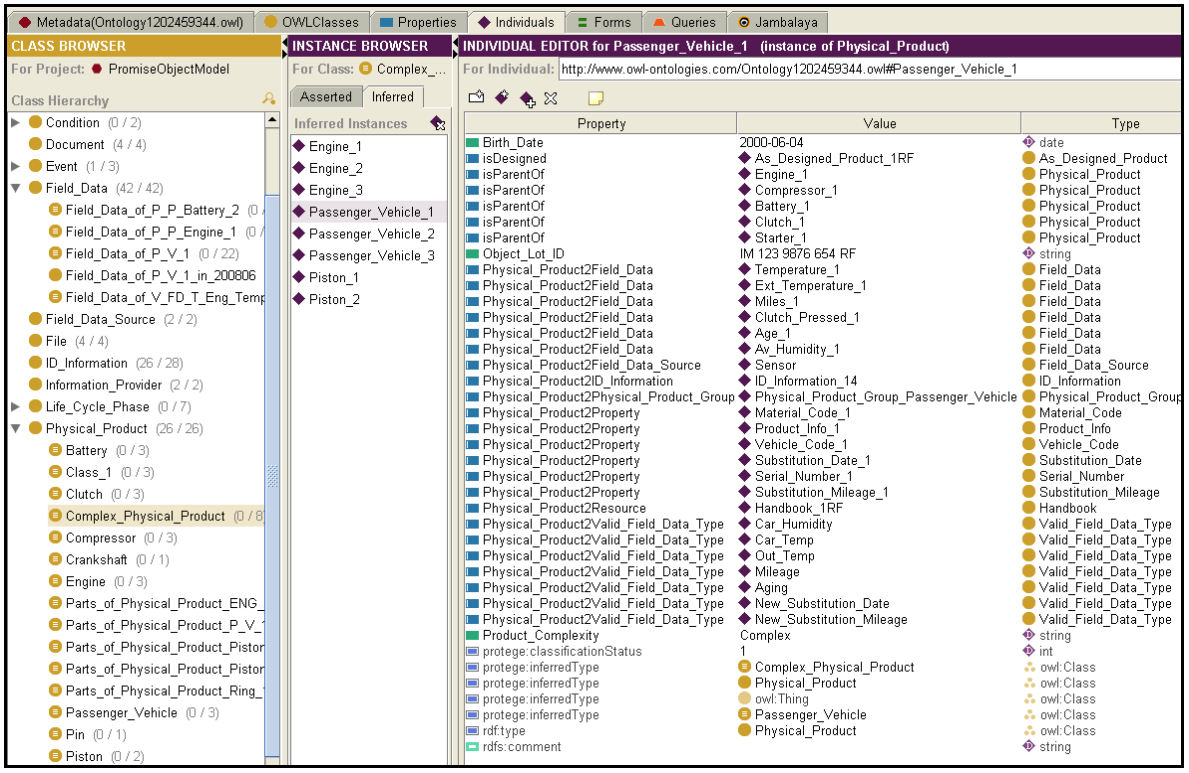

Figure 25: Instances of Physical Product class sorted according to their complexity

5.1.3.2 Field Data Instances

The **Field_Data** class contains all the data which comes from sensors and other sources of tracked physical products. Sorting field data according to the criterion "which physical product they belong to" is useful for several maintenance cases. For instance, in the case of monitoring the status of a product we can have all data about it under one sub-class and query only this class. For achieving this, a sub-class of the **Field_Data** should be created, for each physical product. Firstly, we sorted the data which is being tracked for *Engine_1*. For this reason we added **Field_Data_of_P_P_Engine_1** as sub-class of **Field_Data** with Necessary and Sufficient the following restriction:

$$\text{Field\_Data\_of\_P\_P\_Engine\_1} \equiv \text{Field\_Data} \cap \ni \text{Field\_Data2Physical\_Product.(Engine\_1)}$$

This is translated in human language as: **Field_Data_of_P_P_Engine_1** is a **Field_Data** whose object property `Field_Data2Physical_Product` has as value the instance of the **Physical_Product** class, named *Engine_1*.

Then we run the DL-reasoner to achieve sorting of data. The result for the data of the **Field_Data_of_P_P_Engine_1** is shown in Figure 26. In this figure it is shown that under the **Field_Data** there are forty two instances and the ones (five) related directly to *Engine_1* have been inferred under **Field_Data_of_P_P_Engine_1**. By adding more sub-classes of **Field_Data** with similar rules we can extend this, sorting the field data for all individual physical products. Answers to questions like "which field data belongs directly to *Engine_1*" or "which field data belongs to any other physical product except *Engine_1*?" etc., and in general: "which field data belongs directly to a certain, selected physical product?" have been achieved. Although the data of the simple products are sorted, the data sorted for the complex products are not complete since only the data related directly to them is sorted. This limitation is solved in the next paragraph.

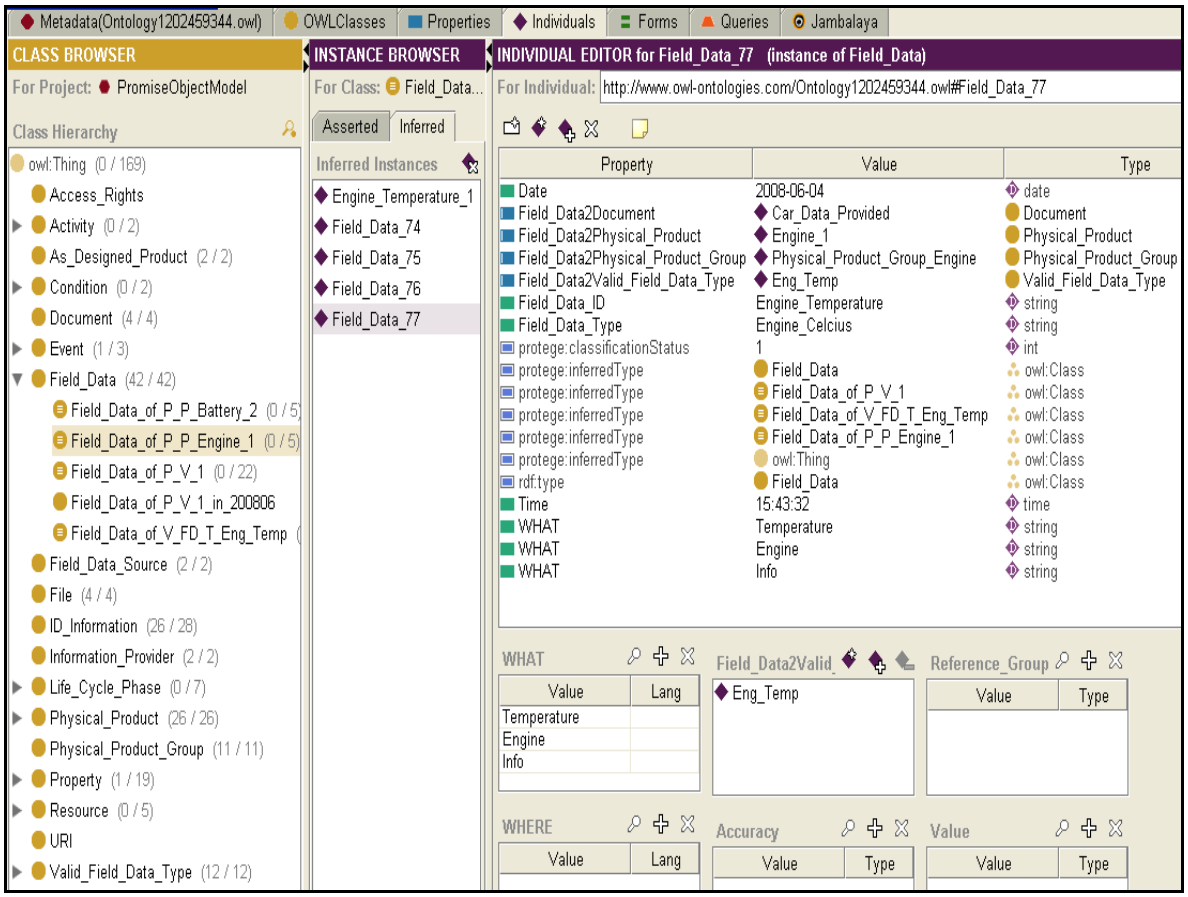

Figure 26: Instances related to Engine_1 instance have been sorted under Field_Data_of_Physical_Product_Engine_1.

Alternatively the **Field_Data** instances are sorted according to the type of the data stored. This is useful in cases of using the data as input for statistical research, defining for instance the real requirements of a client (fleet management) according to real field data. In this case we sorted out the data which is being tracked for engine temperature *Eng_Temp*. Then, we added **Field_Data_of_V_FD_T_Eng_Temp** as sub-class of **Field_Data** with Necessary and Sufficient the following restriction:

$$\text{Field\_Data\_of\_V\_FD\_T\_Eng\_Temp} \equiv \text{Field\_Data} \cap \ni \text{Field\_Data2Valid\_Field\_Data\_Type.(Eng\_Temp)}$$

Rules of this type answer to questions like "which of the tracked field data elements are of a certain, selected valid field datatype?".

To answer to questions like "which properties belong to a certain, selected product", sub-classes were added under **Property** and a restriction on the relation `Property2Physical_Product` was added to each one. Similar categorisations were achieved in the classes **Field_Data_Source** and **Resources**.

Similarly we sorted the data of the classes: **Field_Data**, **Valid_Field_Data_Type**, **Property**, **Field_Data_Source**, **Resources** and **Physical_Product** as shown in Table 3.

Table 3: Sorting of Data overview

Class	Guideline	General Question Answered
Field_Data	**Physical_Product**	Which field data belongs to a certain, selected physical product?
	Valid_Field_Data_Type	Which of the tracked field data are of a certain, selected valid field datatype?
Valid_Field_Data_Type	*Battery_1*	Which of the valid field datatype instances are related to a certain (i.e. *Battery_1*), selected physical product?"
Property	**Physical_Product**	Which properties belong to a certain, selected product?
Field_Data_Source	**Physical_Product_Group**	Which field data sources belong to a certain group, of physical products?
Resources	*MOL_ Battery_1*	Which resources have worked for a certain physical product?
Physical_Product	`Product_Complexity`	Which product consists of more products?

Field data of complex physical products

As it is mentioned in the previous paragraph a limitation appeared while categorising the **Field_Data** instances of complex physical products. The solution was given with the use of **Parts_of_Physical_Product_P_V_1** class and the rest of the classes of its type. In the following example we sorted out the data which is being tracked for *Passenger_Vehicle_1*. Then, we added **Field_Data_of_P_V_1** as sub-class of **Field_Data** with Necessary and Sufficient the following restriction:

$$\text{Field\_Data\_of\_P\_V\_1} \equiv \text{Field\_Data} \cap (\ni \text{Field\_Data2Physical\_Product.(Passenger\_Vehicle\_1)} \cup \exists \text{Field\_Data2Physical\_Product.(Parts\_of\_Physical\_Product\_P\_V\_1)})$$

In this way the **Field_Data_of_P_V_1** class contains all the data which are related to *Passenger_Vehicle_1* and to all its simple or complex components. Similarly we sorted the data of the complex physical products.

Time oriented queries

Accurate estimation of the time intervals for maintenance or of possible breakdowns is aimed at improving the service provided towards more reliable predictive maintenance. Therefore, sorting data according to when they were collected is an essential element of modern PLM systems.

The **Field_Data_of_P_V_1** class was used further to accomplish "when" queries for the complex product *Passenger_Vehicle_1*. More specifically, we achieved separation according to when a field data was recorded and to which complex physical product it belongs to. The construction of the "when" queries was based on the time and date stamps of the data in the **Field_Data** class. The **Field_Data** recorded in June of 2008 about *Passenger_Vehicle_1* were found using the following SWRL rule:

$$\text{Field\_Data\_of\_P\_V\_1(?fdx)} \wedge \text{Date(?fdx, ?zx)} \wedge \text{temporal:after(?zx, "2008-05-31T23:59:999")} \wedge \text{temporal:before(?zx, "2008-06-30T23:59:999")} \rightarrow \text{Field\_Data\_of\_P\_V\_1\_in\_200806(?fdx)}$$

Then they were sorted under **Field_Data_of_P_V_1_in_200806** class by returning the asserted values to the ontology using the Jess rule engine. Answers to questions like: "which field data of **Field_Data_of_P_V_1** was recorded in June 2008?" and in general: "which **Field_Data** was recorded in a certain time period about a complex or simple physical product?" were achieved.

5.1.4 Supporting Decision on Model Extension

Problems concerning system interoperability and data integration have to be solved. In the example in 5.1.3 each engineer or engineering group of an extended enterprise might use different terms describing the same concepts, which causes confusion, problems of interoperability and data integration like in existing systems. In this paragraph we demonstrate how the use of DL rules used in the case study provides a solution to these problems. A number of these applications is presented in [136].

The model was extended in section 5.1.3 according to the needs of the user. The extension is very practical for the users of each copy of the model. However, when the different copies of the model are collected by the OEM (step 10) and merged together (step 11) problems concerning system interoperability and data integration are created. Perhaps two copies use different words to describe the same concept or the level of detail of one copy is more than the one in another copy. This situation causes confusion, problems of interoperability and data integration like in existing systems. In this paragraph we demonstrate how the use of DL rules used in the case study provides a solution to these problems. In this section steps 4 and 12 are performed. Step 4 is performed for the overall testing of the functionality of the model and step 12 is performed for the logical testing of the model.

The model can be extended to facilitate a wide range of various different requirements. Each engineering team may extend it differently, sorting data in different classes with different meanings. On the other hand, at the extended enterprise level, we must have an overview of the data. The key issue for the extended enterprise are the guidelines for the rules to be followed by the engineers (described in step 3). No, new sub-class should be created without having rules. These rules describe the logical concept, each new sub-class represents in real life. Provided this, each team can use its own way to extend the classes.

In this section the use of DLs allows the DL-reasoner to:

- Check the model for its consistency
- Update the class-hierarchy
    - New sub-classes are re-classified to their logical position
    - Any equivalencies are understood by the DL-reasoner

In this way, integration of data and concepts is maintained through DLs and the DL-reasoner, and therefore, the OEM has clear view of the data and its structure in the total number of the copies of the initial model.

Supporting Interoperability and Data Integration

The model can be extended to facilitate a wide range of various different requirements. Each engineering team may extend it differently, sorting data in different classes with different meanings. On the other hand, at the extended enterprise level, we must have an overview of the data. The key issue for the extended enterprise are the guidelines for the rules to be followed by the engineers. No, new sub-class should be created without having rules. These rules describe the logical concept, each new sub-class represents in real life. Provided this, each team can use its own way to extend the initial class-hierarchy.

We assume that the initial has been extended with sub-classes using DL rules. The rules make the model machine-understandable. The reasoner will compare the rules among all existing classes. Thus, when creating a new sub-class with rules and then run the reasoner, the reasoner understands the rules and declares the classes as "equivalent", meaning that they express the same concept. In this way, it declares that the two classes, which have the same rules, are actually the same class but with different name. An example is shown in Figure 27. In this example a new class **Class_1** has been created having the same rules as **Battery** class. The reasoner has coloured the two classes blue and has declared:

$$\text{Battery} \equiv \text{Class\_1}$$

In this way the system indicates that the two classes are equivalent. Answer to questions like: "Does new **Class_1** already exist with a different name?", in general: "does the new class, which has just been created, already exist?" were achieved.

When the engineers will collect all the models developed by their colleagues they have the options:

1. Have different categorisations among different models
2. Find same ruled classes between different models
3. They can categorise the data the way they select

No matter which option is chosen, data integration and system interoperability are preserved and therefore, the overview of the data at the extended enterprise level is preserved.

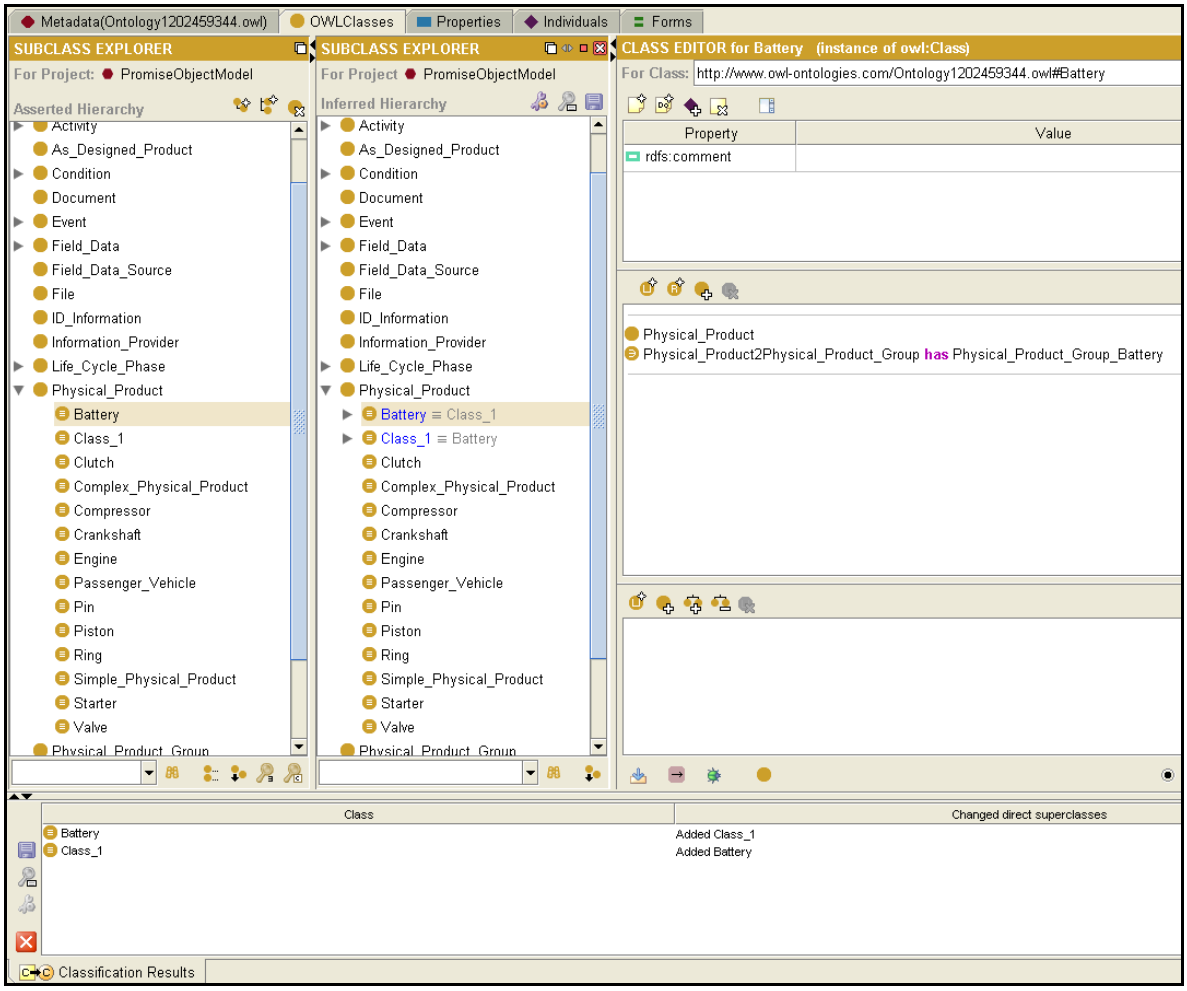

Figure 27: The reasoner re-classified the equivalent classes

Expressivity

The implementation of the ontology model and with the use of the DL-reasoner it has been more user-friendly for human agents and it has increased the ability of expressing the **Physical_Product** instances of the PLM model. The structure and the expressivity of the UML model allowed the engineers to have a very narrow view of the complexity of the physical products. They were limited to seeing only one level higher or one level lower of the physical product (Figure 28, Figure 29 and Figure 30). An example of how the DL-Reasoner understands the complex physical product *Passenger_Vehicle_1* is shown in

Figure 31. This provides the user with the big picture of what is happening inside the model. Answers to questions like: "which are all the physical products that belong to *Passenger_Vehicle_1*?", and in general: "from which physical products each complex physical product consists of?" were achieved. The three-level structure of complexity described in this example can be increased depending on the requirements of each case. This expressivity is also understood by the reasoner and it is used for the re-classification of the model.

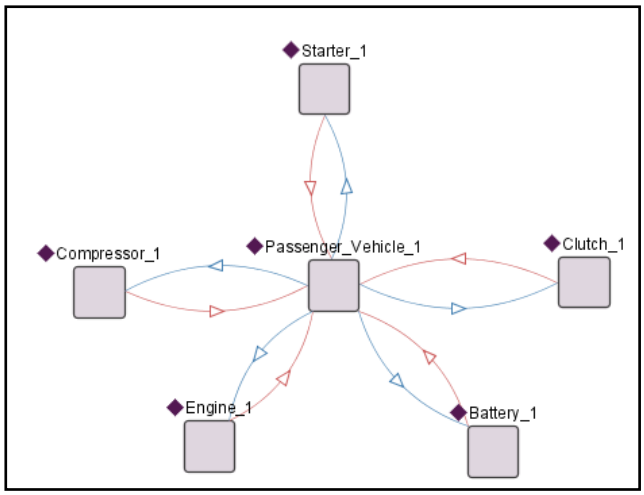

Figure 28: Instances related to *Passenger_Vehicle_1* instance with the properties hasParent and isParentOf.

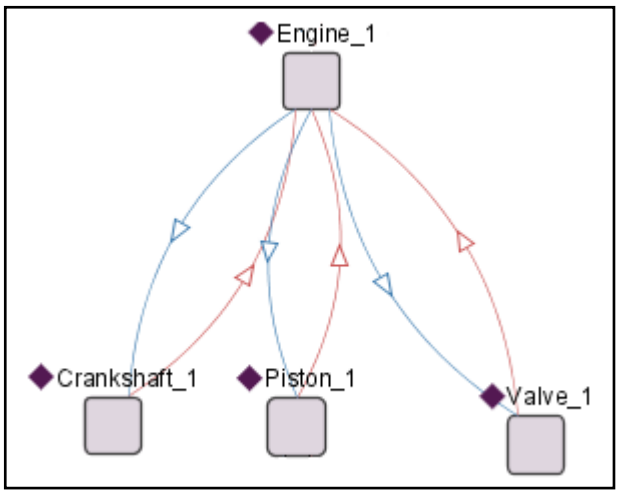

Figure 29: Instances related to *Engine_1* instance with the properties hasParent and isParentOf.

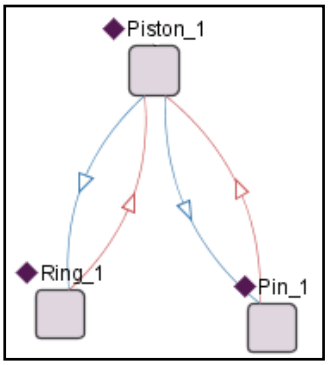

Figure 30: Instances related to *Piston_1* instance with the properties hasParent and isParentOf.

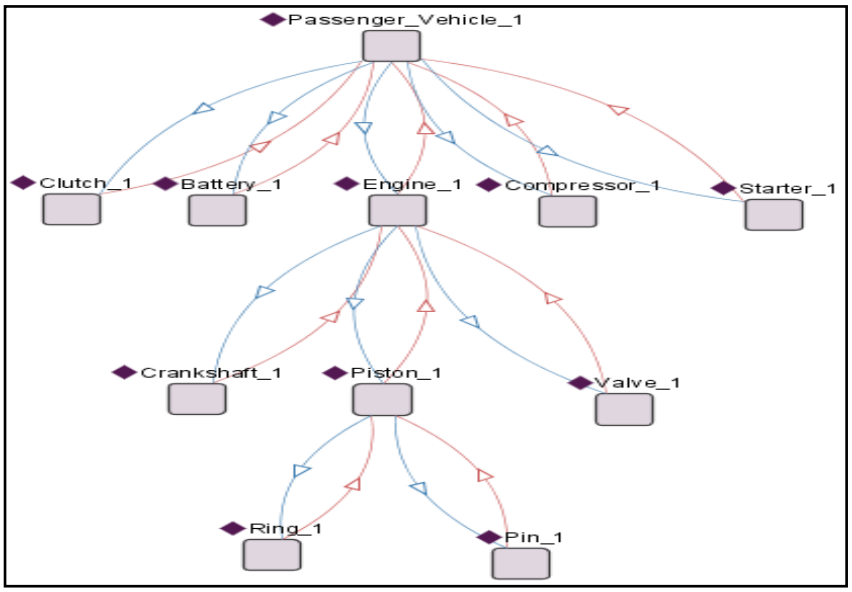

Figure 31: Instances related to *Passenger_Vehicle_1* instance directly and through inheritance due to the transitive properties hasParent and isParentOf.

Re-classification of extended model

Each sub-class added to the model has restrictions as well as relationships which it inherits from its super-class. These are the key elements for re-classification of the sub-classes by the DL-reasoner. In this case study the classes describing the various parts of the cars have been re-classified automatically into a three-level super-class/sub-class chain as shown in Figure 32. The DL-reasoner read the relationships between the rules of all the classes, it identified that the `hasParent`, `isParentOf` object properties are transitive and therefore, it re-classified the classes.

