TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Minna Lanz

Logical and Semantic Foundations of Knowledge Representation for Assembly and Manufacturing Processes

Julkaisu 903 • Publication 903

Tampere 2010

Tampereen teknillinen yliopisto. Julkaisu 903
Tampere University of Technology. Publication 903

Minna Lanz

Logical and Semantic Foundations of Knowledge Representation for Assembly and Manufacturing Processes

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Festia Building, Auditorium Pieni Sali 1, at Tampere University of Technology, on the 6th of July 2010, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2010

ISBN 978-952-15-2393-9 (printed)
ISBN 978-952-15-2410-3 (PDF)
ISSN 1459-2045

Abstract

Partly due to the increase in global competition, the nature of manufacturing paradigms in Europe has been undergoing continuous change in recent decades. Pressures from the market are forcing companies to evolve into new entities of development and innovation. The trend of outsourcing to lower-wage countries, global market situation and geographically dispersed operations are imposing new challenges on manufacturing companies to become collaborative elements in the scheme of the supply chain. It is no new news to anyone that problems are arising when manufacturing experts are situated in Europe, design teams are spread around the globe and the operations are performed somewhere in Asia.

In order to keep up with the competitors the knowledge of the product itself, possible production facilities, and suitable processes has to be shared amongst the teams with ever-increasing speed.The emergence of highly computerised design and control systems is changing the fundamental assumptions of how product and production should be planned and controlled. Centralised control and decision-making systems based on hierarchical control structures and tightly-coupled system interfaces are giving way to globally distributed networks that both enable and require localised and fast adaptation to changes.

There are two problems identified in this thesis that are closely related to the decision-making systems and knowledge share among those. The first problem is the amount of information stored inside companies information systems and its lack of meaning. Especially in the design of products and production systems the proprietary design models are converted from one model to another numerous times. Because of the lack of interoperability between systems, only the basic geometry is exchanged and the rest of the information may or may not be transferred manually to the new design. The discontinuous knowledge flow often results in several different and contradictory models of the same design. This also leads to a situation where the re-use of knowledge becomes a slightly hazardous task, since before the existing knowledge is used someone must verify that all of the disconnected pieces are valid.

The second problem emerges when the product knowledge is used as a set of requirements for the design of the production system. In order for the production to be launched as soon as possible, the production plan is usually verified with a decision support system. However, since decision support systems, either on the design level or the actual production control level, rely on the input knowledge, the lack of it naturally undermines the reliability of the decision making. This is one of the greatest technological barriers to the implementation of the greatly desired complex and adaptive production environment, since without the meaning of the models that are used the reasoning becomes almost impossible.

As several other challenges to implementing an integrated collaborative and dynamically adaptive production environment exist, only two challenges are taken into focus in this thesis. Both challenges have multiple facets, but they do share a common need: a formal rigorous knowledge representation. The approach introduced here aims to create a common knowledge representation between the product, process, and system domains. The aim is not to model everything that can be included in a knowledge representation but to model the common core components and their relations.

The approach in this thesis starts from a feature-based modelling and analysis method and utilises the detailed product knowledge as the core of the knowledge representation. The process-, and system-level knowledge is represented with the necessary detail and the amount of it that is needed that are suitable for a given situation. The thesis introduces a knowledge representation (KR) for combining design information from several different sources into one reference architecture, which can be accessed via a common interface. The chosen approach is tested by means of three separate cases, which validate the KR from different perspectives. The first case study will evaluate the feasibility of utilising geometric and non-geometric features as the prerequisites for process modelling. The second case study evaluates the integrity and expressiveness of the KR by integrating the input from several commercial clients into one representation. The third case study evaluates the KR in a real production environment, where the KR will provide the knowledge for a holonic manufacturing system and save the events and device parameters into the history of the product that is realised after the operations are completed.

Preface

The purpose of this thesis is to provide a description of one approach to adding meaning to the models used in and between the design and manufacturing environments. The thesis focuses on the reasons behind the chosen approach and its results in the form of three separate case studies.

The literature offers multiple solutions for knowledge representations in design and manufacturing environments. Most of those representations focus on specific areas such as product knowledge representations, process descriptions, or hardware-level knowledge representations. This work adds input to the holistic approach, in which some of the previous approaches are combined and developed further to fit the needs of the problem areas. Naturally, this approach does not intend to solve all of the problems related to the field of complex systems, but to provide a novel step for future developments.