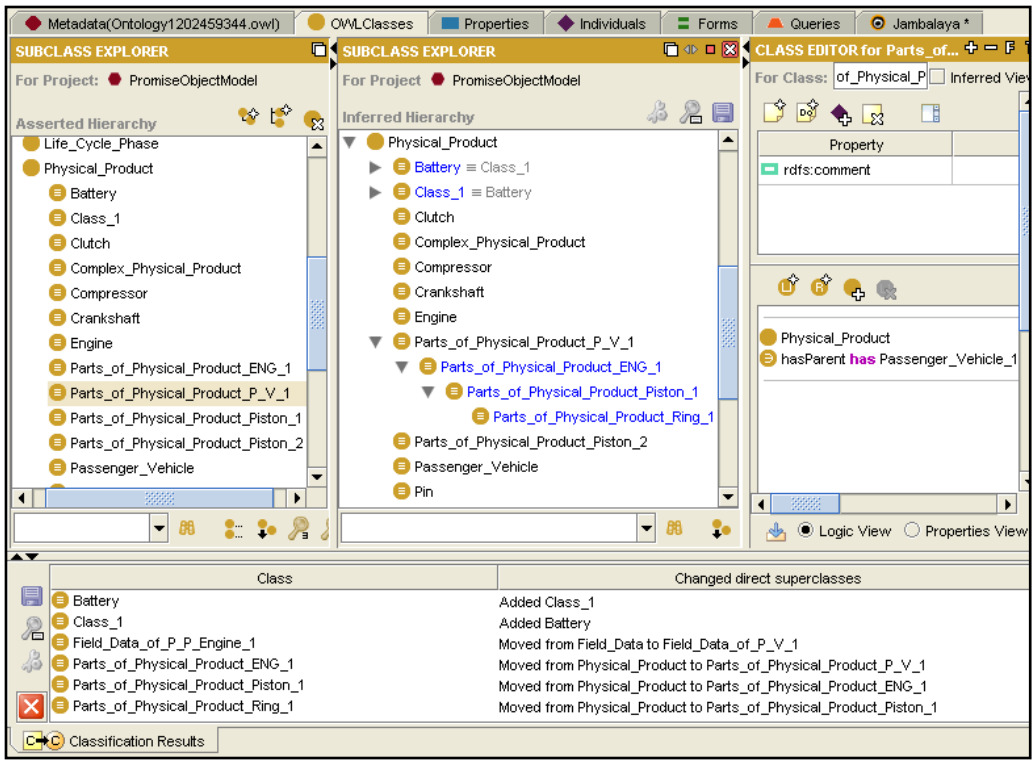

Figure 32: The DL-reasoner has re-classified the Parts classes.

5.1.5 Merging Ontologies

The importance of ontology merging has been well highlighted in section 4.3. The most important aims for the merging are: to achieve auto-mapping of the variations of the models; and to efficiently find out important beneficial elements in one model and use them across the models. In this section the model developed so far is used in a scenario in order to demonstrate how we achieved auto-mapping and how it is possible to share automatically parts of the models whenever required. It should be noted that the states described in this section do not claim to be exhaustive for every different use of the ontology based IT methods and tools, and they depend on the system architecture as well as on the use of DL rules. The scenario intends to be illustrative of possible problems which appear while merging one or more models together.

Demonstration Scenario

In this scenario the OEM has followed steps 1 (*U*=http://www.owl-ontologies.com/Ontology1202459344.owl), 2, 3 and 4. For step 4 the OEM has provided the paradigm and

the guidelines developed in section 5.1.3. Then, the OEM has made a copy of the ontology model (step 5), has changed the URI of the copy to *U'*=http://www.owl-ontologies.com/Ontology1202459344_ Copy_001.owl (step 6), has set as the default namespace of the URI *U* (step 7) and has distributed the models to the partner (step 8). Furthermore, the copy was populated and extended by the partner (step 9). After a month the OEM collects the copy (step 10) and imports it to the extended model developed according to the paradigm until section 5.1.4 (step 11). Thus, in step 12 the aim is that the OEM can use the reasoner (in the same way as it is used in 5.1.4) to reason efficiently the final model after merging. To make this aim real we had to add rules in order to merge Ontologies safely.

After the merging of two or more models is performed, under each class of the final model there are all the rules about this class which exist in all models. Regarding the classes of the final model there are three parameters which are important in order to perform reasoning: the class name, the DL rules of each class and the properties which are used by the rules. These three parameters may get various values. The important element is whether these parameters are the same or different in each one of the models. Thus, after merging there might appear the following eight cases regarding the classes and their DL rules which are summarised in Table 4. The information of the first four columns of this table is:

1. Two or more same named classes have the same DL rules about the same property.
2. Two or more same named classes have the same DL rules about different property.
3. Two or more same named classes have the different DL rules about the same property.
4. Two or more same named classes have the different DL rules about different property.
5. Two or more differently named classes have the same DL rules about the same property.
6. Two or more differently named classes have the same DL rules about different property.
7. Two or more differently named classes have different DL rules about the same property.
8. Two or more differently named classes have different DL rules about different property.

The remaining two columns contain information of how well the reasoner handles the data and the result in the final model regarding the initial classes. It should be noted that cases 1,

2, 3 and 4 are impossible to occur in one individual model since the editor does not allow the creation of a class which will be named the same as an existing class. However, these cases are created while merging two or more models which were developed on different machines.

In practice the overall system works well after the merging for all the cases except for the third case: in which exist classes with the same name with different DL rules about the same property. This is due to the fact that the model (of the paradigm in step 4) was not tested for merging. It was tested only with data and extension within one model. Therefore, the designers of the OEM had not considered adding restrictive rules in the upper classes in order to avoid possible conflicts after merging variations of the initial model.

Table 4: The possible cases after merging.

Case	Class Name	DL rules	Property	Handled	Result
1	Same	Same	Same	Yes	Same class, no changes
2	Same	Same	Different	Yes	Same class with all the restrictions
3	Same	Different	Same	No	Same class with all the restrictions: Problem
4	Same	Different	Different	Yes	Same class with all the restrictions
5	Different	Same	Same	Yes	Equivalent classes
6	Different	Same	Different	Yes	Different classes
7	Different	Different	Same	Yes	Different classes
8	Different	Different	Different	Yes	Different classes

Cases where the system Works well

In this section the examples are made by merging two models since the functionality and results are the same for cases of merging more than two models.

In the first case two classes with the same name have the same DL rules about the same property. In this case the final model contains a class with the common name which contains twice the same rule. For example, in the final model for the **Battery** class we have the following two rules. One from the first model:

Battery ≡ Physical_Product∩∋
Physical_Product2Physical_Product_Group.(Physical_Product_Group_Battery)

And one from the second model:

$$\text{Battery} \equiv \text{Physical\_Product} \cap \ni \text{Physical\_Product2Physical\_Product\_Group.(Physical\_Product\_Group\_Battery)}$$

These rules are well understood as the same rule by the reasoner. Thus, the final model works well for all: consistency, equivalencies, re-classification and the data instances are categorised under the right classes. It should be noted that these same rules become one rule in the case that we merge the two models using OWL 2.

In the second case two classes with the same names have the same DL rules about different properties. This may occur in cases of rules which use numbers i.e. cardinality restrictions. These rules are added in the class of the final model and the result is a class with all the restrictions. For example, a sub-class **Field_Data_Battery** of **Field_Data** class in one model might have maximum cardinality equal to 1 for the property `Field_Data2Physical_Product` and in another model might have maximum cardinality equal to 1 for the property `Field_Data2Valid_Field_Data_Type`. Thus, in this case the class becomes more restricted than in the initial models and hence, new knowledge/value is created for this class.

In the fourth case two classes with the same names have different DL rules about different properties. This may occur with any properties of the classes. The rules are automatically added in the class of the final model and the result is a class with all the restrictions. For example, a sub-class **Field_Data_Battery** of **Field_Data** class in one model might have maximum cardinality equal to 1 for the property `Field_Data2Physical_Product` and in another model the same sub-class might have an existential restriction (at least one) for the property `Field_Data2Valid_Field_Data_Type`. Thus, in this case the class becomes more restricted than in the initial models and hence, new knowledge/value is created for this class.

In the fifth case two classes with different names have the same DL rules about the same property. In this case the system behaves as expected and identifies the equivalencies, in the same way as it has been demonstrated in section 5.1.4 and Figure 27.

In the sixth case two classes with different names have the same DL rules about different properties. This may occur in cases of rules which use numbers i.e. cardinality restrictions. In this case the system considers the classes as different and the DL rules of the initial classes remain unchanged.

In the seventh and eighth cases the system considers the classes as different and the DL rules of the initial classes remain unchanged.

These seven cases are well understood by the reasoner and hence, it is utilised in the final model to provide consistency check, to figure out equivalencies, to perform re-classification of the class-hierarchy and to categorise the data instances under the right classes.

Possible Weakness

The weakness appeared in the third case. In this case the reasoner, using the current structure, does not handle cases of having two classes with the same name but with different semantics and DL rules applied on the same property. *It should be noted that in normal circumstances the result of the reasoner is logical and creates a class with the total number of rules about this class.* For instance, if there are contradicting rules, (i.e. maximum cardinality equal to 1 and cardinality exactly equal to 2 for the same property on the same class) then, the reasoner understands the inconsistency and points out the source of the error. *However, in our structure the reasoner creates undesirable cases such as: a class contains batteries and engines.* For example, we have that **Class_1** class for the one model is defined as a class containing engines:

Class_1≡Physical_Product∩∋
Physical_Product2Physical_Product_Group.(Physical_Product_Group_Engine)

Whereas, for the other model **Class_1** class is defined as a class containing batteries:

Class_1≡Physical_Product∩∋
Physical_Product2Physical_Product_Group.(Physical_Product_Group_Battery)

In this case the reasoner understands that **Class_1** class contains both engines and batteries. In fact it understands that **Class_1** is equivalent to **Battery** and to **Engine**. This

is an abnormal situation but it is possible to happen since the names Class_1, Class_2, etc. are generated automatically by the Protégé editor. The result is shown in Figure 33.

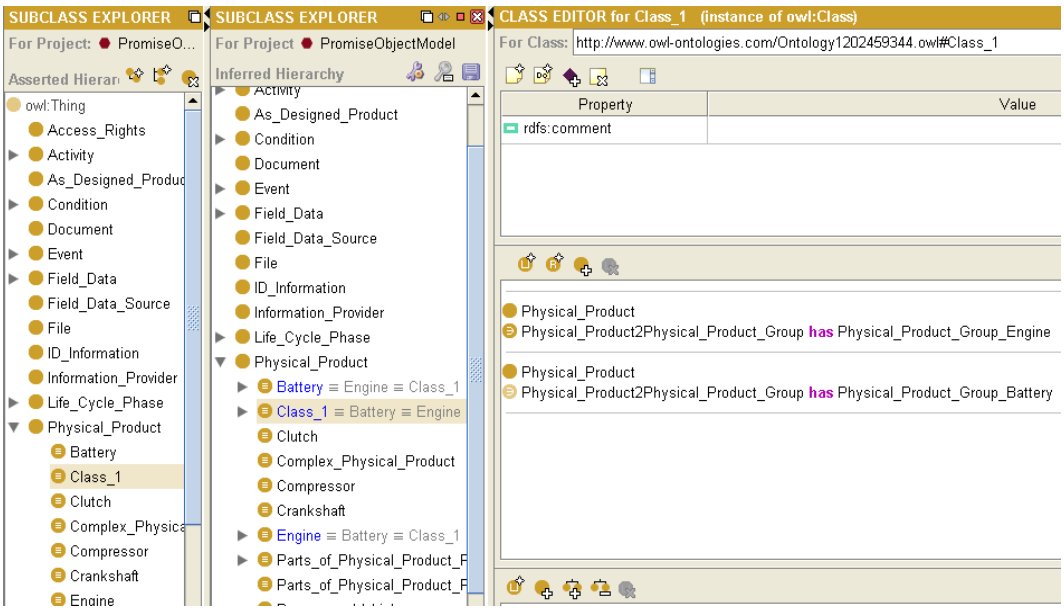

Figure 33: The result of the reasoner after merging.

The question is: how to use the DL rules and the reasoner in order to handle this and similar cases? There are more than one possible solutions to this question and our aim is to provide a solution as much generic as possible. Actually, similar limitations are a part of the unsolved problems for achieving automated modular use of Ontologies and there is research in this direction (for more details see also section 2.2.6 and [56]).

Rules supporting Safe Merging

In this section we are providing a solution for handling the case 3 after merging. Actually, we need to use the DL rules in a way that the reasoner can understand that the above situation is abnormal and warn the user about it.

A simple solution is to set a cardinality restriction on the class **Physical_Product** for the relationship `Physical_Product2Physical_Product_Group`. This limits the number of the DL rules of a sub-class which connect it to the **Physical_Product_Group** class. Therefore, we have set the restriction of Cardinality exactly=1 for the relationship `Physical_Product2Physical_Product_Group` on the **Physical_Product** class. It should be noted that all instances of the **Physical_Product_Group** class have

been declared as "All Different" to each other because otherwise they "could" simply denote the same individual (this derives from the OWA). In this way it is understood by the reasoner that the instance *Physical_Product_Group_Battery* is different from the instance *Physical_Product_Group_Engine*.

After adding this restriction we study how the reasoner handles this case. In the final model we have that **Class_1** has the rules:

Class_1≡Physical_Product∩∋
Physical_Product2Physical_Product_Group.(Physical_Product_Group_Engine)

And:

Class_1≡Physical_Product∩∋
Physical_Product2Physical_Product_Group.(Physical_Product_Group_Battery)

This is breaking the cardinality restriction, since **Class_1** is related to the **Physical_Product_Group** with two different rules. This is read by the reasoner and in the consistency checking it notifies that the model is inconsistent. This is shown in Figure 34. Then, it is important to know "where is the inconsistent concept in the model". Using the pellet reasoner this is notified indirectly when one attempts to compute inference (categorise the instances under the classes). There is a warning each time there is an inconsistency (Figure 35). For example in Figure 35 the explanation says that the reason for the inconsistency is that "*individual Battery_3 has more than one values for property Physical_Product2Physical_Product_Group violating the cardinality restriction*". Thus, the user understands that a cardinality restriction is violated on the relationship `Physical_Product2Physical_Product_Group` and the instance that is trying to do the violation is the instance of the **Physical_Product** class which is named *Battery_3*.

This method is efficient and applicable since only the OEM collects the copies of the models and it sets the restrictions on the upper classes of the model. Actually, this method allows importing one model inside the other since merging is not allowed in OWL 1. In the case that the merging is performed using OWL 2 and the rules are homogenised under one ontology model, then the reasoner specifies exactly the class that is inconsistent. This works even if the ontology is built in the OWL 1.

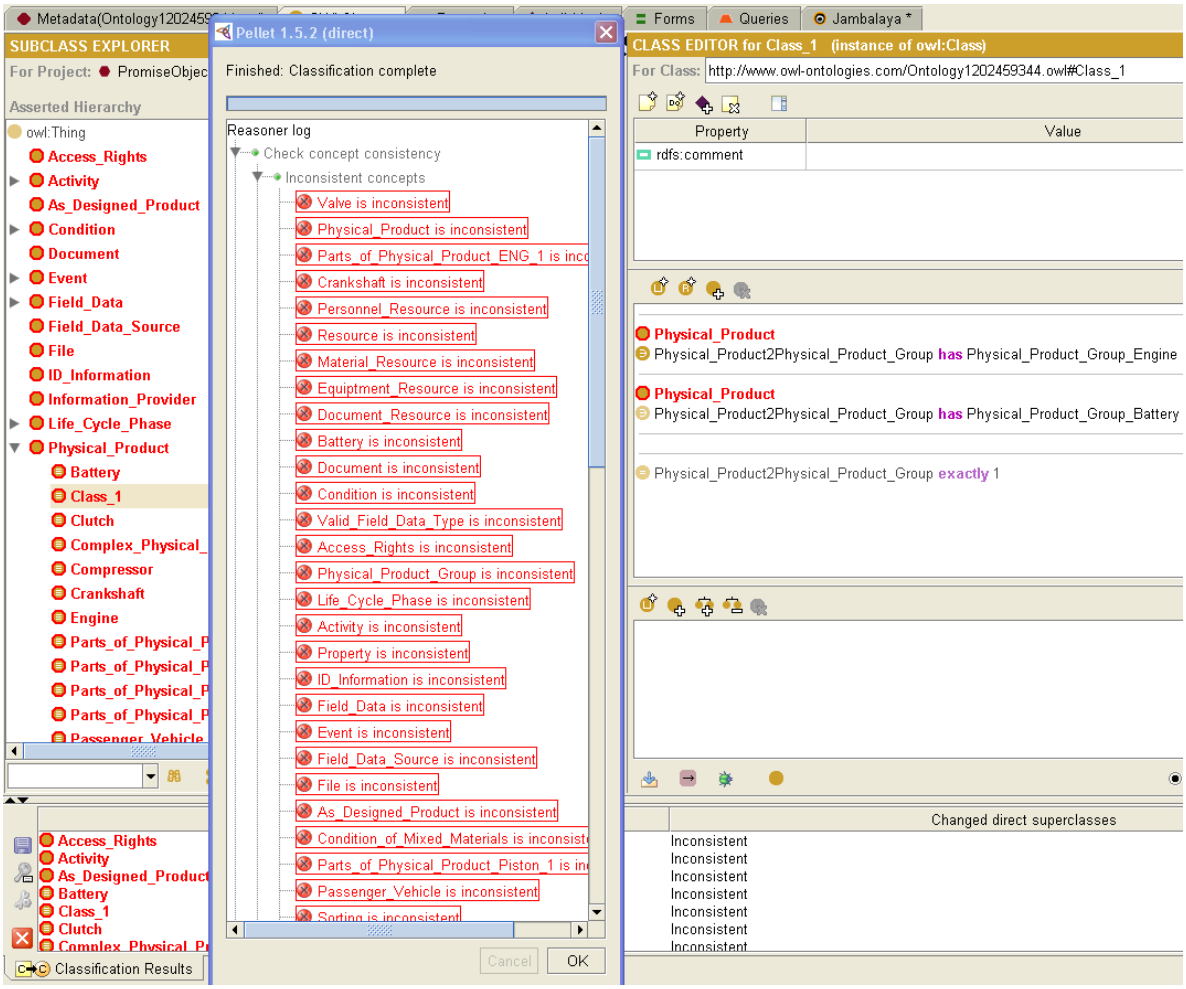

Figure 34: The reasoner shows that the model is inconsistent.

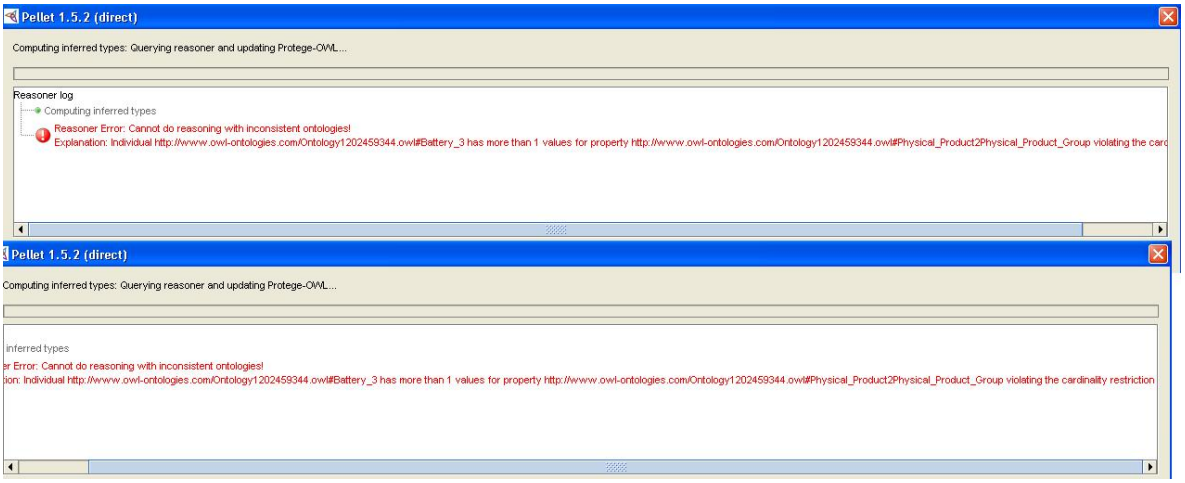

Figure 35: The reasoner provides an explanation of the inconsistency.

5.1.6 Discussion of the Case Study 1

This case study has demonstrated an application of the implementation method described in section 4.6. After implementing the method in the model, a number of applications have been demonstrated which altogether provide solutions towards Closed-Loop PLM. The initial model developed in section 4.2 was extended with sub-classes which had DL restrictions. Having all partners starting from the model of section 4.2, extension with the use of semantics supports data integration and system interoperability through semantics. The use of DLs allows checking the extended ontology model for its consistency and provides class re-classification. As it is shown in paragraph 5.1.4 any new sub-class will be re-classified to its logical position. Moreover, the usefulness of inference is demonstrated; the overall result of the extended model achieves simplicity for MOL engineers-maintenance crew where they may insert data simply at the generic classes and then, data are auto-categorised at the right place by the reasoner. This is of great significance since the MOL actors don't have to change their way of inserting data in comparison to the way they used before extending the model. In paragraph 5.1.3 an example of such use in practice is demonstrated according to various criteria. Thus, it is no longer necessary for the user to know the exact detailed structure of the model and the system automatically prevents the creation of data miss-location or of data duplicates. Finally, merging of two variations of the initial model was achieved and it has been demonstrated how to support the merging by adding simple rules on the upper initial classes of the model.

The implementation of ontologies and the use of DLs provide the following functionalities:

1. The model handles multiple data from multiple physical products by applying DL rules.
2. The model is extensible through the DL rules.
3. The DL-reasoner understands the DL rules and checks the extended ontology model for its consistency.
4. Concept equivalencies-inconsistencies are efficiently handled by applying DL rules, supporting system interoperability and data integration. In the case of merging some more DL rules had been added.

5. The DL-reasoner provides class re-classification. As it is shown in paragraph 5.1.4 any new sub-class will be re-classified to its logical position.
6. The DL-reasoner provides class equivalencies. As it is shown in paragraph 5.1.4 any new sub-class will be re-classified to its logical position.
7. The DL-reasoner reads the DL rules and according to them, infers instances at the logical position in the class-hierarchy (5.1.3).
8. While ontology merging of the different variations of the initial model, the DL-reasoner provides auto-mapping of the models as well as the means for exploiting the total number of rules and knowledge.

It should be noted that the solution provided for achieving the merging, requires further elaboration to develop generic methods and guidelines for model development in order to support merging.

5.2 Case Study 2

This case study demonstrates an application of the "*Duration of Time*" concept (section 4.5 and Figure 20) on an ALM/PLM ontology model, highlighting the capabilities of the final model. A summary of this case study has already been published by Matsokis et al. [137]. The model used is based on the SOM as it is shown in Figure 18 (developed by Matsokis et al. [135]) to which the "*Duration of Time*" concept has been implemented following the steps described in section 4.5.2. The SOM has been made a sub-class of duration of time class (Figure 36) and has been extended to facilitate the case study. It describes the maintenance activities of locomotives and also includes some parts of the model such as documents which engineers are not used treating (seeing) from the time point of view. The importance of this vision is that documents and data change or are changed for technical (or for other maintenance) purposes over time. In this way there is one system monitoring all the elements on a time basis. Even if the parts, equipments, etc. are using different information systems, these systems are easily synchronised and organised together.

Figure 36: PDKM SOM as it is in the Time Centric PLM

The case study describes the application of the model by an authorised locomotive maintenance provider (MP). The MP is specialised on one model/type of locomotives. The MP has two maintenance platforms: Platform A and Platform B; each one has one machine to aid maintenance: Machine A and Machine B; and one mechanic which performs the maintenance on each platform: Mechanic A and Mechanic B; each mechanic uses one tool-box: Tool-Box A and Tool-Box B; and there are 5 documents: Document 1, Document 2, Document 3, Document 4 and Document 5. Document 1 contains the field data from the locomotive and it is updated each time the locomotive visits an authorised MP (one per

locomotive, for this reason we have Document 1a, Document 1b, etc.). Document 2 contains the maintenance history of the Locomotive and it is updated each time the locomotive enters the maintenance (one document per locomotive having a, b, c and d, similarly to Document 1). Document 3 contains the manufacturer's guidelines for performing/ operating maintenance according to the working hours of the locomotive or to the period of time passed since the last maintenance. Document 4 contains the manufacturer's instructions with schemas for removing and replacing parts. Document 5 contains the information about the stock of the spare-parts. To facilitate and to categorise better the data for this application the model was extended accordingly. The developing process was:

- The class **Duration of Time** was made the super-class of the model.
- A time framework for the existing ontology PLM was developed. This framework is introduced in the **Duration of Time** class and has the only "time" properties of the ontology (start_date_time, end_date_time, duration). Thus, all classes and sub-classes of the ontology have the same "time" framework.
- A central reference time CET was chosen. In this way, misunderstandings concerning time in communication between different agents around the globe will be avoided.
- The model was extended to facilitate the case study
- Instances are stored for every physical product, activity, event, process, resource etc. necessary.

The SOM has now become a sub-class of time (Figure 36) and it is extended with several classes to facilitate the resources, their data and activities. The extended part of the class-hierarchy is shown in Figure 37. For the case study we have only three locomotives involved, Locomotive No1, No2 and No3.

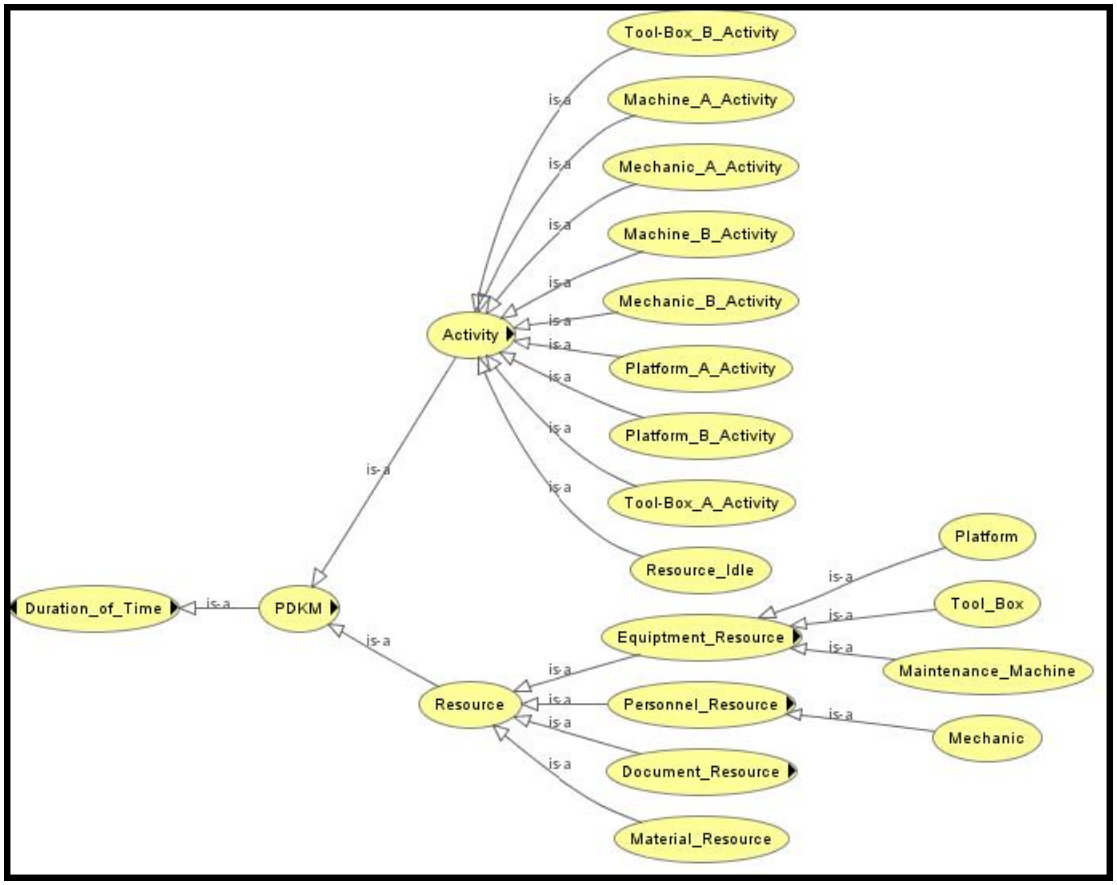

Figure 37: Ontology model extended with necessary classes

5.2.1 System Analysis and Functionality

In this scenario locomotives are visiting the MP with an appointment. The *duration of time* for each resource, activity, etc. is shown in Figure 38. The three colours in the rows are referring to Locomotive No1, No2 and No3 accordingly and show for which locomotive and for how long is each resource used. This could be referring to the future (daily/weekly/monthly etc. schedule according to appointments). All the uncoloured cells of each row represent the time that the resource related to this row is in idle status. Each column represents 5 minutes. These time periods of 5 minutes could have been time periods of any required type such as years, months, days, hours, minutes, seconds, milliseconds.

In Figure 38 we have that Locomotive No1 arrives to the service department and Mechanic A is responsible for it. He updates Document 1a with field data from the locomotive's on-board computer unit and he checks Document 2a which contains its maintenance history. Then, according to the status of the locomotive he reads the manufacturer's guidelines for

this type of locomotive to see the maintenance activities to be performed and decides to replace some parts. He checks document 5 to see if there is any in the local stock and document 4 for the replacing instructions. Similar are the activities for Locomotives No2 and No3 shown in Figure 38 (for Locomotive No2 there is no need to remove/replace parts and Locomotive No3 arranges an appointment out of schedule). In case the MP provides multiple maintenance sites Locomotive No3 would have chosen the closest, soonest available maintenance site. Documents like all resources are seen as *duration of time* elements which appear in the system when they are used.

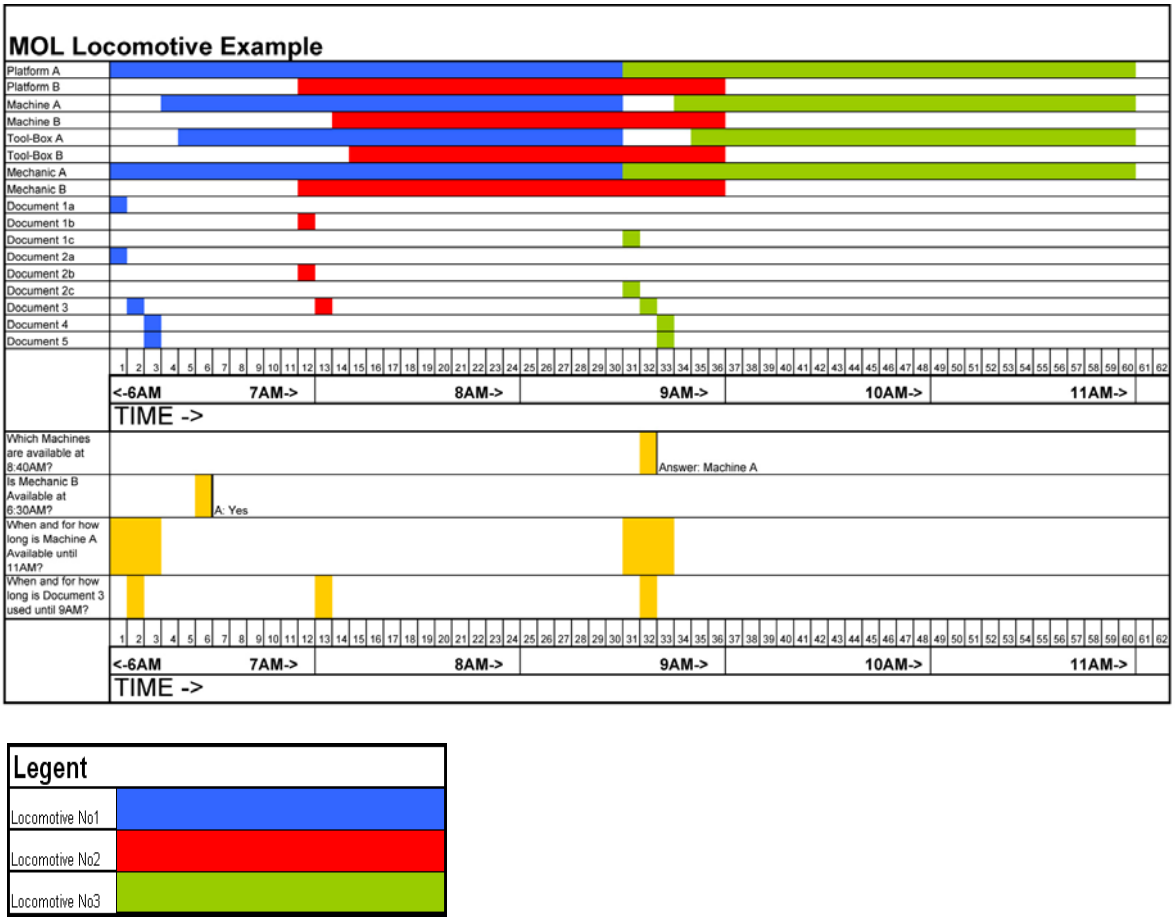

Figure 38: MOL Locomotives as seen from the "duration of time" Point of view with Queries

Using the "*Duration of Time*" approach provides engineers with all the necessary information for the state of each resource at every moment. Engineers can have information according to Which-queries such as "Which machines are available at this time slot?" which is equivalent to "Who is in stand by status at this time" and returns all the non-active

values at that "*Duration of Time*", or according to Availability-queries such as "Is Mechanic A available at a certain time?" or "When and for how long is a certain resource (mechanic or machine or document) available?" which return instances showing availability. This information is used for the best management of the resources. Moreover, the system also provides the information of the duration of time a Locomotive is using each resource.

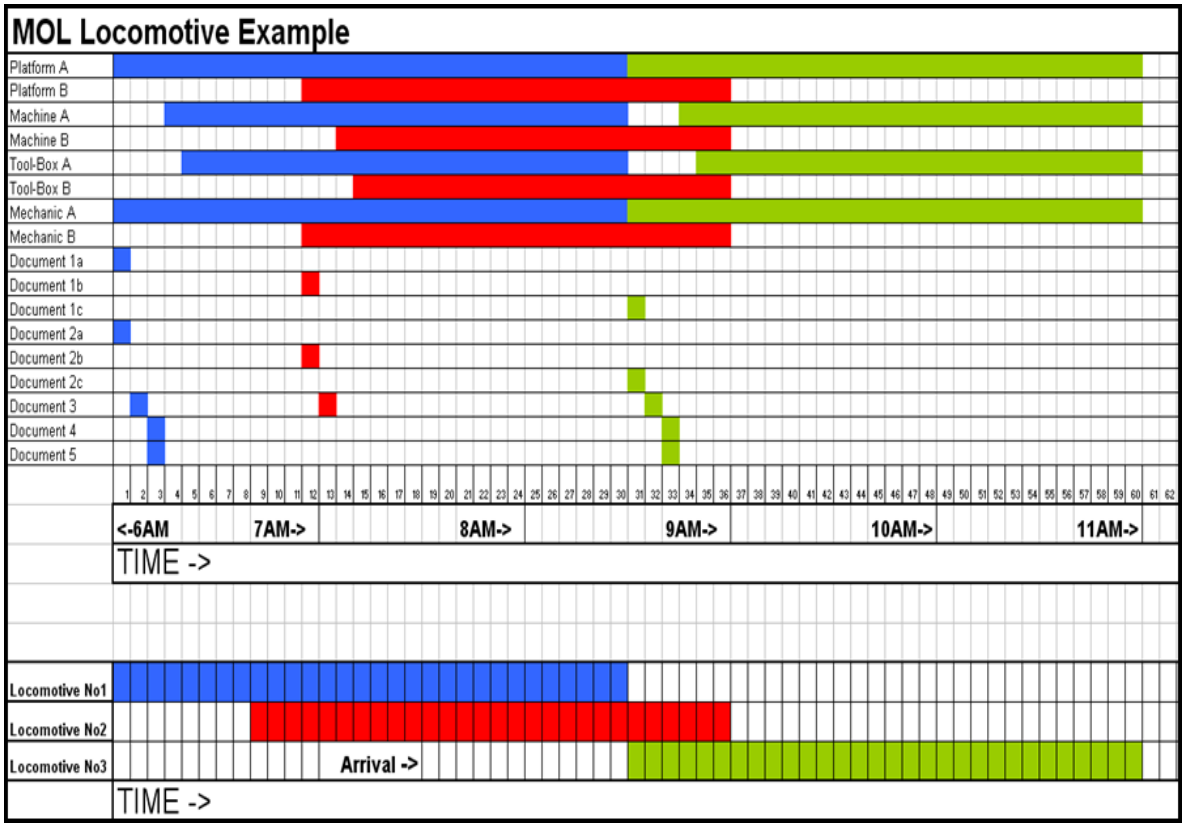

Figure 39: The example of MOL Locomotives along time.

In Figure 38 several examples of the queries are shown. Firstly, a Which-Query is shown, which is applied on the model about the machine and describes "Which Machine(s) is (are) available right now (now=8:40 AM) and for how long?". It returns the idle instance(s) of the available resources or nothing if the resources are not available. Secondly, there is the query "Is Mechanic B available right now (now=6:30AM)?" is shown. This query applies only to the certain resource instance (the query could be more generic like "who is available at this time?") and returns either the idle instance if the resource is available or nothing if the resource instance is not available. Furthermore, Figure 38 shows an example

of "When and for how long is Machine A available until 11am?" query. This query applies to all instances of Machine A and returns the idle instances of Machine A. Finally, an example of "When and for how long is Document 3 used?" query is shown; returning all the time slots during which Document 3 is being used. Furthermore, the system contains the information of the overall availability-usage of resources and maintenance time of the locomotives. The maintenance time per locomotive on real time is shown in Figure 39.

We obtained similar results using SQWRL to query the knowledge base (the list of the rules is provided at Appendix C). In Figure 40 an example of a Which-Query is shown. The query is applied on the machine and describes "Which Machine(s) is available right now (now=8:40 AM) and for how long?". It returns the idle instance of the available resources or nothing if the resources are not available.

SQWRLQueryTab | Which_Machine_is_Available_at_8_40AM

Instance	Machine	Birth Date	End Date	Duration In Minutes
Resource_Idle_73	Maintenance_Machine_A	2009-02-27T08:30:00	2009-02-27T08:45:00	15

Figure 40: An example of Which Queries.

Figure 41 shows an example of "Is Mechanic B available right now (now=6:30AM) and for how long from now?" query. This query applies only to the certain resource instance (the query could be more generic like "who is available at this time?") and returns either the idle instance if the resource is available or nothing if the resource instance is not available.

SQWRLQueryTab | Is_Mechanic_B_Available_at_6_30AM

Instance	Birth Date	End Date	Duration From Now In Minutes
Mechanic B	2009-02-27T06:00:00	2009-02-27T06:55:00	25

Figure 41: An example of Availability on certain time Queries.

Figure 42 shows an example of "When and for how long is Document 3 used?" query. This query applies to all instances and returns the result shown in Figure 42. Figure 43 shows an example of "When and for how long are any resources available?" query. This query applies to all instances and returns the result shown in Figure 43. This figure is only a snapshot and doesn't contain all the results. All the results are shown in Figure 44 after exported to excel.

SQWRLQueryTab | When_n_For_How_Long_Document_3

Instance	Resource	Birth Date	End Date	Duration In Minutes
Resource_Idle_95	Document_Resource_3	2009-02-27T06:00:00	2009-02-27T06:05:00	5
Resource_Idle_96	Document_Resource_3	2009-02-27T06:10:00	2009-02-27T07:00:00	50
Resource_Idle_97	Document_Resource_3	2009-02-27T07:05:00	2009-02-27T08:35:00	90
Resource_Idle_98	Document_Resource_3	2009-02-27T08:40:00	2009-02-27T16:00:00	440

Figure 42: An example of Document 3 Availability; (when and for how long) Queries.

SQWRLQueryTab | When_n_For_How_Long_Resources

Instance	Resource	Birth Date	End Date	Duration In Minutes
Resource_Idle_85	Document_Resource_1a	2009-02-27T06:05:00	2009-02-27T16:00:00	595
Resource_Idle_86	Document_Resource_1b	2009-02-27T06:00:00	2009-02-27T06:55:00	55
Resource_Idle_87	Document_Resource_1b	2009-02-27T07:00:00	2009-02-27T16:00:00	540
Resource_Idle_88	Document_Resource_1c	2009-02-27T06:00:00	2009-02-27T08:30:00	150
Resource_Idle_89	Document_Resource_1c	2009-02-27T08:35:00	2009-02-27T16:00:00	445
Resource_Idle_90	Document_Resource_2a	2009-02-27T06:05:00	2009-02-27T16:00:00	595
Resource_Idle_92	Document_Resource_2b	2009-02-27T07:00:00	2009-02-27T16:00:00	540
Resource_Idle_91	Document_Resource_2b	2009-02-27T06:00:00	2009-02-27T06:55:00	55
Resource_Idle_94	Document_Resource_2c	2009-02-27T08:35:00	2009-02-27T16:00:00	445
Resource_Idle_93	Document_Resource_2c	2009-02-27T06:00:00	2009-02-27T08:30:00	150
Resource_Idle_98	Document_Resource_3	2009-02-27T08:40:00	2009-02-27T16:00:00	440
Resource_Idle_97	Document_Resource_3	2009-02-27T07:05:00	2009-02-27T08:35:00	90
Resource_Idle_96	Document_Resource_3	2009-02-27T06:10:00	2009-02-27T07:00:00	50
Resource_Idle_95	Document_Resource_3	2009-02-27T06:00:00	2009-02-27T06:05:00	5
Resource_Idle_99	Document_Resource_4	2009-02-27T06:00:00	2009-02-27T06:10:00	10
Resource_Idle_100	Document_Resource_4	2009-02-27T06:15:00	2009-02-27T08:40:00	145
Resource_Idle_101	Document_Resource_4	2009-02-27T08:45:00	2009-02-27T16:00:00	435
Resource_Idle_103	Document_Resource_5	2009-02-27T06:15:00	2009-02-27T08:40:00	145
Resource_Idle_102	Document_Resource_5	2009-02-27T06:00:00	2009-02-27T06:10:00	10
Resource_Idle_104	Document_Resource_5	2009-02-27T08:45:00	2009-02-27T16:00:00	435
Resource_Idle_72	Maintenance_Machine_A	2009-02-27T06:00:00	2009-02-27T06:15:00	15

Figure 43: An example of MOL phase-Availability (when and for how long) Queries.

Instance	Resource	Birth Date	End Date	Duration In Minutes
Resource_Idle_85	Document_Resource_1a	"2009-02-27T06:05:00"	"2009-02-27T16:00:00"	595
Resource_Idle_86	Document_Resource_1b	"2009-02-27T06:00:00"	"2009-02-27T06:55:00"	55
Resource_Idle_87	Document_Resource_1b	"2009-02-27T07:00:00"	"2009-02-27T16:00:00"	540
Resource_Idle_88	Document_Resource_1c	"2009-02-27T06:00:00"	"2009-02-27T08:30:00"	150
Resource_Idle_89	Document_Resource_1c	"2009-02-27T08:35:00"	"2009-02-27T16:00:00"	445
Resource_Idle_90	Document_Resource_2a	"2009-02-27T06:05:00"	"2009-02-27T16:00:00"	595
Resource_Idle_92	Document_Resource_2b	"2009-02-27T07:00:00"	"2009-02-27T16:00:00"	540
Resource_Idle_91	Document_Resource_2b	"2009-02-27T06:00:00"	"2009-02-27T06:55:00"	55
Resource_Idle_94	Document_Resource_2c	"2009-02-27T08:35:00"	"2009-02-27T16:00:00"	445
Resource_Idle_93	Document_Resource_2c	"2009-02-27T06:00:00"	"2009-02-27T08:30:00"	150
Resource_Idle_98	Document_Resource_3	"2009-02-27T08:40:00"	"2009-02-27T16:00:00"	440
Resource_Idle_97	Document_Resource_3	"2009-02-27T07:05:00"	"2009-02-27T08:35:00"	90
Resource_Idle_96	Document_Resource_3	"2009-02-27T06:10:00"	"2009-02-27T07:00:00"	50
Resource_Idle_95	Document_Resource_3	"2009-02-27T06:00:00"	"2009-02-27T06:05:00"	5
Resource_Idle_99	Document_Resource_4	"2009-02-27T06:00:00"	"2009-02-27T06:10:00"	10
Resource_Idle_100	Document_Resource_4	"2009-02-27T06:15:00"	"2009-02-27T08:40:00"	145
Resource_Idle_101	Document_Resource_4	"2009-02-27T08:45:00"	"2009-02-27T16:00:00"	435
Resource_Idle_103	Document_Resource_5	"2009-02-27T06:15:00"	"2009-02-27T08:40:00"	145
Resource_Idle_102	Document_Resource_5	"2009-02-27T06:00:00"	"2009-02-27T06:10:00"	10
Resource_Idle_104	Document_Resource_5	"2009-02-27T08:45:00"	"2009-02-27T16:00:00"	435
Resource_Idle_72	Maintenance_Machine_A	"2009-02-27T06:00:00"	"2009-02-27T06:15:00"	15
Resource_Idle_73	Maintenance_Machine_A	"2009-02-27T08:30:00"	"2009-02-27T08:45:00"	15
Resource_Idle_74	Maintenance_Machine_A	"2009-02-27T11:00:00"	"2009-02-27T16:00:00"	300
Resource_Idle_75	Maintenance_Machine_B	"2009-02-27T06:00:00"	"2009-02-27T06:55:00"	55
Resource_Idle_76	Maintenance_Machine_B	"2009-02-27T09:00:00"	"2009-02-27T16:00:00"	420
Resource_Idle_82	Mechanic_A	"2009-02-27T11:00:00"	"2009-02-27T16:00:00"	300
Resource_Idle_83	Mechanic_B	"2009-02-27T09:00:00"	"2009-02-27T16:00:00"	420
Resource_Idle_84	Mechanic_B	"2009-02-27T06:00:00"	"2009-02-27T06:55:00"	55
Resource_Idle_69	Platform_A	"2009-02-27T11:00:00"	"2009-02-27T16:00:00"	300
Resource_Idle_70	Platform_B	"2009-02-27T06:00:00"	"2009-02-27T06:55:00"	55
Resource_Idle_71	Platform_B	"2009-02-27T09:00:00"	"2009-02-27T16:00:00"	420
Resource_Idle_79	Tool_Box_A	"2009-02-27T11:00:00"	"2009-02-27T16:00:00"	300
Resource_Idle_78	Tool_Box_A	"2009-02-27T08:30:00"	"2009-02-27T08:50:00"	20
Resource_Idle_77	Tool_Box_A	"2009-02-27T06:00:00"	"2009-02-27T06:20:00"	20
Resource_Idle_80	Tool_Box_B	"2009-02-27T06:00:00"	"2009-02-27T07:10:00"	70
Resource_Idle_81	Tool_Box_B	"2009-02-27T09:00:00"	"2009-02-27T16:00:00"	420

Figure 44: The result of MOL phase-Availability (when and for how long) Query of Figure 43 exported to excel.

5.2.2 Discussion of the Case Study 2

The scope of this case study is to demonstrate the behaviour of the system after implementing the "*Duration of Time*" concept. It describes the maintenance of locomotives and the description also includes some parts of the model such as documents which engineers are not used treating (seeing) them from the time point of view.

The implementation of the "*Duration of Time*" concept has two aims: to preserve the continuity of information along time (i.e. changes on information and usage of information); and to provide the basis for synchronising different information systems no matter the different products they are tracking (i.e. documents, components, machines, etc.) or the semantics they are using. The importance of the implementation is that once all

different systems are based on the "*Duration of Time*" concept, all information contained on them may be plotted along time. Thus, data from different information systems are easily synchronised and organised together.

This case study has demonstrated that the initial model has been made simpler with the implementation of the "*Duration of Time*" concept. This is due to the fact that the time attributes are unified and in case of model extension these attributes are inherited by the new classes. A number of applications have shown that the system provides complete data visibility and hence, inter-OEMs/Suppliers co-operation for better resources exploitation. Documents like all resources are seen as duration of time elements which appear in the system when they are used. Under this perspective one can have an overview of all documents, resources, etc. of all systems. Using similar queries, engineers are provided with a complete overview of the time slots and they are supported in decision making for optimal management of resources, activities, agents and processes. Moreover, the entire model has been described by the Duration of Time concept and still keeps its previous functionalities. Finally, through time it is very simple to track system or data changes and thus, keep track of all the past states of all the parts of the system. The functionalities are summarised as:

- The concept is easily implemented in PLM models using current technology.
- The models maintain their initial functionalities.
- The system provides complete data visibility.
- This visibility functions also under multi-system circumstances.
- Systems based on the concept may be easily synchronised.
- Optimal management of resources, activities, agents, information and processes through complete overview of the time slots.
- Inter-OEMs/Suppliers co-operation for better resources exploitation through synchronisation.

- Documents or other elements which originally did not have any time data, are now seen as duration of time elements which appear when they are used. Under this perspective one can have an overview of all documents of all systems.

5.3 Case Study 3

This case study combines the use of the ontology based IT methods and tools as they are used is section 5.1 with the implementation of the "*Duration of Time*" concept as shown in section 5.2. It aims at demonstrating several of the new capabilities provided by the ontology model as well as at providing a wider aspect and aid understanding of the benefits of applying time-based models in PLM.

The SMAC-Model is generic and extensible to fulfil the user's requirements. The extension of the initial model is required to be performed using DL rules in order to maintain the interoperability and data integration among the variations of the initial model. Then, the DL-reasoner may be used in order to find equivalencies, consistency and re-classification on classes and to categorise instances. Moreover, to this model we have implemented the "*Duration of Time*" concept. Thus, the model may be easily synchronised with other models using the concept and all the different data stored in the model may be represented along the lifecycle. In the next section a case study presents a number of capabilities of the developed system.

The concept is that the industrial partner uses the SMAC ontology model (Figure 19) to facilitate the information and the data about a lathe machine. The data describes the machine as it is manufactured (BOM, functions, etc.) as well as its lifecycle. The initial SMAC-Model model is extended to facilitate better the user's needs depending on the usage and the type of the machine. In the long term the industrial partner provides its maintenance groups with copies of the ontology model, in order to facilitate the lifecycle data of each machine. Then, the industrial partner collects the different copies and merges the different elements of these copies under one single ontology model in order to have an overview of the status, maintenance, faults etc. of all the machines. A summary of the results of this case study can be found in [138] and in [139].