Acknowledgements

The doctorate is a journey. For parts of it one walks alone and for parts of it one has company to discuss the best route. For me this journey has been a true learning experience ranging from exhilarating boosts of motivation to sometimes mind-boggling levels of frustration, nonetheless, the journey was never boring.

One person can very rarely complete this kind of fundamental yet practical work single handedly. I would like to hereby express my sincere gratitude to my fellow researchers at the Tampere University of Technology (TUT) and at the National Institute of Standards and Technology (NIST) who had the time and patience to discuss and sometimes to argue and debate the possible and impossible options with me during my research. I cannot possibly list all those who contributed to the work. So, I thank You all, even if I only name a few of you here.

I would like to express my utmost gratitude for professor Erno Keskinen's Graduate School and Department of Production Engineering for this opportunity, challenge and freedom to choose a topic that allowed me to explore, learn and grow at my own pace.

I want to thank my supervisor professor Reijo Tuokko for his confidence in my efforts and the success of this work. I am grateful to my external evaluators professor Michel Cotsaftis from Ecole Centrale d'Electronique (ECE) France and Dr. Albert Jones from NIST for good comments and interesting discussions. Your feedback has been a truly great source of motivation towards the final phases of the work.

My sincere thanks go to Ricardo Velez for creating the beginning and to Fernando Garcia, Eeva Järvenpää, Roberto Rodriguez, Timo Kallela, and Pasi Luostarinen - you are the best team one can hope to be part of.

I want to thank my mother Johanna for love and support during these past years and Kimmo for reminding me about the facts of engineering - yes, pushing with a rope rarely works.

Lastly I want to thank my beloved husband Atte Joutsen for simply being there for me, supporting me when the motivation took a dive and rejoicing with me when a challenge was met.

Minna Lanz
Tampere, 2nd of June 2010

Contents

List of Figures

List of Tables

Abbreviations and Acronyms

ADACOR	ADAptive holonic COntrol aRchitecture for distributed manufacturing systems
AI	Artificial Intelligence
ASDM	Assembly State Decomposition Model
BMS	Bionic Manufacturing System
BOM	Bill of Material
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CAPP	Computer Aided Process Planning
CAx	Computer Aided X
CMSD	Core Manufacturing Simulation Data
CoreOnto	Domain Ontology inside the Knowledge Base
DAML-OIL	DARPA Agent Markup Language - Ontology Interchange Language
DB	Database
DiMS	Distributed Manufacturing System
DL	Description logics
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering
eBOM	engineering Bill of Material
ER	Entity Relationships
FOL	First Order Logic
FrMS	Fractal Manufacturing System
FRS	Frame Representation systems
GFP	Generic Frame Protocol
HMS	Holonic Manufacturing System
IEC	International standard for Enterprise-Control system integration
ISO	International Organization for Standardization
JADE	Java Agent Development Framework
JSON	JavaScript Object Notation
KB	Knowledge Base

KBS	Knowledge-based System
KIF	Knowledge Interchange Format
KR	Knowledge Representation
KRF	Knowledge Representation Formulation (method)
mBOM	manufacturing Bill of Material
MSDM	Manufacturing State Decomposition Model
OEM	Original Equipment Manufacturer
ONTOMAS	Ontology for the design of Modular Assembly Systems
OWL	Web Ontology Language
PL	Procedural Language
Pro-FMA	Feature Recognition tool
PPS model	Product-Process-System Model
PSL	Process Specification Language
RDF	Resource Description Framework
STEP	Standard for Exchange of Product model data
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
UML	Universal Modeling Language
VRML	Virtual Reality Modeling Language
W3C	WWW community
X3D	eXtensive 3D, successor of VRML format
XML	eXtensive Mark-up Language

Logical Operators

$\forall$	for all
$\exists$	there exist
$\wedge$	and
$\vee$	or
$\neg$	not
$<$	partOf
$\rightarrow$	if
$\leftrightarrow$	if and only if

Chapter 1

Introduction

The global economy has caused product design, process planning, and production to be more and more distributed around the world. Previously the product design, process planning, and manufacturing were located close together and the knowledge exchange between the design domains was on paper, as illustrated in Figure 1.1.