5.3.1 Functionality

The functionality of the SMAC-Model (Figure 19) is quite simple. Firstly, the list of the physical products is stored in the **Physical_Product** class. The physical products may be complex products which consist of many parts such as vehicles or simple which consist of only one part such as screw. The complexity of the product and its parts is described through a "physical product to physical product" object property `hasParent` and its inverse `isParentOf`. Depending on the requirements of the application, the level of detail which is considered as "simple" may vary. Even for the same product, in different cases, one might have different levels of detail: i.e. the level of detail is different for products of a fleet management company and different for a single user who might be interested to have more detailed model for the one product he is using. The properties of the products are stored in the **Property** class and the **ID_Info** class. The **Physical_Product** is also related to the **Function** class in order to store the (one or more) functions (i.e. rotation, linear movement, store coolant liquid, etc.) a certain product may have. Therefore, whenever a physical product is degraded one or more of its functions are affected. Through the connection of this class with the **Physical_Product** we are able to know which functions are related to each individual physical product and they are or may be affected during its degradation. Furthermore, each physical product is related to the **Life_Cycle_Phase** class which enables each product to be related to one or more instances of a lifecycle phase i.e. to multiple instances of the MOL. This relationship combined with the object properties `hasParent`/`isParentOf` allows the information system to track information about the product through its different phases (and types of usage) and therefore, preserve continuity of information about the physical product. Thus, the model stores information about which data is related to the product for each of its use. For each use there is also a certain user of the **User** class. During its lifecycle the product is monitored with sensors which collect valuable data of different types such as temperature, pressure, velocity, viscosity etc. in various measurement units such as Celsius, bar, m/sec, Pascal-second respectively. The different sensors related to the product are stored in the **Field_Data_Source** class and the types of the data collected are stored in the **Valid_Field_Data_Type** class. The collected data from the sensors is stored in the **Field_Data** class and in documents if necessary. In the **Condition** class it is stored a list

of the required or recommended conditions for the well-functioning of the products. These conditions may vary depending on the product and are created according to the advice of the experts. Then, the data of the **Field_Data** class is compared with the conditions. If one or more conditions are not met, one or more events are created and stored in the **Event** class. Some of the events may be categorised as alarms (instances of the **Alarm** class). Data stored under the **Alarm** class are all the critical events which might also affect the durability of one or more functions of the system. Alarms may have two values: yellow or red; declaring the criticality of the alarm. The criticality is calculated based on the extent of violation of the conditions by the collected field data. Furthermore, there are also events which are inserted by the user under the **Event_Input_of_User** class. These events may have been caused by: the fact that there is place for improvement in the monitoring system i.e. we have not installed a sensor at a place where we should have it; the slow response of the system in an abnormal situation; an external factor i.e. in case of flood or fire in the building. Events, depending on their severity, may trigger activities such as maintenance, part replacement, etc. which are stored in the **Activity** class. To perform activities several resources are used. The available resources are in the **Resource** class. Moreover, activities may cause events (i.e. start, finish, etc.). This part of the model combining activities, events and resources is the part which supports the actual maintenance. Finally, activities are grouped at the **Process** class in order to accumulate knowledge about which activities are performed per process and make the system capable of automatically listing the activities needed depending on the events.

5.3.2 Facilitating Machine Data

The industrial partner has provided us with the bill of material (BOM) of a lathe machine, with a description for each component/part of the machine, with the list of the functions the machine can perform as well as with the information relating each part of the BOM to a specific function. Moreover, the BOM contains all the 1 770 components of the machine with a description and a separate component code for every distinct component. Several components are used in multiple places on the machine and hence, exist multiple times in the BOM.

All data from the BOM has been loaded on the **Physical_Product** class. As soon as all instances are loaded, each instance is related to an instance of the **Physical_Product** class through the properties `hasParent`/`isParentOf` depending on whether the instance consists of more than one components or not. Then, the 1 081 different components of the BOM, were loaded in the developed model as instances of the **Physical_Product_Group** class containing their 6-digit number component code which is used as the `Group_Code`. Through this code duplicates are rejected and if a component exists more than once, it is loaded only once. Therefore, the **Physical_Product_Group** class contains one instance per `Group_Code`. Furthermore, all the information describing the different functions of the machine and the different types of machines is loaded on 164 instances under the **Property** class which has been extended with 14 sub-classes. Finally, each instance of the **Physical_Product** class is related to one instance of the **Physical_Product_Group** class and to one or more instances of the **Property** class depending to how many functions the part is involved in. Appendix D contains the SWRL rules used for making the inserteed instances being related to the right instances.

At this stage a complex environment of more than 3 000 instances and 20 000 triplets has been created. All the data existing under the **Physical_Product** class, is mixed. It is not visible to which function is related each component, to which part of the actual machine each component belongs, etc. aspects which are required to improve the usability of the model. In the following paragraph we extend the model to facilitate the instances according various criteria and we make an extensive use of DL rules and the reasoner in order to check the model for inconsistencies, equivalencies, infer re-classification of the class-hierarchy and infer the instances under the new sub-classes.

5.3.3 Extending the model using DL rules to support reasoning

In this section the model described in Figure 19 is extended to facilitate the information about the lathe machine with the aim of demonstrating the capabilities provided by the use of DLs. In order to improve the created situation the **Physical_Product** class has been extended with 144 sub-classes, one for each "complex" component, which are all placed at the first level of abstraction of the class. Each one of these sub-classes is defined by a DL rule defining which instances should be under each class. This is performed by adding a DL

rule using the object property `hasParent`. For example for the **Arrosage_Canon** class we have the restriction:

$$\text{Arrosage\_Canon} \equiv \text{Physical\_Product} \cap \ni \text{hasParent.}(\textit{Arrosage\_Canon})$$

This rule means that: **Arrosage_Canon** is a **Physical_Product** whose object property `hasParent` has as value the instance of the **Physical_Product** class, named *Arrosage_Canon*. The meaning of each sub-class (in total 144 sub-classes) was defined using similar rules. The hierarchy of classes before (left part of the figure) and after the classification of classes (right part of the figure) are shown in Figure 45. The sub-classes have been re-classified under six levels of abstraction and the **Arrosage_Canon** class now is placed under the second level of abstraction. In Figure 46 the result on the instances focusing on the **Arrosage_Canon** class is shown. All instances which are parts of the *Arrosage_Canon* are categorised under the **Arrosage_Canon** class. In this figure also the sub-class **Porte_Filtre** class with its inferred instances and the object properties `hasParent` and `isParentOf` are shown. These rules allow the reasoner to understand the content of each class and according the rules to re-classify all the sub-classes at the right level of abstraction at six levels and all instances are categorised under the classes.

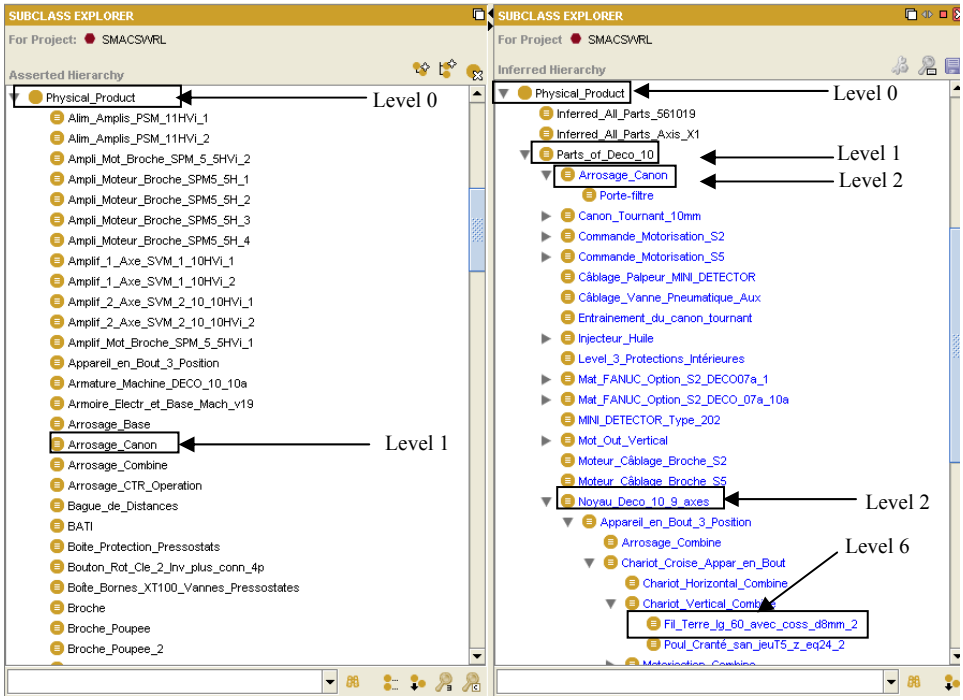

Figure 45: Classes are re-classified under six levels of abstraction.

To fulfil the requirement of tracking a specific type of component which exists in multiple products, more DL rules where added. This is useful i.e. in cases of discovering that a component is faulty due to a design error and announcing a machine recall. Thus, the manufacturer knows which machines contain this type of component and they are recalled. In order to describe this requirement in the model, new sub-classes were added to the **Physical_Product** class containing a DL rule according to the instance of the **Physical_Product_Group** class they are related to. For instance, for collecting all the machine parts which are the component with code 561019 we created the **Inferred_All_Parts_561019** class with the following restriction:

$$\text{Inferred\_All\_Parts\_561019} \equiv \text{Physical\_Product} \cap \ni \text{Physical\_Product2Physical\_Product\_Group.}(\textit{561019})$$

This rule means that: **Inferred_All_Parts_561019** is a **Physical_Product** whose object property `Physical_Product2Physical_Product_Group` has as value the instance of the **Physical_Product_Group** class, named *561019*. In Figure 47 it is shown that two instances have been categorised under this class.

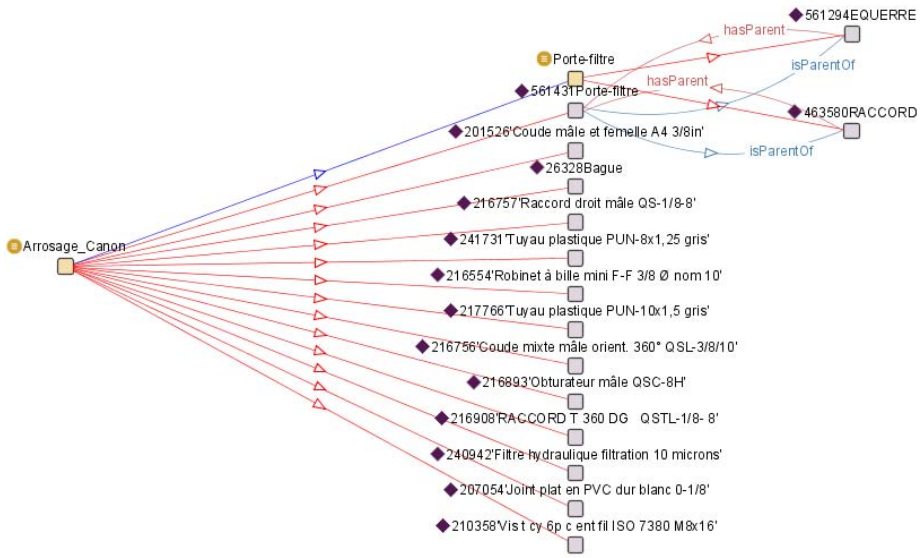

Figure 46: The instances have been inferred under the sub-classes according to the object property `hasParent`.

Another requirement was to make the system able to understand which components are involved in each basic function of the machine. In this way when an abnormality is observed on a component, the system knows which function might be affected and warns

the user. All the parts which are related to Axis X1 where collected under the **Inferred_All_Parts_Axis_X1** class with the following restriction:

$$\text{Inferred\_All\_Parts\_Axis\_X1} \equiv \text{Physical\_Product} \cap \ni \text{Physical\_Product2Property.}(\textit{M-X-01Axis\_X1})$$

The inferred twelve instances of the **Physical_Product** class which are related with Axis X1 are shown in Figure 47. It should be noted that several instances are related to multiple functions. The selected instance in Figure 47 is related to both Axis X1 and Axis X2.

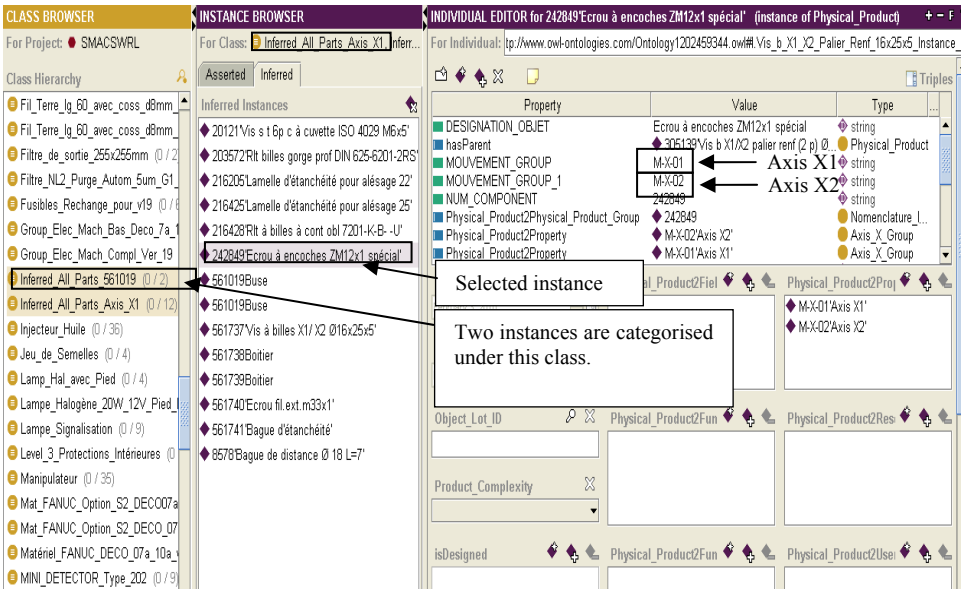

Figure 47: Inferring instances according to Function and to Physical Product Group.

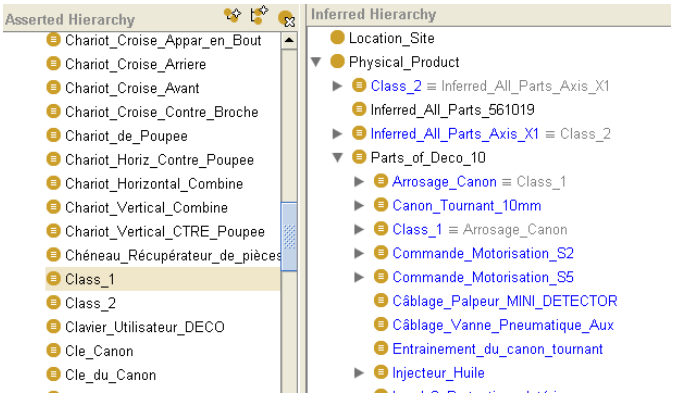

Figure 48: Equivalencies and re-classification.

As long as the model in extended with classes containing DL rules, the reasoner will categorise the new classes under the right position in the class-hierarchy and it will check if the newly created classes actually already exist in the model. An example is shown in

Figure 48 were the classes **Class_1** and **Class_2** were created having DL rules. The reasoner read the rules, checked them for their consistency, re-classified **Class_1** under the second level of abstraction and declared that the meaning of this class is the same with **Arrosage_Canon** and hence these two classes are equivalent. Regarding **Class_2**, there was no need for re-classification, but the class is declared to be the same with **Inferred_All_Parts_Axis_X1** class.

These capabilities are very important in cases of merging one or more ontology models shared among partners. Taking for granted that all partners started from the same initial SMAC-Model and they extended their models according to their needs, when merging them, the reasoner will figure out the new elements of each model. The merged model will contain only the new elements avoiding duplicates. Thus, data integration and interoperability between different model variations are preserved.

5.3.4 Time implementation

The next step was to implement the "*Duration of Time*" concept. As it has been already explained the model contains many instances with data. Therefore, the implementation of the "*Duration of Time*" concept should be performed carefully in order not to lose any data. This is secured by performing the implementation of the concept by the following steps:

1. Set the **Duration of Time** class as a super-class of the model. This class provides the unified time framework for the entire system.
2. Develop a time framework for the existing ontology PLM, and introduce it in the **Duration of Time** class. The datatype properties are: `Start_Date_Time`, `End_Date_Time`, `Duration`.
3. The already existing data of the model are copied from the datatype properties of the pre-implementation classes to the new attributes of the **Duration of Time** class. In our system architecture this is performed using SWRL rules combined with the Jess rule engine.
4. All the time related datatype properties of the pre-implementation classes are deleted from the model. They are expressed by the datatype properties of the **Duration of Time** class
5. CET was selected as central reference time for the model

The model at this stage is generic and preserves the functionalities of the previous models including: the DLs of the SMAC-Model; the connection and the continuity of information belonging to the three phases of the lifecycle beginning of life, middle of life, end of life; and the common time basis for achieving model synchronisation and for tracking down changes in any data in any part of the combination of models.

5.3.5 System Analysis

At this stage a number of sensors are monitoring temperature of certain components of the machine. The system is managing the input data from the sensors and creates events and alarms which might trigger activities. To perform activities a number of resources are required which might be available or not and might be on different systems. The model was extended to facilitate the field data coming from the sensors. The new classes of the model are shown in Figure 49. All these classes are facilitating the collected data from specific sensors which is later processed. It should be noted that the data of each sensor might be affecting more than one components of the machine and possibly one or more functions of the machine.

The sensors are constantly monitoring the temperature on the outer case of specific parts of the machine. As long as the measured value is within the pre-defined normal limits, the sensor does not transmit any data to the system. When the measured value of a sensor exceeds the limits then the sensor starts sending data to the system in real-time. Data is sent to the **Field_Data** class and from there with DL rules it is categorised under the right class. The sensor makes a measurement every 30 seconds. Therefore, each time that a measurement is inserted in the system an instance of field data is created with `Start_Date_Time`, `End_Date_Time` and `Duration`. For example, `Start_Date_Time`=2010-04-11T21:31:00, `End_Date_Time`=2010-04-11T21:31:30 and `Duration`=30. The field data is analysed against conditions in order to create events some of which are categorised as alarms.

The concept is as follows. The temperature is monitored by sensors. Experts have set two types of conditions: the first type is based on thresholds of the temperature; and the second type is based on thresholds of the temperature which are combined with the duration of the threshold violation. The normal working temperature of the current system is 60 degrees

Celsius with a tolerance of ±3 degrees. Therefore, as long as the temperature is measured below 63 degrees, the sensor is not sending any measurements to the system and the temperature is assumed as 60 degrees. When the temperature is measured higher than 63 degrees, the sensor starts sending data to the system. The sensor sends one measurement every 30 seconds. Whenever the temperature falls back, below 63 degrees the sensor stops transmitting data to the system. The same applies for all the sensors of the system. The normal working temperature is 60 degrees Celsius with a tolerance of ±3 degrees. Therefore, as long as the temperature is measured below 63 degrees the sensor is not sending any measurements to the system. It should be noted that temperature below 57 degrees exists only in the cases that the machine is off and hence, there are no conditions for lower temperatures.

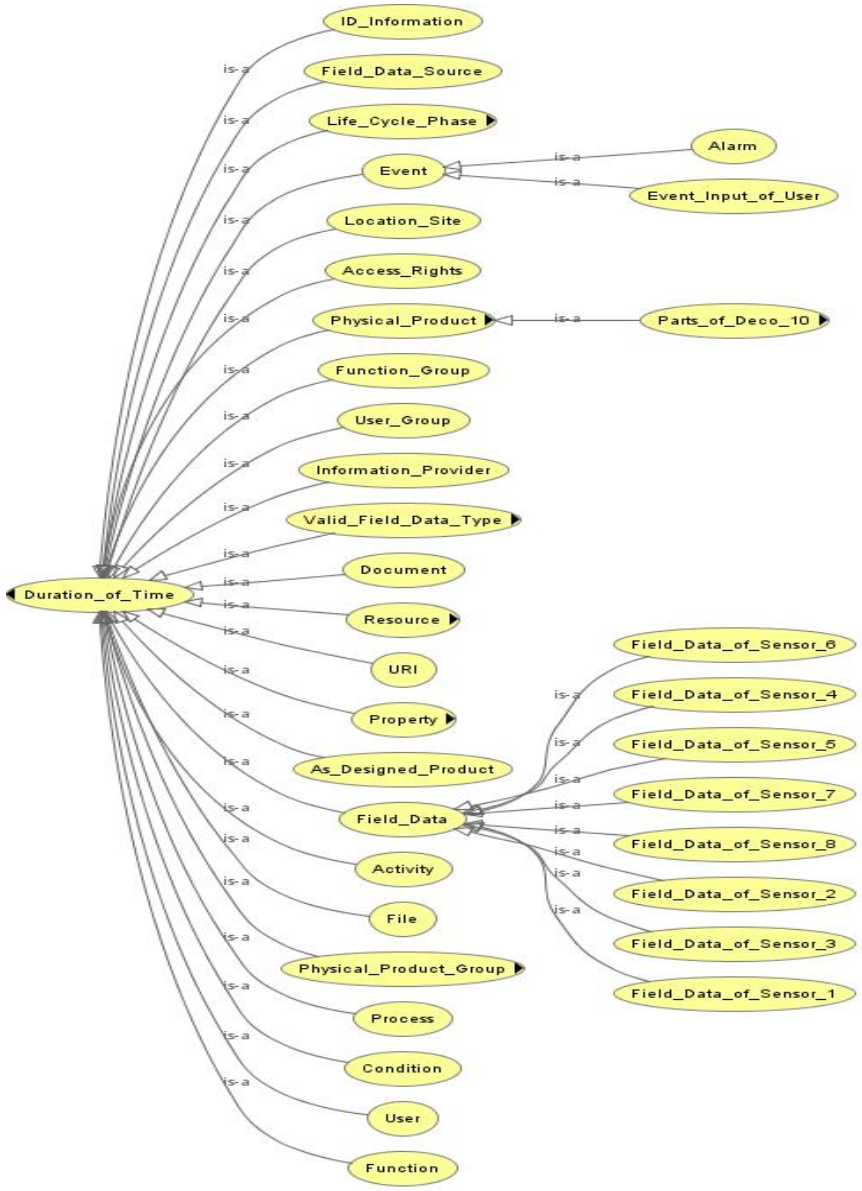

Figure 49: Ontology model extended with necessary classes

The collected data is compared with conditions and this comparison may create events and activities. Table 5 shows the list of: conditions; indicative field data which is necessary for the condition to be met; as well as the created events and alarms. When the threshold of 63 degrees is passed, the sensor starts transmitting data. This process continues as long as the temperature remains higher than 63 degrees. Then, depending on the value of the measurement alarms might be created. There are two ways this can occur: the first case is that an upper threshold has been passed; and the second case is that the measurements are in a certain region of values for more than a certain period of time.

Table 5: A list of the conditions followed creating events and alarms.

Previous Field Data	Current Field Data	Condition	Event	Alarm
60	63	T=63 degrees increasing	Threshold of 63 degrees passed	No
66	67	T=67 degrees increasing	Temperature higher than 67 degrees & increasing	Yellow
64	65	$63<T_{ave}\leq67$ for more than 2 minutes	Temperature between 63<T<67 degrees for more than 2 minutes	Yellow
69	70	T=70 degrees increasing	Temperature higher than 70 degrees & increasing	Red
68	69	$67<T_{ave}\leq70$ for more than 2 minutes	Temperature between 67<T<70 degrees for more than 2 minutes	Red
69	68	T=68 degrees decreasing	Temperature higher than 67 degrees & decreasing	Yellow
65	63	T=63 degrees decreasing	Temperature OK	No

The first condition is that the temperature was measured higher than 63 degrees. Then the sensor starts sending data and an event is recorded. When the temperature is higher than 67 degrees then a yellow alarm is created. If the temperature exceeds the value of 70 degrees then a red alarm is created. Moreover, if the average temperature (T_{ave}) is between 63 and 67 degrees for more than two minutes then this is a yellow alarm. If it is between 67 and 70 degrees for more than two minutes then this is a red alarm. It should be noted that the system is able to understand if the temperature is increasing or decreasing. This is achieved by comparing the current measurement with the previous one.

In Figure 50 it is shown how the system understands the measured values of temperature. In this figure there are three instances of the **Field_Data** class (*FD1*, *FD2* and *FD3*). The width of the instances is indicative to aid understanding. In practice the measurement has one value for temperature and not a range of values and it is considered as a single straight line along time. A measurement is received every 30 seconds. The value of the measurement corresponds to the starting time of the instance. The length of the instance illustrates that the temperature is considered stable until the next measurement arrives. In this way the temperature changes in the system only when a new measurement arrives and not before this time point.

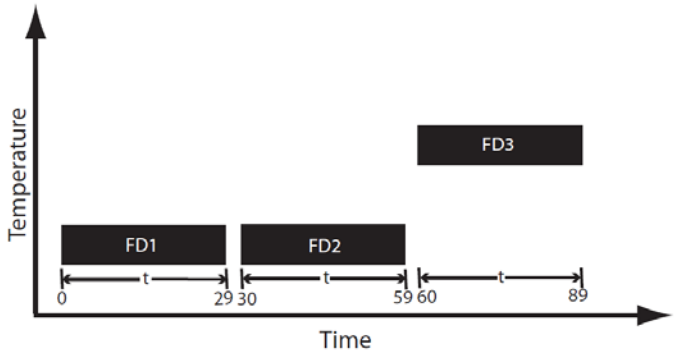

Figure 50: Field data as it is understood by the system.

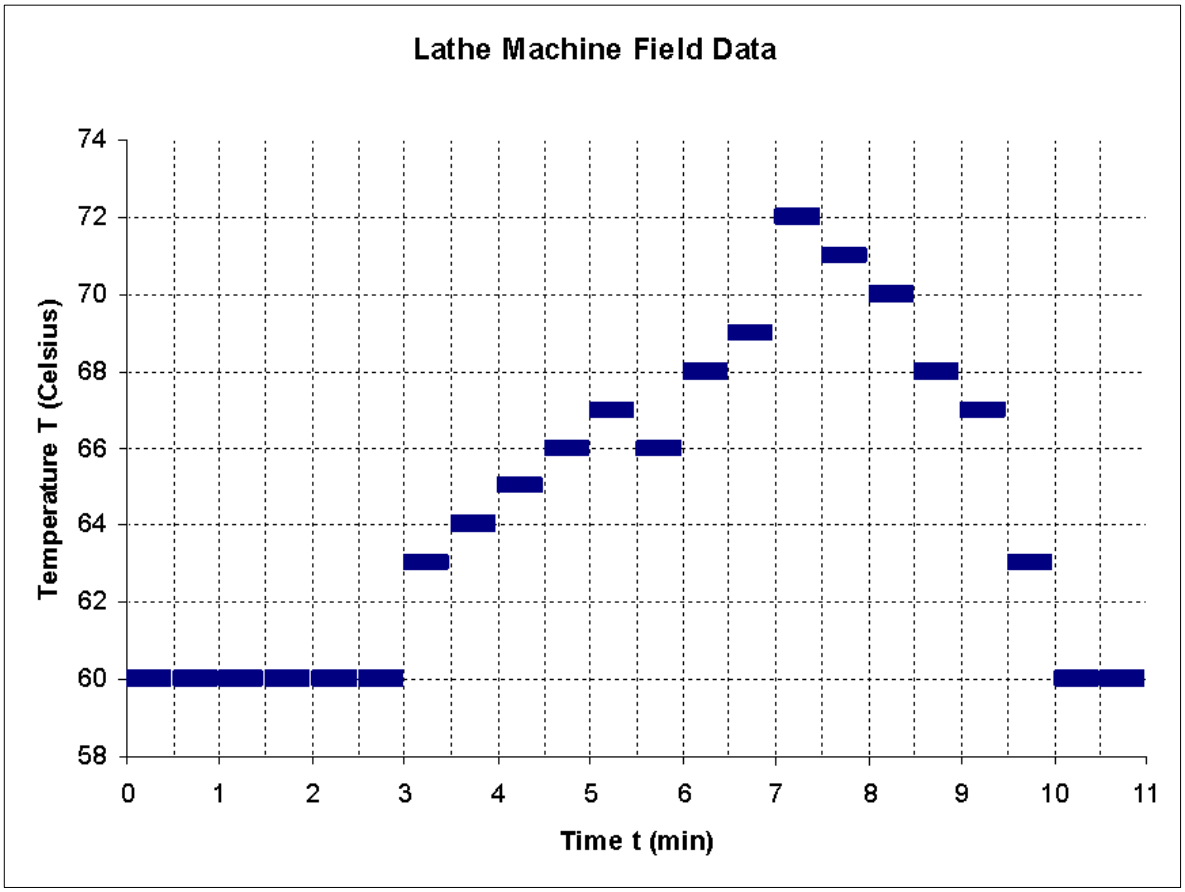

Figure 51: Field data plotted along time.

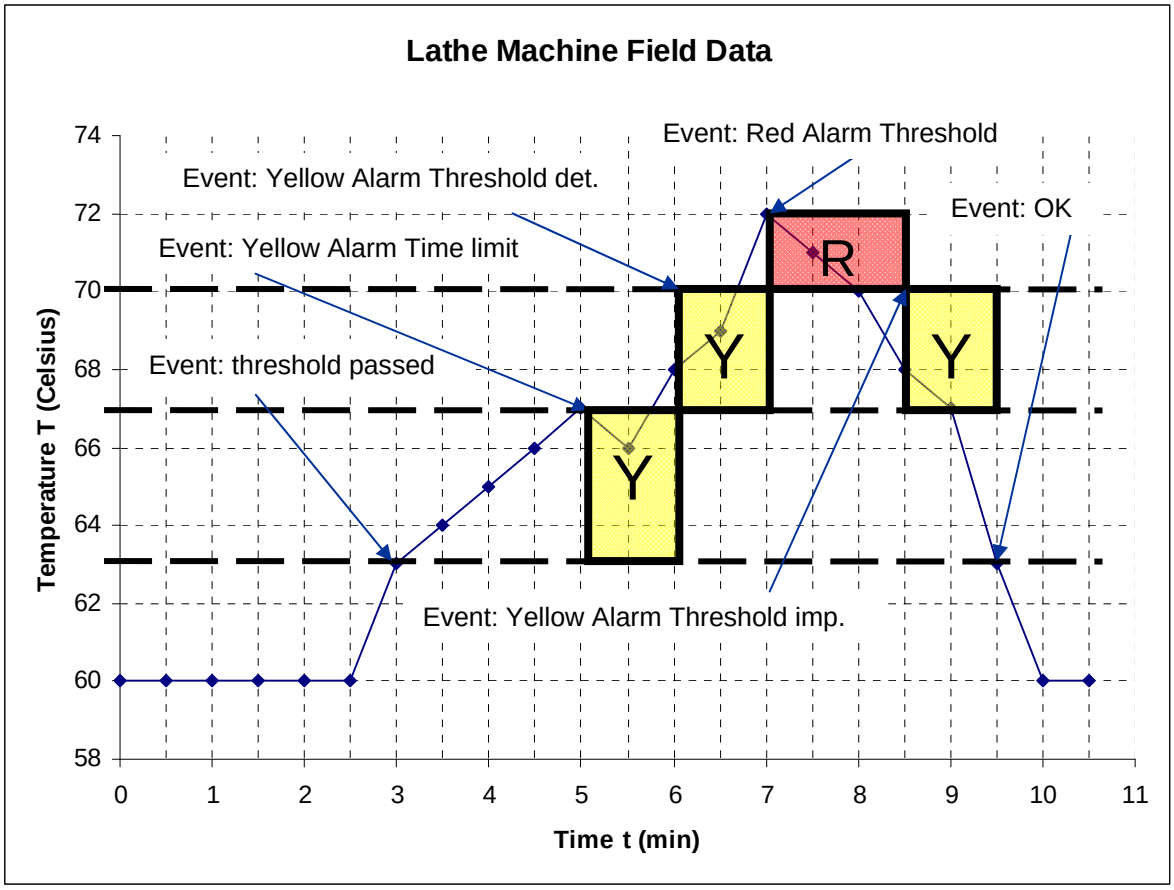

Figure 52: Field data with Event and Alarm Management example.

Measurements which cause events and alarms are plotted along time and are shown in Figure 51. It should be noted that these measurements are hypothetical in order to demonstrate the functionality of the system taking into account all the events. In the beginning until the 3rd minute the temperature is below 63 degrees and there are no measurements transmitted from the sensor. Therefore, it is assumed that the temperature is stable, and has the value of 60 degrees. The behaviour of the system when it receives this data is shown in Figure 52. In this figure, for visualisation reasons the measurements are connected together to form a graph. Three dotted horizontal lines (at 63, 67 and 70 degrees) are showing the thresholds defined by the conditions: for triggering the sensors to start sending measurements; and for generating yellow and red alarms marked with "Y" and "R" respectively. The creation of alarms is performed by using SWRL. Then, alarm and event instances are created either manually or by using the system architecture (Figure 11) with the process of exporting the result to CSV file, making it a spreadsheet and create instances

using the DataMaster tab. Another way is to insert the alarms in the model is by using the Jess rule engine, but this way is not DL safe and therefore was not selected. For more details please see the second part of Appendix D. When the time t=3min the temperature becomes 63 degrees and the system starts receiving measurements. The temperature is measured until t=5min where the temperature is 67 degrees. At that point there is a yellow alarm. The yellow alarm remains valid until t=6min 30 sec since we have that the average temperature is between 63 and 67 degrees for t=2min 30 seconds which is more than the limit of the 2 min. Then, there is a second yellow alarm from t=6min until t=7min since the temperature goes up to 68 degrees, which is higher than the yellow alarm threshold. At t=7min, there is a red alarm since the temperature is 72 degrees. This is followed by a yellow alarm starting at t=8min 30sec and lasting until t=9min 30 sec. Finally, the temperature continues to decrease until it reaches 63 degrees when an event is created meaning that temperature is ok, and the sensor stops transmitting data.

The currently developed information system might be synchronised with other systems using the "*Duration of Time*" concept for managing all the different elements which might be in different enterprises and countries. Data collected from all various systems is easily plotted along time no matter the different semantics or language used.

5.3.6 Discussion of the Case Study 3

This case study is an application of the SMAC-Model (Figure 19) in industrial environment. All the components of a lathe machine, their properties and their functions together with other BOL information have been loaded on the model. Then, the model has been extended with sub-classes using DL rules (5.3.3). Thus, the concept of each class is machine-understandable and the DL-reasoner may be applied on the model. This supports engineers to add more products on the same model; replace parts on the fly; add more levels of sub-classes; etc. Moreover, engineers do not need to know the exact structure of the model, but they simply add "instances" (following the procedure described in 5.3.3) or "classes with DLs" on the initial level of the model. Then they execute the reasoner and as it has been presented classes are checked for their consistency, for equivalencies and they are re-classified to the right position in the model; added and/or existing instances will be categorized under the right classes. These capabilities also may provide support in cases of

merging one or more ontology models (variations of the same initial model) by preserving data integration and interoperability among the models.

Furthermore, the case study includes the monitoring system and the process of data for creating events and alarms which are valuable for maintenance. The model exploits the advantage of the "*Duration of Time*" concept of having unified time attributes which are inherited in the whole model in order to provide complete data visibility and therefore, inter-OEMs/Suppliers co-operation for better resources exploitation. The use of the "*Duration of Time*" concept achieves the synchronisation of the different systems and provides the capabilities of merging the data of different sources under one common time basis. By using the "*Duration of Time*" approach engineers have an overview of all the necessary information for: the state of each resource at every moment; the state of each component of their assets; events; the activities performed; etc. The importance of this is that the system provides information and data along time about the whole system or a combination of systems. For example, one may organise activities, collect field data, manage events, manage resources and arrange spare parts in stock, under one "*Duration of Time*" basis. Moreover, the system allows to track down data and to keep information about changes and updates of the data in the group of systems. The assumption made is that all the co-operating systems are using the "*Duration of Time*" concept. A limitation of the case study is that the model has not been tested in a multi-system environment.

5.4 Conclusion

In this chapter all the developed case studies of this dissertation are presented. The first case study is an implementation of the model shown in Figure 18 and demonstrates well the functionalities of the model after using the system architecture and the related ontology based IT methods and tools. The overall performance of the model proved to provide a number of benefits including the ontology merging. However, merging requires further testing and elaboration, in order to develop methods for model architectures and extensions which support merging. In the second case study the "*Duration of Time*" concept was implemented in the model shown in Figure 18. The model proved to achieve synchronisation of the different elements of the model which might be used for data integration in multi-system and multi-level environments in a later stage. Finally, the third

case study is a combination of the benefits of DL rules with the "*Duration of Time*" concept. In this case study the SMAC-model shown in Figure 19 was extended with sub-classes and DL rules. Furthermore, the "*Duration of Time*" concept was implemented and several of the advantages were demonstrated including processing of field data and the creation of events and alarms. Under this perspective engineers have an overview of all documents, resources, data, etc. of all utilised systems, and they are supported in decision making for optimal management of resources, activities, agents and processes.

6
Model Evaluation

Our aim for evaluating the model is to show the overall functionalities, strengths and weaknesses of the developed models and concepts. Therefore, the evaluation of the model is performed towards highlighting its functionalities and utilisation.

Several techniques and methods for evaluating ontologies have been developed aiming at estimating and evaluating how well an ontology covers a certain domain. It should be noted that there is not a standardised, objective and widely accepted way of performing ontology model evaluation. Evaluation methods developed and described in section 6.1 are evaluating ontologies according to how accurate are the contained definitions and how well they describe a certain domain. In this way the methods judge the definitions contained in the ontologies like comparing the definitions contained in dictionaries. This point of view about evaluation is targeting on ontology re-use which is described in section 2.2.6. More specifically, it is aiming at supporting the selection of the most appropriate definition among a group of ontologies for each use-case.

To perform evaluation we have selected an alternative way of comparisons which are a combination of the results of the survey carried out by Brank et al. [140]. The core of the evaluation is based on the differences in coverage/functionalities/capabilities between the starting SOM model and the model developed in this work. The characteristics of the developed models are acquired by the case studies of chapter 5. Firstly, we check if the initial functionalities are preserved in the final model. Then, we check to which extent are the ontology based IT methods and tools exploited. For this reason, we study the functionalities of the final model and compare them with the theoretical functionalities of the utilised IT methods and tools. Finally, we evaluate the "*Duration of Time*" concept through the results of case studies 2 and 3 of chapter 5.

6.1 Evaluation Methods

Several studies concerning methods for ontology evaluation have been carried out. Brank et al. [140] carried out a survey on ontology evaluation approaches and sorted out four types of approaches. The first is based on comparing the ontology to a "golden standard"; The second is based on using the ontology in an application and evaluating the results; The third involves comparisons with a source of data about the domain to be covered by the ontology and finally, the one where evaluation is done by humans who try to assess how well the ontology meets a set of predefined criteria, standards, requirements, etc. They concluded that the selection of a suitable evaluation approach depends on the purpose of the evaluation, on the application in which the ontology is to be used and on what aspect of the ontology we are trying to evaluate. Obrst et al. [141] describe ontology evaluation strategies used in biomedicine, and make recommendations for future approaches. In this work are also highlighted current ontology evaluation techniques such as: evaluate an ontology in an application, compare an ontology against domain data and perform natural language evaluation. According to the authors the best evaluation of an ontology is whether it is adopted and successfully used. Finally, the authors highlight the need for developing at an interdisciplinary level: a formal and verifiable science base; tested theories that allow prediction; defined units of measure; and well-defined engineering practices. Furthermore, Guarino [142] introduced the basis for a new formal framework for evaluating and comparing ontologies by measuring their "distance" from a reference conceptualisation. In FP6 project KnowledgeWeb [143] researchers have surveyed well-known methods and tools already used to evaluate ontologies according to their usefullness and re-usability with the aim of implementing ontologies in industry. They also described glass box (component-based) and black box (task-based) evaluation; the latter usually applied to ontologies that are tightly integrated with an application, performing specific tasks. Gangemi et al. [144] developed a model for ontology evaluation in order to support the future ontology-user to define which ontology fits best to his application and requirements. Gomez-Perez [145] presents the evaluation criteria used for ontologies and describes the process of evaluation on the standard-units ontology. Obrst et al. [146] also proposed three approaches for ontology evaluation: the development of an ontology evaluation competition, the certification of ontologies, and the development of an ontology maturity model.

The main aim of ontology evaluation approaches mentioned above is to promote the re-use of existing ontologies. They evaluate the ontologies according to selected criteria (consistency, completeness, conciseness, expandability, sensitiveness, etc.) in order to provide new users with sufficient information about the content of the ontologies and about the extent of the domain coverage. Therefore, the user utilises this information to re-use an already developed ontology (as a whole or part of it) which covers his domain and fulfils his requirements.

Our approach of evaluation consists of three steps:

- First step is to compare the functionalities and concepts of the final model(s) with the ones of the initial model
- Second step is to compare the capabilities and functionalities of the developed model with those of the utilised IT methods and tools
- Third step is to evaluate the applicability, efficiency and simplicity of the "*Duration of Time*" concept

These three steps, their aims and details are described in the following paragraphs.

6.2 Evaluation Aim

The aim of our evaluation is to highlight the added value to Closed-Loop PLM models from this work. The first step is performed in order to verify whether all the functionalities and concepts of the initial model are preserved in the developed model. Thus, if this criterion is fulfilled, the domain coverage of the developed model is at least the same as the one of the initial model. The second step is performed to figure out to which extent the model implements and utilises the functionalities of the IT methods and tools. This is an indicator of how much the ontology model makes a difference in comparison to the initial SOM model in terms of re-usability and extensibility by preserving compatibility among the variations of the initial models. Moreover, this shows the level of utilisation of reasoning for fulfilling requirements, for supporting decision and for creating new knowledge. Finally, the third step is evaluating the results of implementing the "*Duration of Time*" concept in PLM models. The aim is to show the benefits, the applicability of the concept as well as its simplicity.

6.3 Evaluation Process and Results

In the first step we have to check if all the initial functionalities are preserved. The initial model is able to: describe the model as it is; to keep the information about the current and the past different usages of the product including the information of "which product is part of another product"; and to connect this information with the different lifecycle phases. The developed model (Figure 18) preserves the first (describe the model as it is) and the third (connect product's information with the different lifecycle phases) functionalities since it maintains the initial structure. The second functionality, which is to collect the information of "which product is currently or was part of another product", is covered, in the new structure, using the relationships: `isParentOf-hasParent` and the `Physical_Product2Life_Cycle_Phase` and its inverse (for details please see also section 4.2.4).

Table 6: Functionality Comparison of initial and developed model

Functionality	PROMISE SOM	OWL-DL Model	SMAC-Model
Describe Product as it is	Yes	Yes	Yes
Track history of Part Of	Yes	Yes	Yes
Manage data of BOL-MOL-EOL	Yes	Yes	Yes
Executable	No	Yes	Yes
Load Data on the Model	No	Yes	Yes
Multiple Products under one source	No	Yes	Yes
Consistency	No	Yes	Yes
Equivalencies	No	Yes	Yes
Re-classification	No	Yes	Yes
Inference	No	Yes	Yes
Import/Export Data	No	Yes	Yes
Import multiple Models under one source	No	Yes	Yes
Extended Coverage on Maintenance	No	No	Yes

Moreover, there are additional functionalities which appeared with the use of new tools and some modifications in the initial architecture. The functionalities have already been presented in the case studies in chapter 5 (case studies 1 and 3) and they are summarised in

Table 6. It should be noted that in this table the initial model is compared with both the OWL-DL Model and the SMAC-Model. From the contents of Table 6 it is obvious that the final model is: executable; data about a single or multiple products may be loaded on the model; the reasoner is used to check the model about its consistency, to check if there are equivalencies, to re-classify the class-hierarchy and to provide inference. Furthermore, the ontology editor provides the means to import and export data from and to spreadsheets and it provides the ability to import multiple models under one source. Finally, the SMAC-Model inherits the capabilities and the functionalities of the OWL-DL Model and extends the domain coverage towards maintenance with a number of new upper classes, relationships and attributes (Figure 19).

Table 7: Comparison of the developed model with the capabilities of the used IT methods and tools

Functionality	**OWL-DL Model**	**Ontology Based IT methods and tools**
Consistency	Yes	Yes
Equivalencies	Yes	Yes
Re-classification	Yes	Yes
Inference	Yes	Yes
Executable	Yes	Yes
Load Data on the Model	Yes	Yes
Multiple Products under one source	Yes	Yes
Import/Export Data	Yes	Yes
Merge Models	Yes	Yes
Simple Calculations	Yes	Yes
Merge Models	No	Yes (only in OWL 2)
Restrict All the classes	No	Yes
Automatic Data Process for Events	No	Yes

In the second step we performed the comparison of the capabilities of the utilised ontology-based IT methods and tools. The results are summarised in Table 7. The functionalities listed under the first column are a summary of the capabilities presented in chapter 2. The functionalities of the developed models are deriving from the case studies 1 and 3. The

capabilities using the DL-reasoner are used for extracting new knowledge in the model. This new knowledge was logically hidden in the model (in the forms of equivalencies, subsumptions, inference) and with the use of the reasoner it becomes tangible and can be used by the engineers. It should be noted that the upper level of the model has not been restricted with additional DL rules in order to keep this level generic. Emphasis was given on the extension of the upper level to a higher level of detail. In the evaluation we focus on the functionalities which are not fully covered. The most important not supported (by the system architecture) functionality is the ontology merging. This is a limitation of the standard of OWL 1 in which we developed our model. However, the Protégé-OWL editor allowed us to import a number of models under one source and then, apply the reasoner which is sufficient for our needs. In the case that one wishes to achieve the merging, he has to pass to OWL 2. In our case to achieve merging, we merged the models in OWL 2 and then opened the model in OWL 1 (section 5.1.5). The merged model run without any problems but the process of merging is not directly supported by the used architecture since we had to switch to an ontology editor (from protégé 3.4 to protégé 4) that supports OWL 2. It should be noted that OWL 2 became a W3C recommendation only in October of 2009 [48] and still several support tools (i.e. SWRL) are not compatible with OWL 2 editors. Another, task which has not been performed is to restrict with DL rules the upper classes of the model. This task initially was performed by the relationships and the attributes of each class. In this way, the advantage is that the classes are not heavily restricted in order to keep the generic concepts of the initial model and to keep the model flexible (i.e. "product" may be something that provides a service or something that one may buy or sale or use or a combination of the above). However, this restriction might be necessary in the cases of merging different models together as it is shown in section 5.1.5. Finally, there has not been developed an automatic data processing system which will be creating events and alarms. In the case study 3 the relevant events and alarms are created using SWRL rules but these rules should be run manually. This technical limitation may be solved in a later stage of implementation of the model.

Up to this point we have addressed research questions 1, 2 and 3. For question 1 we have provided the development process and an implementation method in chapter 4 and examples of implementation in case studies 1 and 3. Of course this might be subject to

changes in the future due to the new IT methods and tools (i.e. OWL 2) which might be developed and used. Regarding question 2 we have shown in Table 7 that following the implementation method and using the system architecture of chapter 4, almost all the available functionalities of the utilised IT methods and tools have been implemented into the developed model. In this framework one may apply the introduced implementation methods in order to develop a model with inheriting "efficiently" the characteristics of the utilised IT methods and tools. It should be noted that by the term "efficiently" by no means we claim that we have tested the functionality towards computing power and resource use. The models provided the expected results in a reasonable time *(~10 min.)* on an average current PC *(2x2.0Ghz, 2GB RAM)*. Finally, regarding question 3 the benefits and opportunities provided for the PLM models are listed in the rows of Table 7 which include advancements towards integration and interoperability shown in case studies 1 and 3 such as model merging.

In the third step we evaluated the capabilities and functionality of the developed "*Duration of Time*" concept. In case studies 2 and 3 the concept proved to be easily implemented in already developed models even if they already contain data (case study 3). The concept is not altering the structure or the notions of the model in which it is inserted, and at the same time it provides the time point of view of those concepts. This addresses the research question 4 meaning that it is possible to describe all elements through time. Moreover, the concept allows synchronising and applying time-based queries on multi-level systems. Therefore, it provides a first level of integration and interoperability since it by-passes the usage of different semantics. This addresses the research questions 5 and 6. The approach provides an overview of all the necessary information for: the state of each resource at every moment; the state of each component of their assets; events; the activities performed; etc. Thus, the system provides information and data along time about the whole system or a combination of systems. In this way the product data becomes PLM system independent since it is described through time. This addresses the research question 6 and it is the new value added by this concept. Finally research question 7 is being addressed in case study 3 in which the ontology model, the data and the "*Duration of Time*" concept are all under one source. However, the applicability of the concept remains to be tested and validated in large scale multi-level cases and will be evaluated by its adaptation and successful use.

6.4 Conclusion

The main source of evaluation are the case studies, the background works describing the capabilities of the ontology based IT methods and tools as well as the background works which are the basis of the models and methods developed in this dissertation. Firstly, the developed models were compared with the initial model on the domain coverage and the models proved to maintain the initial coverage and functionalities. Secondly, the results of the case studies were utilised as a source of the capabilities of the developed models and they were compared with the theoretical capabilities of the used IT methods and tools. There some operational imperfections of the developed models which are mainly due to technical limitations. Nevertheless, the DL rules, the reasoner as well as the rest of the system architecture elements are excessively exploited. The possible improvements would derive from the usage of newer W3C standards, and the automation of several procedures. Lastly, the "*Duration of Time*" concept, judging from the case studies, proved to be easily applicable even in already developed models with data and it does not influence the already existing semantics of the models. Its main advantage is the data integration and interoperability that it provides over the common time basis.

7
Conclusions and Future Perspectives

This dissertation deals with the implementation of ontology-based IT methods and tools in PLM models. A number of models, methods and case studies have been developed, in which IT methods and tools have been implemented. In chapter 4 are presented: the system architecture; the translation of the initial SOM model from UML into an executable OWL ontology; the extension of the model to facilitate maintenance activities; and a novel time concept the "*Duration of Time*" concept, which exploits the objectivity and the universal status of time and uses it as a reference-basis to integrate different models. Moreover, this work introduces a generic implementation method of the developed models and concepts in the system architecture. In chapter 5 three case studies are presented implementing the developed models and the system architecture of chapter 4. They demonstrate the applicability of the models, the functionalities of the models as well as the benefits they provide for Closed-Loop PLM. Finally, in chapter 6 the developed models and concepts are evaluated.