As the distribution of design, planning, and operations took place, close collaboration became more difficult. Today design teams are located on every continent and the design process is going on around the clock; see Figure 1.2. In such a geographically and temporally divided environment, effective and proficient collaboration between design and manufacturing teams and the factory floor is crucial in maintaining product quality, production efficiency, and organisational competency.

Global competition has changed the nature of manufacturing paradigms in recent decades. Turbulent production environments, short product life-cycles, and the frequent introduction of new products require more adaptive sys-

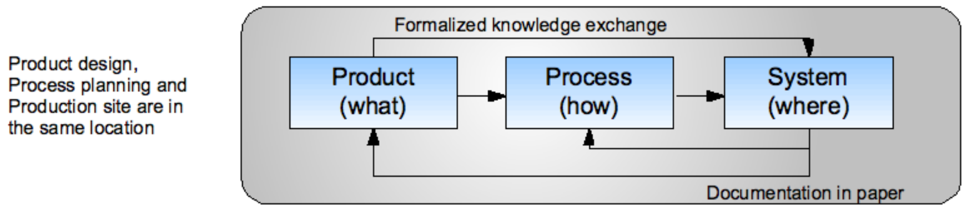
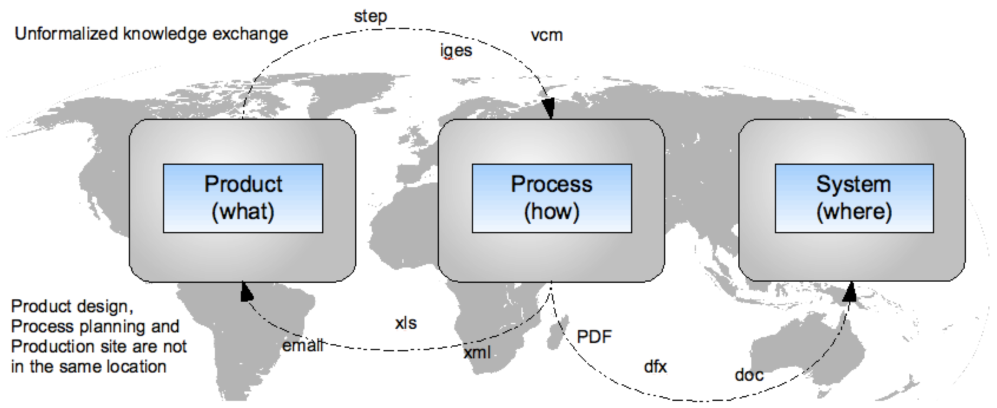


Figure 1.1: Product design, process planning, and production were previously closer to each other than today



Unformalized knowledge exchange
step
iges
vcm
Product (what)
Process (how)
System (where)
Product design, Process planning and Production site are not in the same location
xls
email
xml
PDF
dfx
doc

Figure 1.2: Distribution of product design, process planning, and production around the globe

tems that can rapidly respond to the required changes, whether the changes are based on product design changes or changes in the production it self. The emergence of highly computerised design and control systems is changing the fundamental assumptions of how production should be planned and controlled. Centralised systems based upon the need to share data by point-of-need acquisition are giving way to global distributed design and production networks that both enable and require localised and fast adaptation to production changes. Because of the resulting greater complexity of heterogenous interacting components, the production system can no longer be guided and controlled only on the basis of the sole capability of a human operator predicting the future state of a dynamic system on the basis of static models of its initial state [10, 12, 13, 58].

European industry has started to acknowledge that traditional approaches are reaching the limits of their technological and computational feasibility. Complexity science offers the tools to confront these challenges head on by emphasising a shift from purely logic-based rational design to a distributed design approach harnessing the capacity of self-organisation that is adapted to the natural complexity and changeability of the real world, both natural and man-made [10, 12, 13].

The adaptivity, proactivity, and self-organisation of production entities cannot be reached without new intelligent information systems[1] that are capa-

[1]The new intelligent information systems refer to tomorrow's systems that can, semi-automatically or even automatically, utilise the information saved into current IT-systems

ble of utilising and reasoning with the information saved into the current information and design systems used in various phases during the product life-cycle. The development of these kinds of systems presents a very ambitious challenge, not only because of the underlying logic, but also because of the knowledge acquisition. The reasoning within these systems relies on the information obtained from the current IT systems. In order to perform reasoning, the intelligent system has to query information from other systems. Again, in order to do that, the control system needs to know what kind of information resides inside the design systems providing the required information [31].