7.1 Conclusions

The conclusions of this dissertation are related to the two main contributions of this dissertation: the development of an ontology approach for PLM with the use of the relevant methods and tools; and the introduction of the original "*Duration of Time*" concept. In this dissertation it has been demonstrated that there are several benefits, functionalities and capabilities added to PLM models.

The work presented in this dissertation, provides the following generic conclusions:

- "How" to implement DL rules (and related IT methods and tools) in the existing ontology models. The main conclusion of this is that one should first understand in

depth the concepts described by the initial model or standard. Then, the aim is to describe the same concept with the available methods and tools, and to simplify the model whenever possible. It is not necessary to strictly follow the initial model structure, since it is possible that this will prevent the model from acquiring new functionalities of the utilised methods and tools.

- "Why" to do this implementation? This is performed in order to introduce the functionalities of the IT methods and tools in the PLM models and create or discover new knowledge. Such implementations have lead to providing benefits towards: consistency checking, checking for equivalent concepts, re-classification of the class hierarchy and inference of the instances under the right place of the model.
- "What" are the benefits for the PLM models? Models are able to store data about multiple products; models have become extensible with characteristics of merging and auto-mapping.
- What is the "new knowledge" generated? Well, with the use of equivalencies and re-classification it is made possible to find out which classes describe concepts which are equivalent to other classes, to position the classes at the right level in the class-hierarchy and to infer instances under the classes. These functionalities are provided automatically without the requirement of the user to know the detailed structure of the model. Therefore, the user has an overview of the concepts that already exist in the model and the system identifies which instances (data) are useful also in other parts of the model.

The implementation of ontologies and the use of DLs have provided the following functionalities shown in detail in Case Studies 1 and 3:

- The DL-reasoner understands the DL rules and checks the extended ontology model for its consistency; identifies equivalencies among the classes; re-classifies classes to their logical position in the model; and infers instances at their logical position in the class-hierarchy.
- The model is extensible and the extended models are compliant to each other with the proper use of DL rules.

- The model facilitates and handles multiple data from multiple physical products.

As it has been shown in Case Study 1 in section 5.1.5, these functionalities aid ontology merging. They could be used to achieve auto-mapping of the variations of the models and to efficiently find out important beneficial elements in one model and then, use them across the models. Regarding the merging, the conclusion is that more research still should be performed in this field in order to provide methodologies for ontology and rule development in order to support automated ontology merging [56]. The research could also be towards providing model-developing rules for safe extension and consistency after merging.

The second part of the conclusions derives from the development of the "*Duration of Time*" concept. The implementation of the "*Duration of Time*" concept is aiming at preserving the continuity of information along time as well as at providing the basis for synchronising different information systems. These advantages are valid no matter the different products tracked, the different information systems used or the semantics used. Therefore, once all different systems are based on the "*Duration of Time*" concept, all information contained on them may be plotted along time. In this way, data from different information systems are easily synchronised and organised together.

The functionalities of models implementing the "*Duration of Time*" concept are presented in detail in Case Studies 2 and 3 and they are summarised as:

- The concept is easily implemented in PLM models using current technology while the models maintain their initial functionalities.
- The system provides complete data visibility which functions also under multi-system circumstances.
- Systems based on the concept may be easily synchronised.
- Partners achieve better resources exploitation through synchronisation.

It should be noted that the concept still, remains to be tested in a real commercial multi-system environment.

Moreover, the results of the three case studies are used as the source of performing the evaluation of this work. The evaluation process has demonstrated that the model has improved in several aspects (chapter 6). Firstly, we performed the comparison of the functionalities and concepts of the final model(s) with the ones of the initial model and we concluded that the initial functionalities are preserved and that the final models contain new functionalities as well as concepts. Secondly, we compared the capabilities and the functionalities of the developed models with those of the utilised IT methods and tools, and we concluded that the IT methods and tools are exploited in a great extend. However, technological advances are actively changing and therefore, new opportunities might appear by implementing the most recent methods and tools. Finally, we evaluated the applicability, efficiency and simplicity of the "*Duration of Time*" concept, which proved to be promising for first level integration of information systems.

7.2 Future Perspectives

This work could be extended in several aspects related to the technology used, the developed concepts and models.

- The use of newer ontology-based IT methods and tools such as OWL 2 which can provide extra functionalities to the existing models according to our initial investigations discussed in section 6.3.
- Future research may enrich the models through an exhaustive mapping with well known standards such as MIMOSA and ISO-15926. This will provide added value to the models and possibly be a basis for their wider adoption and successful use by the PLM community.
- Future research may also attempt to provide a widely accepted standard of terms and meanings (similar i.e. to biology). This will create a reference-basis for mapping existing and future models and standards.
- Engineers could collaborate with computer scientists and define methodologies for developing and extending models towards allowing ontology merging. Models developed using such methodologies would guarantee seamless ontology merging.

- The provided method for developing models implementing new technologies can be utilised for creating ontology models for various applications such as manufacturing processes, resources, generic components, product families per industrial sector, identification of best practices, etc.
- Regarding the "*Duration of Time*" concept future applications would include multi-system and multi-level environments as well as possible implementation of the concept in a commercial PLM platform.

References

[1] Sääksvuori A. and Immonen A. *Product Lifecycle Management*. 3rd Edition; Springer-Verlag; 2008

[2] Stark J. *Product lifecycle management: 21st century paradigm for product realisation*. Springer-Verlag; 2005

[3] Kiritsis D., Bufardi A. and Xirouchakis P. "Research issues on product lifecycle management and information tracking using smart embedded systems". *Advanced Engineering Informatics 2003;* 17 (3-4), pp. 189-202

[4] Noy N.F. and McGuinness D.L. *Ontology development 101: A guide to creating your first ontology*. Stanford University; 2001.

[5] McGuinness D.L., Fikes R., Rice J. and Wilder S. "An Environment for Merging and Testing Large Ontologies". *In Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000;* Breckenridge, Colorado, USA 12-15 April 2000.

[6] Grossmann G. *"Horizontal and Vertical Integration of Object Oriented Information Systems Behaviour"*. PhD thesis, University of South Australia. 2008, URL: http://arrow.unisa.edu.au:8080/vital/access/manager/Repository/unisa:35964

[7] PROMISE deliverable DR5.4: Generic PEID roadmap for each group. http://www.promise.no/downloadfile.php?i=b7b16ecf8ca537235938941160717 00c last accessed April 2010.

[8] International Society of Engineering Asset Management: ISEAM www.iseam.org last accessed April 2010.

[9] Berners-Lee T. "Semantic Web on XML". *XML 2000;* Washington DC, USA, 3-8 December 2000, http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

[10] Signore O. "Representing Knowledge in the Semantic Web". *Open Culture Conference - accessing and sharing Knowledge (organised by the Italian office of W3C);* June 27-29, 2005 Milan, Italy. http://www.w3c.it/papers/openCulture2005.pdf

[11] "SWRL: A Semantic Web Rule Language Combining OWL and RuleML". http://www.w3.org/Submission/SWRL/. Last accessed January 2010.

[12] Prud'hommeaux E., Seaborne A. "SPARQL Query Language for RDF". *W3C Recommendation 15 January 2008.* http://www.w3.org/TR/rdf-sparql-query/ Last accessed January 2010.

[13] Bray T., Paoli J., Sperberg-McQueen C.M., Maler E. and Yergeau F. "Extensible Markup Language (XML) 1.0 (Fifth Edition)". *W3C Recommendation 26 November 2008.* http://www.w3.org/TR/2008/REC-xml-20081126/, Last accessed January 2010.

[14] Daconta M.C., Obrst L.J. and Smith K.T. *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*. Wiley; 2003.

[15] Zeid I. and Gupta S.M. "Disassembly knowledge representation via XML". *Proceedings of International Conference SPIE Environmentally Conscious Manufacturing;* 2000, vol. 4193, pp. 179-185

[16] Fallside D.C. and Walmsley P. "XML Schema Part 0: Primer Second Edition". *W3C Recommendation 28 October 2004.* http://www.w3.org/TR/xmlschema-0/ Last accessed January 2010.

[17] Miller E. "An introduction to the resource description framework". *Bulletin of the American Society for Information Science;* 1998, 25 (1) pp. 15-19

[18] Decker S., Melnik S., Van Harmelen F., Fensel D., Klein M., Broekstra J., Erdmann M. and Horrocks I. "The Semantic Web: The Roles of XML and RDF". *IEEE Internet Computing;* 2000, 4 (5), pp. 63-74

[19] Klein M. "XML, RDF, and relatives". *IEEE Intelligent Systems;* 2001, 16 (2), pp. 26-28

[20] Klyne G. and Carroll J.J. "Resource description framework (RDF): Concepts and abstract syntax". *W3C Recommendation 10 February 2004.* http://www.w3.org/TR/rdf-concepts, Last accessed January 2010.

[21] Manola F. and Miller E. "RDF Primer". *W3C Recommendation 10 February 2004.* http://www.w3.org/TR/rdf-primer, Last accessed January 2010.

[22] Klyne G. "Information modeling using RDF-Constructs for Modular Description of Complex Systems" http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.6732&rep=rep1&type=pdf, Last accessed May 2010.

[23] Brickley D. and Guha R. "RDF Vocabulary Description Language 1.0: RDF Schema". *W3C Recommendation 10 February 2004.* http://www.w3.org/TR/2004/REC-rdf-schema-20040210/, Last accessed January 2010.

[24] McGuinness D.L. and Harmelen F. "OWL Web Ontology Language Overview". *W3C Recommendation 10 February 2004.* http://www.w3.org/TR/2004/REC-owl-features-20040210/, Last accessed January 2010.

[25] Bruijn J., Polleres A. and Fensel D. "Web Service Modelling Language (WSML)". *Deliverable D20*. 18 July 2004. http://www.wsmo.org/2004/d20/v0.1/20040629/d20v0.1_20040629.pdf, Last accessed January 2010.

[26] Baader F., Calvanese D., McGuinness D.L., Nardi D. and Patel-Schneider P.F., editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press; 2003

[27] Horrocks I. and Sattler U. "Description Logics: Basics, Applications, and More". *The International Joint Conference on Automated Reasoning;* June 18-23, 2001, Siena, Italy

[28] Cardoso J. "The Semantic Web Vision: Where Are We?". IEEE Intelligent Systems; 22 (5), pp. 84-88

[29] Jena-A Semantic Web Framework for Java: http://jena.sourceforge.net/ Last accessed January 2010.

[30] Racer: http://www.racer-systems.com/ Last accessed January 2010.

[31] Pellet Reasoner: http://clarkparsia.com/pellet/ Last accessed January 2010.

[32] Fact++: http://owl.man.ac.uk/factplusplus/ Last accessed January 2010.

[33] The OWL reasoner of Jena 2: http://jena.sourceforge.net/inference/#owl Last accessed January 2010.

[34] Sirin E., Parsia B. and Grau B.C., Kalyanpur A and Katz Y. "Pellet: A practical OWL-DL reasoner". *Web Semantics;* 2007, 5 (2), pp. 51-53

[35] Sirin E. and Parsia B. "Pellet: An OWL-DL reasoner". *Proceedings of the 2004 International Workshop on Description Logics (DL2004);* Whistler, BC, Canada June 6-8, 2004, pp. 212-213

[36] Drummond N. and Shearer R. "The Open World Assumption". University of Manchester. Last updated 11 October 2006. http://www.cs.man.ac.uk/~drummond/presentations/OWA.pdf, Last accessed January 2010.

[37] RuleML: http://www.ruleml.org/

[38] O'Connor M.J., Knublauch H., Tu S.W., Grossof B., Dean M., Grosso W.E. and Musen M.A. "Supporting Rule System Interoperability on the Semantic Web with SWRL". *Fourth International Semantic Web Conference (ISWC-2005);* Galway, Ireland, 2005; Springer, LNCS 3729 pp. 974-986

[39] Grosof B.N., Horrocks I., Volz R. and Decker S. "Description logic programs: combining logic programs with description logic". *Proceedings of the 12th international conference on World Wide Web;* May 20-24 2003, Budapest, Hungary. pp. 48-57

[40] Horrocks I., Parsia B., Patel-Schneider P.F. and Hendler J. "Semantic Web architecture: Stack or two towers?". *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics;* 2005, 3703 LNCS, pp. 37-41

[41] The Protégé Ontology Editor and Knowledge Acquisition System. http://protege.stanford.edu/

[42] SWRL protégé plug-in: http://protegewiki.stanford.edu/index.php/SWRLTab. Last accessed January 2010.

[43] The Jess rule engine: http://herzberg.ca.sandia.gov/. Last accessed January 2010.

[44] SWRL Jess Tab: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLJessTab. Last accessed January 2010.

[45] SQWRL: The Semantic Query-Enhanced Web Rule Language. http://protege.cim3.net/cgi-bin/wiki.pl?SQWRL. Last accessed January 2010.

[46] Collections SQWRL: http://protege.cim3.net/cgi-bin/wiki.pl?CollectionsSQWRL, Last accessed May 2010.

[47] Grau B.C., Horrocks I., Motik B., Parsia B., Patel-Schneider P.F. and Sattler U. "OWL 2: The next step for OWL". *Web Semantics;* 2008, 6 (4), pp. 309-322

[48] Bao J., Calvanese D., Grau B.C., Dzbor M. et al. (OWL Working Group). "OWL 2 Web Ontology Language". *W3C Recommendation 27 October 2009;* http://www.w3.org/TR/owl2-overview/#ack, Last accessed January 2010.

[49] Stumme G. and Maedche A. "FCA-merge: bottom-up merging of Ontologies". *Proceedings of the 17th International Conference on Artificial Intelligence (IJCAI'01);* Seattle, WA, USA, 2001, pp. 225-230.

[50] Noy N.F. and Musen M.A. "The PROMPT suite: Interactive tools for ontology merging and mapping". *International Journal of Human Computer Studies;* 2003, 59 (6), pp. 983-1024.

[51] Kotis K., Vouros G.A. and Stergiou K. "Towards automatic merging of domain ontologies: The HCONE-merge approach". *Web Semantic;* 2006, 4 (1), pp. 60-79.

[52] Grau B.C., Horrocks I., Kazakov Y. and Sattler U. "Just the right amount: Extracting modules from Ontologies". *16th International World Wide Web Conference, WWW2007;* May 8-12, 2007, Banff, Alberta, Canada, pp. 717-726

[53] Jiménez-Ruiz E., Grau B.C., Sattler U., Schneider T. and Berlanga R. "Safe and economic re-use of ontologies: A logic-based methodology and tool support". *5th European Semantic Web Conference, ESWC 2008;* June 1-5, 2008, Tenerife, Canary Islands, Spain, pp. 185-199

[54] Grau B.C., Horrocks I., Kazakov Y. and Sattler U. "Ontology reuse: Better safe than sorry". *20th International Workshop on Description Logics (DL-2007);* 8-10 June 2007 Brixen-Bressanone, Italy

[55] Grau B.C., Horrocks I., Kazakov Y. and Sattler U. "Modular reuse of ontologies: Theory and practice". *Journal of Artificial Intelligence Research;* 2008, 31 pp. 273-318

[56] Corcho O., Fernández-López M. and Gómez-Pérez A. "Ontological Engineering: What are Ontologies and How Can We Build Them?", in Cardoso J editor. *Semantic Web Services: Theory, Tools and Applications*; IGI Global (former Idea Group). Hersey, Pennsylvania, USA, 2007, pp. 44-70

[57] Denny M. "Ontology Tools Survey, Revisited". July 14, 2004. http://www.xml.com/pub/a/2004/07/14/onto.html. The complete list is found here: http://www.xml.com/2004/07/14/examples/Ontology_Editor_Survey_2004_Table_-_Michael_Denny.pdf Last accessed January 2010.

[58] SWOOP editor: http://code.google.com/p/swoop/ Last accessed January 2010.

[59] OntoStudio: http://www.ontoprise.de/en/home/products/ontostudio/ Last accessed January 2010.

[60] Escórcio L. and Cardoso J. "Editing Tools for Ontology Creation", in Cardoso J editor. *Semantic Web Services: Theory, Tools and Applications*; IGI Global (former Idea Group). Hersey, Pennsylvania, USA, 2007, pp. 71-95

[61] Kalyanpur A., Parsia B., Sirin E., Grau B.C. and Hendler J. "Swoop: A Web Ontology Editing Browser". *Web Semantics;* 2006, 4 (2), pp. 144-153

[62] Kifer M., Lausen G. and Wu J. "Logical foundations of object-oriented and frame-based languages". *Journal of the ACM ;* 1995, 42 (4), pp. 741-843

[63] Eclipse software editor. http://www.eclipse.org/ Last accessed April 2010.

[64] Golbreich C. and Wallace E.K. "OWL 2 Web Ontology Language New Features and Rationale". *W3C Recommendation 27 October 2009*. http://www.w3.org/TR/owl2-new-features/, Last accessed January 2010.

[65] Motik B., Grau B.C., Horrocks I., Wu Z. and Fokoue A., Lutz C. "OWL 2 Web Ontology Language Profiles". *W3C Recommendation 27 October 2009*. http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/, Last accessed January 2010.

[66] Rumbaugh J. *The UML Reference Manual*. Addison-Wesley; 1999.

[67] France R., Evans A., Lano K. and Rumpe B. "The UML as a formal modeling notation". *Computer Standards and Interfaces;* 1998, 19 (7), pp. 325-334

[68] Evans A.S. "Reasoning with UML Class Diagrams". *Second IEEE Workshop on Industrial Strength Formal Specification Techniques*, Oct 20-23, 1998, Boca Raton, FL, USA

[69] Berardi D., Calvanese D. and De Giacomo G. "Reasoning on UML class diagrams". *Artificial Intelligence*; 2005, 168 (1-2), pp. 70-118

[70] MIMOSA (Machinery Information Management Open Systems Alliance) www.mimosa.org last accessed March 2010.

[71] OSA-EAI Technical Architecture Summary V3.2 [release Dec 2007].

[72] Mathew A., Zhang L., Zhang S and Ma L. "A Review of the MIMOSA OSA-EAI Database for Condition Monitoring Systems". *Proceedings of the 1st World Congress on Engineering Asset Management (WCEAM)*; 11 – 14 July 2006, Gold Coast, Australia, pp. 837-846

[73] OSA-CBM Primer V3.2 [Aug 2006].

[74] Keller K., Wiegand D., Swearingen K., Reisig C., Black S., Gillis A. and Vandernoot M. "An Architecture to Implement Integrated Vehicle Health Management Systems". *IEEE Systems*

Readiness Technology Conference (AUTOTESTCON); Valley Forge, PA, USA, 20-23 Aug 2001, pp. 2-15.

[75] Byington C.S., Kalgren P.W., Dunkin B.K. and Donovan B.P. "Advanced diagnostic/prognostic reasoning and evidence transformation techniques for improved avionics maintenance". *Proceedings of the IEEE Aerospace Conference*; 13 March, 2004, Big Sky, Montana, USA. (5) pp. 3424-3434.

[76] Chidambaram B., Gilbertson D.D. and Keller K. "Condition-based monitoring of an electro-hydraulic system using open software architectures". *Proceedings of the IEEE Aerospace Conference*; 5-12 March, 2005, Big Sky, Montana, USA. pp. 3532 - 3539.

[77] Lebold M., Reichard K. and Boylan D. "Utilizing DCOM in an open system architecture framework for machinery monitoring and diagnostics". *Proceedings of the IEEE Aerospace Conference*; 8-15 March, 2003, Big Sky, Montana, USA. (3) pp. 1227-1236.

[78] Voisin A., Levrat E., Cocheteux P. and Iung B. "Generic prognosis model for proactive maintenance decision support- application to pre-industrial e-maintenance test bed". *Journal of Intelligent Manufacturing*; 2010, 21 (2), pp. 177-193.

[79] Mun D., Lee S., Kim B. and Han S. "ISO 15926-based data repository and related web services for sharing lifecycle data of process plants". *Proceeding of the international conference on Product Lifecycle Management-PLM 09 2009*; 6-8 July 2009, Bath, UK.

[80] Teijgeler H. "The process industries and the ISO 15926 Semantic web". (2007) OntoConsult: http://www.infowebml.ws/Topics/papers/15926SW.htm (last accessed June 2009)

[81] Batres R., Shimada Y. and Fuchino T. "A graphical approach for representing hazard scenarios". *American Institute of Chemical Engineers – Global Congress on Process Safety; Topical 1 (1007), AIChe*; 2008, New Orleans, USA, pp. 386-391

[82] Sandsmark N. and Mehta S. "Integrated information platform for reservoir and subsea production systems". *Proceedings of the 13th Product Data Technology Europe Symposium (PDT 2004)*; 18-20 October 2004, Stockholm, Sweden.

[83] Gulla J.A., Tomassen S.L. and Strasunskas D. "Semantic interoperability in the Norwegian petroleum industry". *5th International Conference on Information Systems ICIS 2006*; 10-13 Dec 2006, Milwaukee, WI, USA

[84] Tomassen S.L., Gulla J.A. and Strasunskas D. "Document space adapted ontology: Application in query enrichment". *Proceedings of the 11th International Conference on Applications of Natural Language to Information Systems, NLDB 2006;* May 31 - June 2, 2006. Klagenfurt, Austria. pp. 46-57

[85] Strasunskas D. "Resource Monitoring and Rule-Based Notification. Applications in Subsea Production Systems". In Khosrow-Pour, M. (Ed.) Managing Worldwide Operations and Communications with Information Technology (*Proc. of the 2007 IRMA International Conference;* 19-23 May 2007, Vancouver, Canada); IDEA Group Publishing, pp. 1211-1213.

[86] Klüwer J.W., Skjæveland M.G. and Valen-Sendstad M. "ISO 15926 templates and the Semantic Web". *W3C Workshop on Semantic Web in Energy Industries, Part I: Oil & Gas*; 9-10 Dec 2008, Houston, TX, USA

[87] Price D. and Bodington R. "Applying semantic web technology to the life cycle support of complex engineering assets". *Proceedings in the Third International Semantic Web Conference-ISWC 2004;* 7-11 November, 2004, Hiroshima, Japan. pp. 812-822

[88] OASIS 'Product Life Cycle Support Technical Committee', Available at: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=plcs last accessed April 2010.

[89] Stell J.G. and West M. "A 4-dimensionalist mereotopology". *Proceedings of The third Formal Ontology in Information Systems conference FOIS 2004;* 4-6 November, 2004, Torino, Italy. pp. 261-272

[90] PROMISE FP6 project: www.promise.no, December 2008.

[91] PROMISE Research Deliverable 9.2: http://www.promise.no/downloadfile.php?i=69adc1e107f7f7d035d7baf04342e1ca, :ast accessed December 2008.

[92] Cassina J., Tomasella M., Taisch M. and Matta A. "A new closed-loop PLM Standard for mass products". *Journal of Product Development*, 8 (2) (2009), pp. 141-161

[93] Spackman K.A., Campbell K.E. and Côté R.A. "SNOMED RT: A Reference Terminology for Health Care". *Journal of the American Medical Informatics Association 1997*; 4 (SUPPL.), pp. 640-644

[94] Humphreys B.L. and Lindberg D.A.B. "The UMLS project: Making the conceptual connection between users and the information they need". *Bulletin of the Medical Library Association 1993*; 81 (2), pp. 170-177

[95] SNOMED CT: http://www.ihtsdo.org/snomed-ct/, Last accessed March 2010.

[96] Bodenreider O., Smith B., Kumar A. and Burgun A. "Investigating subsumption in SNOMED CT: An exploration into large description logic-based biomedical terminologies". *Artificial Intelligence in Medicine 2007*; 39 (3), pp. 183-195

[97] Horrocks I., Li L., Turi D. and Bechhofer S. "The Instance Store: DL Reasoning with Large Numbers of Individuals". *Proc. of the 2004 Description Logic Workshop (DL 2004) 2004*; pp. 31-40

[98] Brandt S. "Reasoning in ELH w.r.t. General Concept Inclusion Axioms". *Technical Report LTCS-Report 04-03*, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2004. http://lat.inf.tu-dresden.de/research/reports/2004/Brandt-LTCS-04-03.pdf

[99] Golbeck J., Fragoso G., Hartel F., Hendler J., Oberthaler J. and Parsia B. "The National Cancer Institute's thésaurus and ontology". *Web Semantics*; 2003; 1 (1), pp. 75-80

[100] The Gene Ontology (GO) http://www.geneontology.org/, Last accessed March 2010.

[101] Rogers J. and Rector A. "The GALEN ontology". *Medical Informatics Europe (MIE 96)*; 1996, Copenhagen, Denmark. IOS Press; pp. 174-178

[102] Niles I. and Pease A. "Towards a Standard Upper Ontology". *In Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, Chris Welty and Barry Smith, eds; October 17-19, 2001; Ogunquit, Maine, USA. pp. 2-9

[103] DOLCE ontology. http://www.loa-cnr.it/DOLCE.html, Last accessed March 2010.

[104] Masolo C., Borgo S., Gangemi A., Guarino N. and Oltramari A. "Ontology Library". *WonderWeb Deliverable D18* (of IST Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web) http://wonderweb.man.ac.uk/deliverables/documents/D18.pdf, Last accessed March 2010.

[105] Batres R., West M., Leal D., Price D., Masaki K., Shimada Y., Fuchino T. and Naka Y. "An upper ontology based on ISO 15926". *Computers and Chemical Engineering*, 2007, 31 (5-6), pp. 519-534

[106] Smith B. "Against idiosyncrasy in ontology development". *Proceedings of the Formal Ontology in Information Systems (FOIS 2006)*, 2006, pp. 15-26

[107] Leal D. "ISO 15926 "Life cycle data for process plant": An overview". *Oil and Gas Science and Technology*, (2005), 60 (4), pp. 629-637

[108] Hakkarainen S., Hella L., Strasunskas D. and Tuxen S. "A semantic transformation approach for ISO 15926". *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 4231 LNCS;* 2006, pp. 281-290

[109] Fiorentini X., Gambino I., Liang V.C., Foufou S., Rachuri S., Bock C. and Mani M. "Towards an Ontology for Open Assembly model". *Proceeding of the international conference on Product Lifecycle Management-PLM 07 2007*; 11-13 July, 2007, Bergamo, Italy; pp 445-456

[110] Fiorentini X., Rachuri S., Mahesh M., Fenves S. and Sriram Ram D. "Description Logic for Product Information Models". *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference 2008;* DETC2008-49348

[111] Tektonidis D., Bokma A., Oatley G. and Salampasis M. "ONAR: An ontologies-based service oriented application integration framework". *Proceedings of First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA) 2005*; pp. 68-77, Geneva, Switzerland.

[112] Lee J.H. and Suh H.W. "Owl-based product ontology architecture and representation for sharing product knowledge on a web". *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference 2007;* DETC2007; 2 PART B, pp. 853-861

[113] Brandt S.C., Morbach J., Miatidis M., Theißen M., Jarke M. and Marquardt W. "An ontology-based approach to knowledge management in design processes". *Computers and Chemical Engineering 2007*; 32 (1-2), pp. 320-342

[114] Zhang W.Y. and Yin J.W. "Exploring Semantic Web technologies for ontology-based modeling in collaborative engineering design". *International Journal of Advanced Manufacturing Technology 2007*; 36 (9-10), pp. 833-843

[115] Suh S.H., Shin S.J., Yoon J.S. and Um J.M. "UbiDM: A new paradigm for product design and manufacturing via ubiquitous computing technology". *International Journal of Computer Integrated Manufacturing*; 2008, 21 (5), pp. 540-549

[116] Chang X. and Terpenny J. "Ontology-based data integration and decision making for product e-design". *Robotics and Computer Integrated Manufacturing*; 2009, 25 (6), pp.863-870

[117] Jun H.B., Kiritsis D., and Xirouchakis P. "A primitive ontology model for product lifecycle meta data in the closed-loop PLM". In: Gonçalves RJ., Müller JP., Mertins K., and Zelm M., editors. *Enterprise Interoperability II: New Challenges and Approaches*, Springer verlag London Limited; 2007, pp. 729-740

[118] Aziz H., Gao J., Maropoulos P. and Cheung W.M. "Open standard, open source and peer-to-peer tools and methods for collaborative product development". *Computers in Industry;* 2005, 56 (3), pp. 260-271

[119] Sider T. *Four-dimensionalism: An Ontology of Persistence and Time.* Oxford University Press; 2001

[120] ISO 15926-2:2003 Integration of lifecycle data for process plant including oil and gas production facilities: Part 2 – Data model: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=29557, Last accessed March 2010.

[121] West M. "Some industrial experiences in the development and use of Ontologies". *EKAW 2004 Workshop on Core Ontologies in Ontology Engineering*, 8 Oct. 2004, Whittlebury Hall, Northamptonshire, UK. pp. 1-14

[122] Roddick J.F., Egenhofer M.J., Hoel E., Papadias D. and Salzberg B. "Spatial, temporal and spatio-temporal databases - Hot issues and directions for PhD research". *SIGMOD Record 2004;* 33 (2), pp. 126-131

[123] Zhang C. and Hammad A. "Spatio-temporal issues in infrastructure lifecycle management systems". *Proceedings, 1st Annual Conference - Canadian Society for Civil Engineering;* 2-4 June, 2005, Toronto, Canada. pp. FR-131-1 to FR-131-10

[124] Roddick J.F., Hornsby K. and Spiliopoulou M. "An Updated Bibliography of Temporal, Spatial, and Spatio-temporal Data Mining Research". *Temporal, spatial, and spatio-temporal data mining: first international workshop, TSDM 2000*; 12 Sep, Lyon, France, pp. 147–163. Heidelberg/Berlin: Springer Verlag.

[125] Tanenbaum A.S. and Steen M.V. *Distributed Systems: Principles and Paradigms*. Upper Saddle River, NJ : Pearson/Prentice Hall; 2007

[126] Horrocks I. "Semantic Web: The story so far". ACM International Conference Proceedings vol. 225, pp. 120-125, *Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*, Banff, Canada; 2007.

[127] Wallace E. and Noy N.F. "Simple part-whole relations in OWL Ontologies". *W3C 2005*: http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/index.html, Last accessed March 2010.

[128] Rasovska I., Chebel-Morello B. and Zerhouni N. "Process of s-maintenance: decision support system for maintenance intervention". *In: Proceedings of the 10th IEEE conference on emerging technologies and factory automation*, vol. 2, 2005, Catania, Italy. pp. 679-686

[129] TORNOS SA. http://www.tornos.com/

[130] Bangemann T., Rebeuf X., Reboul D., Schulze A., Szymanski J., Thomesse J.P., Thron M. and Zerhouni N. "PROTEUS-Creating distributed maintenance systems through an integration platform". *Computers in Industry*; 57 (6), pp. 539-551

[131] Rasovska I., Chebel-Morello B. and Zerhouni N. "A conceptual model of maintenance process in unified modeling language". *Proceedings at 11th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2004)*; April 5-7, 2004, Salvador, Brazil

[132] Karray M.H., Morello-Chebel B. and Zerhouni N. "Towards a maintenance semantic architecture". *The Fourth World Congress on Engineering Asset Management (WCEAM)*; 28-30 Sep. 2009 Athens, Greece. pp. 98-111. London: Springer Verlag.

[133] Matsokis A., Karray M.H., Morello-Chebel B. and Kiritsis D. "An Ontology-based Model for providing Semantic Maintenance". *1st IFAC workshop on Advanced Maintenance Engineering, Services and Technology (A-MEST'10)*; 1-2 July 2010, Lisbon, Portugal (Accepted)

[134] Matsokis A. and Kiritsis D. "Time-Centric Product Lifecycle Management System and Method for developing the Same". (PCT filing serial number: PCT/EP2010/053238)

[135] Matsokis A. and Kiritsis D. "An Ontology-based Approach for Product Lifecycle Management". *Computers in Industry;* 61 (8), pp.787–797

[136] Matsokis A. and Kiritsis D. "Ontology Applications in PLM". *International Journal of Product Lifecycle Management*; Accepted in 2010. In press.

[137] Matsokis A. and Kiritsis D. "An advanced method for time treatment in product lifecycle management models". *The Fourth World Congress on Engineering Asset Management (WCEAM)*, 28-30 Sep. 2009 Athens, Greece. pp. 120-126. London: Springer Verlag.

[138] Matsokis A., Zamofing S. and Kiritsis D. "A Ontology-based Modelling for Complex Industrial Asset Lifecycle Management: a Case Study". *In Proc. the 7th International Product Lifecycle Management Conference;* 12-14 July, 2010, Bremen, Germany, (Accepted)

[139] Matsokis A. and Kiritsis D. "Ontology-Based Implementation of an Advanced Method for Time Treatment in Asset Lifecycle Management". *The Fifth World Congress on Engineering Asset Management (WCEAM)*, (2010), Brisbane, Australia. (submitted)

[140] Brank J., Grobelnik M. and Mladenić D. "A Survey of Ontology Evaluation Techniques". *Eprints pascal-network 2005;* 00001198/, Last accessed March 2010.

[141] Obrst L., Ashpole B., Ceusters W., Mani I., Ray S. and Smith B. "The Evaluation of Ontologies: Toward Improved Semantic Interoperability". *National Institute of Standards*, 2007: http://www.mel.nist.gov/msidlibrary/doc/eval_ontologies.pdf, Last accessed March 2010.

[142] Guarino N. "Toward a formal evaluation of ontology quality". *IEEE Intelligent Systems 2004;* 19 (4), pp. 74-81 (under: Why evaluate ontology technologies? Because it works!)

[143] Methods for ontology evaluation. KnowledgeWeb FP6 deliverable D1.2.3: http://starlab.vub.ac.be/research/projects/knowledgeweb/KWeb-Del-1.2.3-Revised-v1.3.1.pdf, Last accessed March 2010.

[144] Gangemi A., Catenacci C., Ciaramita M. and Lehmann J. "Modelling ontology evaluation and validation". *Lecture Notes in Computer Science* (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 4011 LNCS 2006; pp. 140-154

[145] Gomez-Perez A. "Evaluation of Ontologies". *International Journal of Intelligent Systems 2001;* Volume 16, Issue 3, pp. 391-409

[146] Obrst L., Hughes T. and Ray S. "Prospects and Possibilities for Ontology Evaluation: The view from National Center for Ontological Research". *4th International EON Workshop (EON 2006) located at the 15th International World Wide Web Conference 2006;* http://km.aifb.uni-karlsruhe.de/ws/eon2006/eon2006obrstetal.pdf, Last accessed March 2010.

Appendix A: OWL Model full list of relationships and attributes per class

This section contains tables containing the full list of object and datatype properties per class of the model developed in section 4.2 (Figure 18). Table 8 contains the list of the classes and their object properties and Table 9 contains the list of the classes and their attributes.

Table 8: List of Object Properties

Domain Class	Object Property	Range Class
Access_Rights		
Activity	Activity2Life_Cycle_Phase	Life_Cycle_Phase
Activity	Involves	Resource
Activity	Causes	Event
As_Designed_Product	As_Designed_Product2Condition	Condition
As_Designed_Product	As_Designed_Product2Field_Data_Source	Field_Data_Source
As_Designed_Product	As_Designed_Product2Physical_Product_Group	Physical_Product_Group
As_Designed_Product	As_Designed_Product2Property	Property
As_Designed_Product	As_Designed_Product2Valid_Field_Data_Type	Valid_Field_Data_Type
As_Designed_Product	hasDefined	Physical_Product
Condition	Condition2As_Designed_Product	As_Designed_Product
Condition	Condition2Event	Event
Condition	Condition2Physical_Product	Physical_Product
Condition	Condition2Property	Property
Document	Document2Document_Resource	Document_Resource
Document	Document2Field_Data	Field_Data
Document	Document2File	File
Document_Resource	Document_Resource2Document	Document
Equiptment_Resource	Equiptment_Resource2Property	Property
Event	Event2Condition	Condition
Event	Event2Field_Data	Field_Data
Event	Event2Life_Cycle_Phase	Life_Cycle_Phase
Event	Event2Resource	Resource
Event	Triggers	Activity
Field_Data	Field_Data2Document	Document
Field_Data	Field_Data2Event	Event
Field_Data	Field_Data2Life_Cycle_Phase	Life_Cycle_Phase
Field_Data	Field_Data2Physical_Product	Physical_Product
Field_Data	Field_Data2Physical_Product_Group	Physical_Product_Group

Field_Data	Field_Data2Valid_Field_Data_Type	Valid_Field_Data_Type
Field_Data_Source	Field_Data_Source2As_Designed_Product	As_Designed_Product
Field_Data_Source	Field_Data_Source2ID_Information	ID_Information
Field_Data_Source	Field_Data_Source2Physical_Product	Physical_Product
Field_Data_Source	Field_Data_Source2Valid_Field_Data_Type	Valid_Field_Data_Type
File	File2Document	Document
ID_Information	ID_Information2Field_Data_Source	Field_Data_Source
ID_Information	ID_Information2Information_Provider	Information_Provider
ID_Information	ID_Information2Physical_Product	Physical_Product
ID_Information	ID_Information2URI	URI
Information_Provider	Information_Provider2ID_Information	ID_Information
Life_Cycle_Phase	Life_Cycle_Phase2Activity	Activity
Life_Cycle_Phase	Life_Cycle_Phase2Event	Event
Life_Cycle_Phase	Life_Cycle_Phase2Field_Data	Field_Data
Life_Cycle_Phase	Life_Cycle_Phase2Physical_Product	Physical_Product
Life_Cycle_Phase	Life_Cycle_Phase2Resource	Resource
Material_Resource	Material_Resource2Property	Property
Personnel_Resource	Personnel_Resource2Property	Property
Physical_Product	hasParent	Physical_Product
Physical_Product	isParentOf	Physical_Product
Physical_Product	Physical_Product2Condition	Condition
Physical_Product	Physical_Product2Field_Data	Field_Data
Physical_Product	Physical_Product2Field_Data_Source	Field_Data_Source
Physical_Product	Physical_Product2ID_Information	ID_Information
Physical_Product	Physical_Product2Life_Cycle_Phase	Life_Cycle_Phase
Physical_Product	Physical_Product2Physical_Product_Group	Physical_Product_Group
Physical_Product	Physical_Product2Property	Property
Physical_Product	Physical_Product2Resource	Resource
Physical_Product	Physical_Product2Valid_Field_Data_Type	Valid_Field_Data_Type
Physical_Product	isDesigned	As_Designed_Product
Physical_Product_Group	Physical_Product_Group2As_Designed_Product	As_Designed_Product
Physical_Product_Group	Physical_Product_Group2Field_Data	Field_Data
Physical_Product_Group	Physical_Product_Group2Physical_Product	Physical_Product
Physical_Product_Group	Physical_Product_Group2Valid_Field_Data_Type	Valid_Field_Data_Type
Property	Property2As_Designed_Product	As_Designed_Product
Property	Property2Condition	Condition
Property	Property2Equiptment_Resource	Equiptment_Resource
Property	Property2Material_Resource	Material_Resource
Property	Property2Personnel_Resource	Personnel_Resource
Property	Property2Physical_Product	Physical_Product
Resource	Resource2Activity	Activity
Resource	Manages	Event
Resource	Resource2Life_Cycle_Phase	Life_Cycle_Phase
Resource	Resource2Physical_Product	Physical_Product

URI	URI2ID_Information	ID_Information
Valid_Field_Data_Type	Valid_Field_Data_Type2As_Designed_Product	As_Designed_Product
Valid_Field_Data_Type	Valid_Field_Data_Type2Field_Data	Field_Data
Valid_Field_Data_Type	Valid_Field_Data_Type2Field_Data_Source	Field_Data_Source
Valid_Field_Data_Type	Valid_Field_Data_Type2Physical_Product	Physical_Product
Valid_Field_Data_Type	Valid_Field_Data_Type2Physical_Product_Group	Physical_Product_Group

Table 9: List of Datatype Properties

Domain Class	Datatype Property
Activity	ActivityDescription
Activity	ActivityDuration
Activity	ActivityFinishingDate
Activity	ActivityFinishingTime
Activity	ActivityID
Activity	ActivityStartingDate
Activity	ActivityStartingTime
As_Designed_Product	BoM
As_Designed_Product	CAD_Model
As_Designed_Product	Condition_State
As_Designed_Product	Costs_Information
As_Designed_Product	Materials_Information
As_Designed_Product	Product_State
As_Designed_Product	Product_Type_ID
As_Designed_Product	Property_State
As_Designed_Product	Tests_n_Specifications
As_Designed_Product	Variants_Information
Condition	Action_When_Met
Condition	Action_When_Not_Met
Condition	Condition_ID
Condition	Condition_Type_ID
Condition	ConditionDatatypeProperties
Document	Document_DatatypeProperties
Document	Document_ID
Document	Document_Type
Equiptment_Resource	Equiptment_Type
Equiptment_Resource	QA_test_and_Specification
Event	EventDatatypeProperties
Event	EventFlag
Event	Leaving_Product_State
Event	Time_Stamp
Event	Triggering_Condition
Event	Event_Name
Event	Entering_Product_State
Field_Data	Accuracy
Field_Data	Field_Data_ID
Field_Data	Field_Data_Type
Field_Data	Reference_Group_ID

Field_Data	Value
Field_Data	WHAT
Field_Data	WHEN
Field_Data	WHERE
Field_Data	WHO
Field_Data	Field_Data_Group_ID
Field_Data_Source	Source_ID
Field_Data_Source	Source_Type
File	File_DatatypeProperties
File	File_ID
File	File_Type
ID_Information	Alt_Pres
ID_Information	ID_Type
ID_Information	ID
Information_Provider	ID
Information_Provider	Alt_Pres
Information_Provider	ID_Type
Information_Provider	Type
Life_Cycle_Phase	Product_State_Its_Own
Life_Cycle_Phase	Starting_Date_Time
Life_Cycle_Phase	Finishing_Date_Time
Life_Cycle_Phase	Residual_Life
Material_Resource	Material_Lot
Material_Resource	Material_Type
Material_Resource	QA_test_and_Specification
Personnel_Resource	E_Mail
Personnel_Resource	Personnel_Type
Personnel_Resource	Qualification_test_and_Specification
Personnel_Resource	Telephone
Physical_Product	Birth_Date
Physical_Product	End_Date
Physical_Product	Object_Lot_ID
Physical_Product	Product_Complexity
Physical_Product_Group	Group_Code
Physical_Product_Group	Group_Type
Property	Property_Name
Property	Property_Value
Property	Category
Resource	Resource_Description
Resource	Resource_ID
Resource	Resource_Location
Resource	Resource_State
URI	URI_String
URI	URIDatatypeProperty

URI	Type
Valid_Field_Data_Type	Definition_Domain
Valid_Field_Data_Type	Measuring_Unit
Valid_Field_Data_Type	Valid_Field_Data_Type_Category
Valid_Field_Data_Type	Valid_Field_Data_Type_ID
Valid_Field_Data_Type	Value_Type

Appendix B: Merging of two or more OWL ontologies in OWL 1 and in OWL 2

While attempting to achieve ontology merging with OWL we faced several difficulties due to language limitations. As it is very well described by Grau et al. [47] in OWL 1 the merging of ontologies is not supported and this has been taken into account for the recently released OWL 2 which supports it. The concept is that experts would provide copies of the initial ontology to various partners to extend them according to their needs. Then these copies will be collected by the OEM and they will be imported into the initial ontology model. The requirement is that the final model with all the imported ontologies contains all the new elements which did not exist in the initial ontology model and at the same time duplicates of any elements are avoided. In this way we can apply the reasoner in the model as a whole in order to extract knowledge.

In theory

In OWL 1 the merging of ontologies is not supported directly. However, there is an indirect method to go around this limitation and achieve a similar to the desired result by importing one or more ontologies into the other. Of course, in this case the ontologies are not merged into one single ontology but they are called and opened on the same project. Thus, the reasoner may be applied on the project as a whole (this means that it is applied on the total number of DL rules, classes etc.) which is the desired result.

In OWL 2 after importing one or more ontologies into the other we are able to create a third ontology which will contain all the elements of the initial ontologies.

In Practice

In practice there is a technical problem which doesn't allow the user to acquire the desired result neither in OWL 1 nor in OWL 2. The technical problem derives from the fact that the initial ontology *O* is defined by the URI *U* and its copies have exactly the same URI *U*.

For example we have two copies of the initial model which are copy A and copy B. The result in OWL 1 is: when the initial model or one copy A is loaded on the Protégé-OWL

and then we load another copy B, the tool loads copy B and sees it as recursive ontology. This means that the tool sees the copy B as already loaded. The tool does not perform any type of comparison between copy A and copy B since it sees them like they are the same ontology. Therefore, the tool loads only the elements of the first ontology, which in this case is copy A. In the general similar case, the tool does not perform any type of comparison between the two or more copies (copy A and copy B) of the (same) ontology since it understands it as "already loaded" and hence, it does not load any element of the copy on the model.

In OWL 2 when the initial model (or one copy A) is loaded, we have to import the other ontology (copy B) on the tool and then save the final project as merged. However, the tool works similarly as in OWL 1 and sees the ontology as "already loaded" and the tool again does not load the elements of the copies. Therefore, a real merging of classes, instances, properties, DLs etc. cannot be performed.

Solution

The solution found was to change the URI of each copy to a unique URI combined with an ascending three digit number xxx in the end such as *U'*=http://www.owl-ontologies.com/Ontology1202459344_ Copy_001.owl, where xxx=001. All initial elements of the copy (classes, object and datatype properties, instances) keep their original URI *U* of the initial ontology *O*. Moreover, the default namespace of the copy is set to the URI *U* and therefore, all the new elements added to the copy have the namespace of the initial ontology *O*.

The process steps to be followed in order to achieve the desired result are:

1. Make copy of the initial ontology
2. Change its URI to *U'*
3. Set as its default namespace the URI *U* of ontology *O*
4. Share it with the partners/ Provide partners with copies
5. Partners extend their copies according to their needs following rules
6. Collect the different local copies
7. Merge them together (import one ontology into the other)

8. Execute the reasoner to find equivalencies, consistency and re-classification on classes and to categorise instances

For step 7 it should be noted that it is *indifferent* which ontology is loaded first in the tool and the loading order after it. In all cases all elements will be loaded.

This technique allows loading all the elements on the tool with both OWL 1 and OWL 2. It should be noted that in OWL 2 there are also other solutions i.e. create a new ontology and merge on it first the initial model and then the copies one by one. What actually happens is that we import ontology *O'* into *O* and then all the elements of both *O'* and *O* have the same URI *U* of ontology *O*.

Appendix C: SWRL rules for Case Study 2

This section contains the queries and rules of the case study 2 in section 5.2. Actually the second part of the rules is an SQWRL construct which displays the results in spreadsheet like format. Similar rules might be constructed in order to obtain various types of results.

The SWRL Query for Figure 40

```
Activity(?acx) ^
ActivityID(?acx, "Idle") ^
Maintenance_Machine(?mmx) ^
Activity2Resource(?acx, ?mmx) ^
Starting_Date_Time(?acx, ?zx) ^
Finishing_Date_Time(?acx, ?dx) ^
temporal:after(?dx, ?zx) ^
temporal:after(?dx, "2009-02-27T08:41:000") ^
temporal:before(?zx, "2009-02-27T08:41:000") ^
temporal:duration(?minute, ?zx, ?dx, temporal:Minutes)
→
sqwrl:select(?acx, ?mmx, ?zx, ?dx, ?minute) ^
sqwrl:orderBy(?acx) ^
sqwrl:columnNames("Instance", "Machine", "Birth Date ", "End Date", "Duration In Minutes ")
```

The SWRL Query for Figure 41

```
Activity(?acx) ^
ActivityID(?acx, "Idle") ^
Activity2Resource(?acx, Mechanic_B) ^
Starting_Date_Time(?acx, ?zx) ^
Finishing_Date_Time(?acx, ?dx) ^
temporal:after(?dx, ?zx) ^
temporal:after(?dx, "2009-02-27T06:31:000") ^
temporal:before(?zx, "2009-02-27T06:31:000") ^
temporal:duration(?minute, "2009-02-27T06:30:000", ?dx, temporal:Minutes)
→
sqwrl:select("Mechanic B", ?zx, ?dx, ?minute, ?acx, ?zx, ?dx, ?minute) ^
sqwrl:orderBy(?acx) ^
sqwrl:columnNames("Instance", "Birth Date ", "End Date", "Duration From Now In Minutes ")
```

The SWRL Query for Figure 42

```
Activity(?acx) ^
ActivityID(?acx, "Idle") ^
Activity2Resource(?acx, Document_Resource_3) ^
Starting_Date_Time(?acx, ?zx) ^
Finishing_Date_Time(?acx, ?dx) ^
temporal:after(?dx, ?zx) ^
temporal:duration(?minute, ?zx, ?dx, temporal:Minutes)
→
sqwrl:select(?acx, "Document_Resource_3", ?zx, ?dx, ?minute) ^
sqwrl:columnNames("Instance", "Resource", "Birth Date ", "End Date", "Duration In Minutes ")
```

The SWRL Query for Figure 43

```
Activity(?acx) ^
ActivityID(?acx, "Idle") ^
Resource(?rex) ^
Activity2Resource(?acx, ?rex) ^
Starting_Date_Time(?acx, ?zx) ^
Finishing_Date_Time(?acx, ?dx) ^
temporal:after(?dx, ?zx) ^
temporal:duration(?minute, ?zx, ?dx, temporal:Minutes)
→
sqwrl:select(?acx, ?rex, ?zx, ?dx, ?minute) ^
sqwrl:orderBy(?rex) ^
sqwrl:columnNames("Instance", "Resource", "Birth Date ", "End Date", "Duration In Minutes ")
```

Appendix D: SWRL rules for Case Study 3

This section contains the queries and rules of the case study 3 in section 5.3 and it is separated into two parts.