In order to develop production systems into proactive and adaptive smart factories [10], there is a need to look to the grassroots level, because the requirements tend to be somewhat high for the information systems available on the market today. The main problem is that the models and documents created with various systems are meaningless regarding their knowledge content for systems other than their authoring system. Where are the intelligent information systems supposed to get the knowledge they need to perform automatic reasoning? How can the reasoning be done if there is no knowledge or even information available?

1.1 Business-Level Problems

At the moment, the industrial world has produced numerous design systems for different processes. Currently, companies are storing almost all of the information they create via computers. Servers are saving, replicating, and archiving the collected information non-stop. The amount of stored information today is, to put it simply, huge [14].

The storing of the information or size of the models[2] are no longer seen as grand challenges. The problem today is the meaning of the stored information. The design knowledge remains locked in the authoring system. In a large-scale company there can be up to hundreds of different design support systems, versions, and ad hoc applications built on top of those which are

for example in order to perform dynamic production planning and reconfiguration or to automatically design production systems by mapping the product and process requirements to the capabilities provided by the system components.

[2]The term 'model' is used to refer any product, process, or resource representation created using CAx systems.

used to create the information of the current product, process, and/or production systems. All of the systems use their proprietary data structures and vaguely described semantics. This leads to challenges in sharing information, since none of these are truly able to share data beyond geometric visualisations.

This leads to several problems from the knowledge management point of view. The first problem is that the communication and cooperation between departments' "design domains" becomes time-consuming. Current modelling systems and data transfer standards do not support design collaboration effectively in terms of the capture, formulation, exchange, and integration of design information and knowledge within the design and manufacturing cycle. In most cases, in order to exchange information the models have to be created over and over again to be suitable for use in another design domain. The actual exchange of design information is not done via models, but rather with other design documents such as MS Word, MS Excel, or email [31].

The second problem is that with every remake and update the model actually loses information. This happens because the second tier does not require all of the information created in the first phase and when there is a remake only what the second tier sees as important information is saved. Depending on the purpose of the design task to be solved, the entity being modelled is viewed from different viewpoints or aspects. For example, if one wants to analyse the manufacturability of a product, the designer is interested in different aspects from those considered by someone who is analysing the functionality or appearance of the product. The entity being modelled is the same - the product - but the viewpoint is completely different and, therefore, the model is also different. Because of this fact, during the product development process, an incredible number of different plans, models, and other documentation are created. Moreover, these models do not have any real connection to each other, even though some of the information, 3D geometry for example, is the same and the changes made on the geometry level will affect both of the models.The result is that there are multiple sources contributing specific knowledge to several isolated models and revisions instead of to a *"master mode"l*, which is the main model each party uses and to which it contributes results.

Järvenpää et. al. [31] stated on the basis of a field study in a large global company that the phases in product and process design - roughly divided into product design, process planning, and production system design - are closely related. The phases utilise the requirements from each other. But the documentation created during these design phases, and their many sub-phases, is hardly connected. The next phase in the product development process usually creates its documentation from scratch or by manually copy-pasting from other documentation. This leads to a serious problem with updating the models and documents.

The updating affects only those very models or documents that are being updated, while those created on the basis of them become obsolete. One example of such a "snapshot" design approach is making the assembly process plans or work instructions with MS Excel or such and taking several snapshots of the CAD model, authored, for example, in CATIA. The resulting process plan is like a comic book, which is not dynamic, and the usability and efficiency of the plan are greatly affected by the viewpoints from where the snapshots are taken [31]. As soon as the CAD model is updated, the assembly process plan or work instruction documentation becomes obsolete, because these two models do not have any integration. However, in reality these obsolete documents continue their existence and in many cases are used as the main knowledge source for part of the design team.

The third problem, which follows from these two, is the re-use of the knowledge. At the moment, the re-use of existing information is quite impossible because of the several models that are meaningless and, in most cases, contradictory to the corresponding design documentation. For example, the production planning knows that there are several stations, robots, and grippers that can be re-used in the production of the next product family. They are also aware that these need only minor modification. However, despite the amount of stored information and number of existing devices, there is no up-to-date complete and easily available information on the interfaces, life-cycle data, and dimensions of the station and its components. The new stations or lines are therefore designed by starting with a new design project and an empty factory floor.