First Part

The first part contains examples of the SWRL rules which were used to give the right form to the imported instances from the excel spreadsheet. When one imports data from an excel spreadsheet, each row corresponds to an instance and the data contained in the different columns of the row corresponds to a datatype attribute. This creates instances without object properties which is not always desirable. In our case we needed to have object properties in each new instance relating it to other instances such as `hasParent`, `Physical_Product2Physical_Product_Group`, `Physical_Product2Property`, `Physical_Product2Function`, etc. To make this process we used SWRL rules which were imported in the Jess rule engine. The result of the Jess rule engine was returned back to the OWL-DL model (triangle in Figure 11):

- This rule relates the instances of the class **Parts_of_Deco_10** with an instance of the **Physical_Product_Group** class

Parts_of_Deco_10(?pdx) ^
//IF there is an instance (?pdx) of the class Parts_of_Deco_10
NUM_COMPONENT(?pdx, ?ncx) ^
//AND IF the instance (?pdx) has value (?ncx) for the datatype attribute NUM_COMPONENT
Nomenclature_Deco_10(?ppx) ^
//AND IF there is an instance of the class Nomenclature_Deco_10 (?ppx)
Group_Code(?ppx, ?gcx) ^
//AND IF the instance (?ppx) has value (?gcx) for the datatype attribute Group_Code
swrlb:equal(?ncx, ?gcx)
//AND IF the values (?ncx) and (?gcx) are the same
→
Physical_Product2Physical_Product_Group(?pdx, ?ppx)
//Then relate the instances (?pdx) and (?ppx) through the relationship Physical_Product2Physical_Product_Group

- This rule relates the instances of the class **Parts_of_Deco_10** with an instance of the **Property** class

Parts_of_Deco_10(?pdx) ^
//IF There is an instance (?pdx) of the class Parts_of_Deco_10
MOUVEMENT_GROUP(?pdx, ?ncx) ^
//AND IF the instance (?pdx) has value (?ncx) for the datatype attribute MOUVEMENT_GROUP
Property(?ppx) ^
//AND IF there is an instance of the class Property (?ppx)
Property_Value(?ppx, ?gcx) ^
//AND IF the instance (?ppx) has value (?gcx) for the datatype attribute Property_Value
swrlb:equal(?ncx, ?gcx)
//AND IF the values (?ncx) and (?gcx) are the same
→
Physical_Product2Property(?pdx, ?ppx)
//Then relate the instances (?pdx) and (?ppx) through the relationship Physical_Product2Property

- With similar rules all the instances are related to other instances of the model.

Second Part

The second part contains the SWRL rules which where developed to be applied on the Field Data. These rules refer to the defined conditions for creating events and alarms. The system reads the data and warns the user with a message about the possible events and alarms. Then, it is up to the user to evaluate the messages and create the events or alarms. Actually, it should be noted that there is a way to create automatically the instances of events and alarms in the system, which is shown at the end of this section. However, this method is not DL-safe and therefore, was not selected.

- **Condition:** T=63 degrees increasing, **Event:** Threshold of 63 degrees passed

Field_Data(?fdx) ^
//IF there is an instance (?fdx) of the class Field_Data
Field_Data(?fdy) ^
//IF there is an instance (?fdy) of the class Field_Data
Value(?fdx, ?vax) ^
//AND IF the instance (?fdx) has value (?vax) for the datatype attribute Value
Value(?fdy, ?vay) ^
//AND IF the instance (?fdy) has value (?vay) for the datatype attribute Value
Start_Date_Time(?fdx, ?stx) ^
//AND IF the instance (?fdx) has Start_Date_Time (?stx)
Start_Date_Time(?fdy, ?sty) ^

//AND IF the instance (?fdy) has Start_Date_Time (?sty)
Field_Data2Field_Data_Source(?fdx, Field_Data_Source_1) ^
//AND IF the instance (?fdx) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
Field_Data2Field_Data_Source(?fdy, Field_Data_Source_1) ^
//AND IF the instance (?fdy) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
temporal:after(?stx, ?sty) ^
//AND IF the instance Start_Date_Time (?stx) is after (?sty)
temporal:duration(30, ?stx, ?sty, temporal:Seconds) ^
//AND IF the "distance" between the two Start_Date_Time instances (?stx) and (?sty) is 30 seconds. With this rule we make sure that the two instances are one after the other.
swrlb:greaterThanOrEqual(?vax, 63) ^
//AND IF the value of (?vax) is greater than OR equal to 63
swrlb:equal(?vay, 60) ^
//AND IF the value of (?vay) is equal to 60
→
sqwrl:select(?fdx, ?vax, ?stx, "T=63 degrees increasing", "Threshold of 63 degrees passed", "No") ^
//Then display the list of the instances (?fdx) and their attributes (?vax) and (?stx) as well as the strings "T=63 degrees increasing", "Threshold of 63 degrees passed" and "No"
sqwrl:columnNames("Instance", "Value", "Start Date Time", "Condition", "Event", "Alarm")
//AND Then name the columns accordingly.

- **Condition:** T=67 degrees increasing, **Event:** Temperature higher than 67 degrees & increasing

Field_Data(?fdx) ^
//IF there is an instance (?fdx) of the class Field_Data
Field_Data(?fdy) ^
//IF there is an instance (?fdy) of the class Field_Data
Value(?fdx, ?vax) ^
//AND IF the instance (?fdx) has value (?vax) for the datatype attribute Value
Value(?fdy, ?vay) ^
//AND IF the instance (?fdy) has value (?vay) for the datatype attribute Value
Start_Date_Time(?fdx, ?stx) ^
//AND IF the instance (?fdx) has Start_Date_Time (?stx)
Start_Date_Time(?fdy, ?sty) ^
//AND IF the instance (?fdy) has Start_Date_Time (?sty)
Field_Data2Field_Data_Source(?fdx, Field_Data_Source_1) ^
//AND IF the instance (?fdx) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
Field_Data2Field_Data_Source(?fdy, Field_Data_Source_1) ^
//AND IF the instance (?fdy) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source

temporal:after(?stx, ?sty) ^
//AND IF the instance Start_Date_Time (?stx) is after (?sty)
temporal:duration(30, ?stx, ?sty, temporal:Seconds) ^
//AND IF the "distance" between the two Start_Date_Time instances (?stx) and (?sty) is 30 seconds
swrlb:greaterThanOrEqual (?vax, 67) ^
//AND IF the value of (?vax) is greater than OR equal to 67
swrlb:lessThan(?vay, 67) ^
//AND IF the value of (?vay) is less than 67
→
sqwrl:select(?fdx, ?vax, ?stx, "T=67 degrees increasing", "Temperature higher than 67 degrees & increasing", "Yellow") ^
//Then display the list of the instances (?fdx) and their attributes (?vax) and (?stx) as well as the strings "T=67 degrees increasing", "Temperature higher than 67 degrees & increasing" and "Yellow"
sqwrl:columnNames("Instance", "Value", "Start Date Time", "Condition", "Event", "Alarm")
//AND Then name the columns accordingly.

- **Condition:** 63<Tave≤67 for more than 2 minutes, **Event:** Temperature between 63<T<67 degrees for more than 2 minutes

Field_Data(?fdx) ^
//IF there is an instance (?fdx) of the class Field_Data
Field_Data(?fdy) ^
//IF there is an instance (?fdy) of the class Field_Data
Field_Data(?fdz) ^
//IF there is an instance (?fdz) of the class Field_Data
Field_Data(?fdw) ^
//IF there is an instance (?fdw) of the class Field_Data
Value(?fdx, ?vax) ^
//AND IF the instance (?fdx) has value (?vax) for the datatype attribute Value
Value(?fdy, ?vay) ^
//AND IF the instance (?fdy) has value (?vay) for the datatype attribute Value
Value(?fdz, ?vaz) ^
//AND IF the instance (?fdz) has value (?vaz) for the datatype attribute Value
Value(?fdw, ?vaw) ^
//AND IF the instance (?fdw) has value (?vaw) for the datatype attribute Value
Start_Date_Time(?fdx, ?stx) ^
//AND IF the instance (?fdx) has Start_Date_Time (?stx)
Start_Date_Time(?fdy, ?sty) ^
//AND IF the instance (?fdy) has Start_Date_Time (?sty)
Start_Date_Time(?fdz, ?stz) ^
//AND IF the instance (?fdz) has Start_Date_Time (?stz)
Start_Date_Time(?fdw, ?stw) ^
//AND IF the instance (?fdw) has Start_Date_Time (?stw)

Field_Data2Field_Data_Source(?fdx, Field_Data_Source_1) ^
//AND IF the instance (?fdx) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
Field_Data2Field_Data_Source(?fdy, Field_Data_Source_1) ^
//AND IF the instance (?fdy) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
Field_Data2Field_Data_Source(?fdz, Field_Data_Source_1) ^
//AND IF the instance (?fdz) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
Field_Data2Field_Data_Source(?fdw, Field_Data_Source_1) ^
//AND IF the instance (?fdw) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
temporal:after(?stx, ?sty) ^
//AND IF the instance Start_Date_Time (?stx) is after (?sty)
temporal:after(?sty, ?stz) ^
//AND IF the instance Start_Date_Time (?sty) is after (?stz)
temporal:after(?stz, ?stw) ^
//AND IF the instance Start_Date_Time (?stz) is after (?stw)
temporal:duration(30, ?stx, ?sty, temporal:Seconds) ^
//AND IF the "distance" between the two Start_Date_Time instances (?stx) and (?sty) is 30 seconds
temporal:duration(30, ?sty, ?stz, temporal:Seconds) ^
//AND IF the "distance" between the two Start_Date_Time instances (?sty) and (?stz) is 30 seconds
temporal:duration(30, ?stz, ?stw, temporal:Seconds) ^
//AND IF the "distance" between the two Start_Date_Time instances (?stz) and (?stw) is 30 seconds
swrlb:add(?sum, ?vax, ?vay, ?vaz, ?vaw) ^ swrlb:divide(?avg, ?sum, 4.0) ^
//Calculate the average (?avg)
swrlb:greaterThan(?avg, 63) ^
//AND IF the value of (?avg) is greater than 63
swrlb:lessThanOrEqual(?avg, 67) ^
//AND IF the value of (?avg) is less than OR equal to 67
→
sqwrl:select(?avg, "63<Tave<=67 for more than 2 minutes", "Temperature (average) 63<Tave<=67 for more than 2 minutes", "Yellow") ^
//Then display the average (?avg) and the strings "63<Tave<=67 for more than 2 minutes", " Temperature (average) 63<Tave<=67 for more than 2 minutes " and "Yellow"
sqwrl:columnNames("Average", "Condition", "Event", "Alarm")
//AND Then name the columns accordingly.

- **Condition:** T=70 degrees increasing, **Event:** Temperature higher than 70 degrees & increasing

Field_Data(?fdx) ^
//IF there is an instance (?fdx) of the class Field_Data

Field_Data(?fdy) ^
//IF there is an instance (?fdy) of the class Field_Data
Value(?fdx, ?vax) ^
//AND IF the instance (?fdx) has value (?vax) for the datatype attribute Value
Value(?fdy, ?vay) ^
//AND IF the instance (?fdy) has value (?vay) for the datatype attribute Value
Start_Date_Time(?fdx, ?stx) ^
//AND IF the instance (?fdx) has Start_Date_Time (?stx)
Start_Date_Time(?fdy, ?sty) ^
//AND IF the instance (?fdy) has Start_Date_Time (?sty)
Field_Data2Field_Data_Source(?fdx, Field_Data_Source_1) ^
//AND IF the instance (?fdx) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
Field_Data2Field_Data_Source(?fdy, Field_Data_Source_1) ^
//AND IF the instance (?fdy) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
temporal:after(?stx, ?sty) ^
//AND IF the instance Start_Date_Time (?stx) is after (?sty)
temporal:duration(30, ?stx, ?sty, temporal:Seconds) ^
//AND IF the "distance" between the two Start_Date_Time instances (?stx) and (?sty) is 30 seconds
swrlb:greaterThanOrEqual(?vax, 70) ^
//AND IF the value of (?vax) is greater than OR equal to 70
swrlb:lessThan(?vay, 70) ^
//AND IF the value of (?vay) is less than 70
→
sqwrl:select(?fdx, ?vax, ?stx, "T=70 degrees increasing", "Temperature higher than 70 degrees & increasing", "Yellow") ^
//Then display the list of the instances (?fdx) and their attributes (?vax) and (?stx) as well as the strings "T=70 degrees increasing", "Temperature higher than 70 degrees & increasing" and "Red"
sqwrl:columnNames("Instance", "Value", "Start Date Time", "Condition", "Event", "Alarm")
//AND Then name the columns accordingly.

- **Condition:** 67<Tave≤70 for more than 2 minutes, **Event:** Temperature between 67<T<70 degrees for more than 2 minutes

Field_Data(?fdx) ^
//IF there is an instance (?fdx) of the class Field_Data
Field_Data(?fdy) ^
//IF there is an instance (?fdy) of the class Field_Data
Field_Data(?fdz) ^
//IF there is an instance (?fdz) of the class Field_Data
Field_Data(?fdw) ^
//IF there is an instance (?fdw) of the class Field_Data

Value(?fdx, ?vax) ^
//AND IF the instance (?fdx) has value (?vax) for the datatype attribute Value
Value(?fdy, ?vay) ^
//AND IF the instance (?fdy) has value (?vay) for the datatype attribute Value
Value(?fdz, ?vaz) ^
//AND IF the instance (?fdz) has value (?vaz) for the datatype attribute Value
Value(?fdw, ?vaw) ^
//AND IF the instance (?fdw) has value (?vaw) for the datatype attribute Value
Start_Date_Time(?fdx, ?stx) ^
//AND IF the instance (?fdx) has Start_Date_Time (?stx)
Start_Date_Time(?fdy, ?sty) ^
//AND IF the instance (?fdy) has Start_Date_Time (?sty)
Start_Date_Time(?fdz, ?stz) ^
//AND IF the instance (?fdz) has Start_Date_Time (?stz)
Start_Date_Time(?fdw, ?stw) ^
//AND IF the instance (?fdw) has Start_Date_Time (?stw)
Field_Data2Field_Data_Source(?fdx, Field_Data_Source_1) ^
//AND IF the instance (?fdx) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
Field_Data2Field_Data_Source(?fdy, Field_Data_Source_1) ^
//AND IF the instance (?fdy) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
Field_Data2Field_Data_Source(?fdz, Field_Data_Source_1) ^
//AND IF the instance (?fdz) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
Field_Data2Field_Data_Source(?fdw, Field_Data_Source_1) ^
//AND IF the instance (?fdw) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
temporal:after(?stx, ?sty) ^
//AND IF the instance Start_Date_Time (?stx) is after (?sty)
temporal:after(?sty, ?stz) ^
//AND IF the instance Start_Date_Time (?sty) is after (?stz)
temporal:after(?stz, ?stw) ^
//AND IF the instance Start_Date_Time (?stz) is after (?stw)
temporal:duration(30, ?stx, ?sty, temporal:Seconds) ^
//AND IF the "distance" between the two Start_Date_Time instances (?stx) and (?sty) is 30 seconds
temporal:duration(30, ?sty, ?stz, temporal:Seconds) ^
//AND IF the "distance" between the two Start_Date_Time instances (?sty) and (?stz) is 30 seconds
temporal:duration(30, ?stz, ?stw, temporal:Seconds) ^
//AND IF the "distance" between the two Start_Date_Time instances (?stz) and (?stw) is 30 seconds
swrlb:add(?sum, ?vax, ?vay, ?vaz, ?vaw) ^ swrlb:divide(?avg, ?sum, 4.0) ^
//Calculate the average (?avg)
swrlb:greaterThan(?avg, 67) ^

//AND IF the value of (?avg) is greater than 67
swrlb:lessThanOrEqual(?avg, 70) ^
//AND IF the value of (?avg) is less than OR equal to 70
→
sqwrl:select(?avg, "67<Tave<=70 for more than 2 minutes", "Temperature (average) 67<Tave<=70 for more than 2 minutes", "Red") ^
//Then display the average (?avg) and the strings "67<Tave<=70 for more than 2 minutes", " Temperature (average) 67<Tave<=70 for more than 2 minutes " and "Red"
sqwrl:columnNames("Average", "Condition", "Event", "Alarm")
//AND Then name the columns accordingly.

- **Condition:** T=68 degrees decreasing, **Event:** Temperature higher than 67 degrees & decreasing

Field_Data(?fdx) ^
//IF there is an instance (?fdx) of the class Field_Data
Field_Data(?fdy) ^
//IF there is an instance (?fdy) of the class Field_Data
Value(?fdx, ?vax) ^
//AND IF the instance (?fdx) has value (?vax) for the datatype attribute Value
Value(?fdy, ?vay) ^
//AND IF the instance (?fdy) has value (?vay) for the datatype attribute Value
Start_Date_Time(?fdx, ?stx) ^
//AND IF the instance (?fdx) has Start_Date_Time (?stx)
Start_Date_Time(?fdy, ?sty) ^
//AND IF the instance (?fdy) has Start_Date_Time (?sty)
Field_Data2Field_Data_Source(?fdx, Field_Data_Source_1) ^
//AND IF the instance (?fdx) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
Field_Data2Field_Data_Source(?fdy, Field_Data_Source_1) ^
//AND IF the instance (?fdy) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
temporal:after(?stx, ?sty) ^
//AND IF the instance Start_Date_Time (?stx) is after (?sty)
temporal:duration(30, ?stx, ?sty, temporal:Seconds) ^
//AND IF the "distance" between the two Start_Date_Time instances (?stx) and (?sty) is 30 seconds
swrlb:greaterThanOrEqual (?vay, 68) ^
//AND IF the value of (?vay) is greater than OR equal 68
swrlb:lessThanOrEqual (?vax, 68) ^
//AND IF the value of (?vax) is less than OR equal to 68
swrlb:greaterThanOrEqual (?vax, 67) ^
//AND IF the value of (?vax) is greater than OR equal to 67
→
sqwrl:select(?fdx, ?vax, ?stx, "T=68 degrees decreasing", "Temperature higher than 67 degrees & decreasing", "Yellow") ^

//Then display the list of the instances (?fdx) and their attributes (?vax) and (?stx) as well as the strings "T=68 degrees decreasing", "Temperature higher than 67 degrees & decreasing" and "Yellow"
sqwrl:columnNames("Instance", "Value", "Start Date Time", "Condition", "Event", "Alarm")

//AND Then name the columns accordingly.

- **Condition:** T=63 degrees decreasing, **Event:** Temperature OK

Field_Data(?fdx) ^
//IF there is an instance (?fdx) of the class Field_Data
Field_Data(?fdy) ^
//IF there is an instance (?fdy) of the class Field_Data
Value(?fdx, ?vax) ^
//AND IF the instance (?fdx) has value (?vax) for the datatype attribute Value
Value(?fdy, ?vay) ^
//AND IF the instance (?fdy) has value (?vay) for the datatype attribute Value
Start_Date_Time(?fdx, ?stx) ^
//AND IF the instance (?fdx) has Start_Date_Time (?stx)
Start_Date_Time(?fdy, ?sty) ^
//AND IF the instance (?fdy) has Start_Date_Time (?sty)
Field_Data2Field_Data_Source(?fdx, Field_Data_Source_1) ^
//AND IF the instance (?fdx) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
Field_Data2Field_Data_Source(?fdy, Field_Data_Source_1) ^
//AND IF the instance (?fdy) is related to the instance "Field_Data_Source_1" through the relationship Field_Data2Field_Data_Source
temporal:after(?stx, ?sty) ^
//AND IF the instance Start_Date_Time (?stx) is after (?sty)
temporal:duration(30, ?stx, ?sty, temporal:Seconds) ^
//AND IF the "distance" between the two Start_Date_Time instances (?stx) and (?sty) is 30 seconds
swrlb:greaterThan (?vay, 63) ^
//AND IF the value of (?vay) is greater than 63
swrlb:lessThanOrEqual (?vax, 63) ^
//AND IF the value of (?vax) is less thanOR equal to 63
→
sqwrl:select(?fdx, ?vax, ?stx, "T=63 degrees decreasing", "Temperature OK", "No") ^
//Then display the list of the instances (?fdx) and their attributes (?vax) and (?stx) as well as the strings "T=63 degrees decreasing", "Temperature OK" and "No"
sqwrl:columnNames("Instance", "Value", "Start Date Time", "Condition", "Event", "Alarm")

//AND Then name the columns accordingly.

As it is described in the rules above, the alarms and events are not created automatically, but they are added by the user. The problem is that the machine might create multiple

alarms by using the same data. For example, if one fires the rules on a data set A then a group of alarms is created. Then, if one loads more data on the model and fires the rule again on the whole data, then the instances describing the alarms of the data set A will be created again. Thus, in the case that we fire the rule on the same data n times we will get n instances of the same alarm originating from the same field data, which is undesirable.

A rule to create a red alarm (of the type Temperature higher than 70 degrees & increasing) is shown below:

Field_Data(?fdx) ^
Field_Data(?fdy) ^
Value(?fdx, ?vax) ^
Value(?fdy, ?vay) ^
Start_Date_Time(?fdx, ?stx) ^
Start_Date_Time(?fdy, ?sty) ^
Field_Data2Field_Data_Source(?fdx, Field_Data_Source_1) ^
Field_Data2Field_Data_Source(?fdy, Field_Data_Source_1) ^
temporal:after(?stx, ?sty) ^
temporal:duration(30, ?stx, ?sty, temporal:Seconds) ^
swrlb:greaterThanOrEqual(?vax, 70) ^
swrlb:lessThan(?vay, 70) ^
swrlx:makeOWLIndividual(?inst, ?fdx)
//make a new instance (?inst) for each Field Date (?fdx) that fulfils the above criteria
→
Alarm(?inst) ^
//make (?inst) an instance of Alarm class
Field_Data2Event(?fdx, ?inst) ^
//Relate the instance (?fdx) to the instance (?inst) through the relationship Field_Data2Event
Alarm_Flag(?inst, “Red”) ^
//Red Alarm
Time_Stamp(?inst, ?stx)
//The time-date is the same as the one of the Field Data (?fdx) that created it.

Curriculum Vitae

Aristeidis Matsokis

Dipl.-Ing. Electrical Engineer

Address:
Chemin des Avelines 5
CH-1004. Lausanne e-mail: amatsokis@hotmail.com

Background

Aristeidis Matsokis has received his Diploma in Electrical Engineering in 2006 from the University of Patras, Greece. Since 2006 he is working on his PhD Thesis and as a research assistant in the Computer-Aided Design and Production Laboratory (LICP) of the Swiss Federal Institute of Technology in Lausanne (EPFL).

Education

- **Oct 2006-Oct 2010**: PhD thesis in Manufacturing Systems and Robotics in EPFL
- **Sep 2000-Mar 2006**: Diploma in Electrical Engineering, University of Patras, Greece

Experience-Projects

Research assistant

- **IMS 2020 Roadmap**: aims at developing a roadmap on Global Sustainable Manufacturing, Products and Services for 2020.
- **SMAC Interreg IV** project: semantic-maintenance and life cycle project aims at improving the performance of products through using semantic systems and techniques in maintenance. In this project we developed the ontology model for semantic maintenance.

Teaching assistant

In European Global Product Realisation (E-gpr)

- Feb 2009-June 2009 (Coaching group of students on semester project)
- Feb 2008-June 2008 (Coaching group of students on semester project)
- Feb 2007-June 2007 (Coaching group of students on semester project)

Publications

Scientific Journals

- Matsokis A. and Kiritsis D. **Ontology Applications in PLM**. *International Journal of Product Lifecycle Management*; (accepted, in press)
- Matsokis A. and Kiritsis D. **An Ontology-based Approach for Product Lifecycle Management.** *Computers in Industry;* 61 (8), pp.787–797

Conference Proceedings

- Matsokis A. and Kiritsis D. **Ontology-Based Implementation of an Advanced Method for Time Treatment in Asset Lifecycle Management.** *Proceedings in the 5th World Congress in Engineering Asset Management (WCEAM 2010)*; 25-27 Oct. 2010, Brisbane, Australia (accepted)
- Matsokis A., Zamofing S., and Kiritsis D. **Ontology-based Modelling for Complex Industrial Asset Lifecycle Management: a Case Study**. *Proceedings The 7th International Conference on Product Lifecycle Management (PLM10)*; 12-14 July, 2010, Bremen, Germany (in press)
- Matsokis A., Karray M.H., Morello-Chebel B. and Kiritsis D. **An Ontology-based Model for providing Semantic Maintenance**. *1st IFAC workshop on Advanced Maintenance Engineering, Services and Technology (A-MEST'10)*; 1-2 July 2010, Lisbon, Portugal (in press)
- Matsokis A. and Kiritsis D. **An Advanced Method for Time treatment in Product Lifecycle Management Models.** *Proceedings in the 4th World Congress in Engineering Asset Management (WCEAM 2009)*; 28-30 Sep. 2009, Athens, Greece. pp. 120-126. London: Springer Verlag.
- Matsokis A. and Kiritsis D. **Ontology Applications in PLM**. *Proceedings in The 6th International Conference on Product Lifecycle Management (PLM09)*; 6-8 July, 2009, Bath, UK (in press)

Patents

Matsokis A. and Kiritsis D. **Time-Centric Product Lifecycle Management System and Method for developing the Same**. (PCT filing serial number: PCT/EP2010/053238).

Languages

English	**Greek**	**German**	**French**	**Spanish**
Fluent, CPE	Native	Intermediate: Mittelstufe	Basic:B1	Basic:A2

Memberships

- Technical Chamber of Greece
- ACIDE
- Founding member of AEGEL: Association des Etudiantes Grecs de Lausanne