There is great potential for the re-use of the models and corresponding physical components. But once again the problem is that there is no formal definition, such as an ontology, which would explain what the system is, what its capabilities are, how long it has operated, and, moreover, what the virtual model used to describe it is.

The fourth knowledge management-related problem shifts the knowledge management and representation problem to a process design problem. Since the design and production phases of the manufacturing process are the next tiers in the knowledge flow, the knowledge management problems directly affect the efficiency of process design and manufacturing. The fourth problem is actually the utilisation of knowledge that has been created as a basis for semiautomatic reasoning about manufacturing system design and production control. Reasoning based on non-existent knowledge is somewhat impossible.

Because of the problems discussed above, decision support systems are at best unreliable and rely heavily on their users' expert knowledge. This is one of the greatest technological barriers to the implementation of a complex and adaptive production environment[3], since without the meaning of the knowledge or as a result of the lack of suitable models the reasoning becomes virtually impossible. There are several other barriers to achieving an integrated collaborative and dynamically adaptive production environment as well, but from the information-sharing point of view the greatest is the lack of formal knowledge representations. This often forces the experts to manually transfer information between otherwise automatable functions, with the concomitant introduction of errors, ambiguity, and misinterpretation. This also leaves production planning and control systems unable to share the information with the required speed and accuracy. It can be argued that the problem with decision making is somewhat solved, unfortunately at considerable expense, as a result of highly customised information mapping using translations and ad hoc implementations knit together. The result is often an unstable structure of mainly shallow (meaningless) information and a system that does not meet the requirements set by the industry.

1.2 Technical-Level Problems

There exist curious mixture of international and national standards and models that are making the unambiguous knowledge exchange difficult, error prone and time consuming. This underlines the need for a clear, unambiguous, and standardised interoperability infrastructure. As Ray and Jones stated [57], such an infrastructure still does not exist.

However, there are three approaches that have been used to create such a knowledge exchange infrastructure. The first approach is the point-to-point

[3]The interest of this thesis lies in next-generation manufacturing systems that are recognised as dynamic and evolving systems. These systems are envisioned as being highly adaptive entities where the adaptation relies on the knowledge representation of these systems.

customised solution, where dedicated interfaces are created between the design tools. In the long run, as versions change, the maintenance of each of the interfaces becomes expensive. The second approach is the one-size-fits-all solution decided by the proprietary interface of the OEM for design and planning and knowledge exchange between parties. While this is very cost-efficient for the OEM, it causes nightmarish expense for their partners who are working with several OEMs. Moreover, the global outsourcing has also made this approach practically impossible to implement and maintain over a longer period of time. The third solution is a neutral and open reference architecture based on published standards or internationally accepted models. Naturally, this architecture has to have well-defined meanings associated with the information entities of the model. Without explicit, rigorous definitions of the meaning of those entities, there is a great danger of continuing along the path of misunderstandings.

To represent the meaning of models, in recent years, industrial standards have been defined in a more computer-readable form, most notably since the emergence of eXtensive Mark Up (XML)-based formats and computing power. As a language, XML has a number of advantages for developers and implementers, because these specifications can be compiled by computers, databases can be built automatically, and certain kinds of testing can be performed more easily. According to the observations by Ray [56], along the way XML markups have been used as a substitute for modelling the information - a dangerous shortcut that only works in communities that already share a common understanding and representation of the meaning and usage of terms. A far better approach to integration is to adopt one of the emerging semantic technologies, such as Web Ontology Language (OWL), or first-order logic [56].

This evolution of representational power toward formal semantics, and the systems integration capabilities that could follow, are shown in Figure 1.3. The lowest level in the figure shows the current state of the art, where the XML-based standards are utilised with relative ease within the IT sector, but not fully utilised in more conservative industry sectors. The second step, formal semantics, offers the generation of standardised representation that is formal enough to be parsed with computers. The third step is self-describing systems, where the systems can provide formal descriptions of their content and interfaces. This requires a formal semantic definition language that is rigorous enough to support logical inference.

The fourth level that Ray [56] proposes is self-integrating systems. These

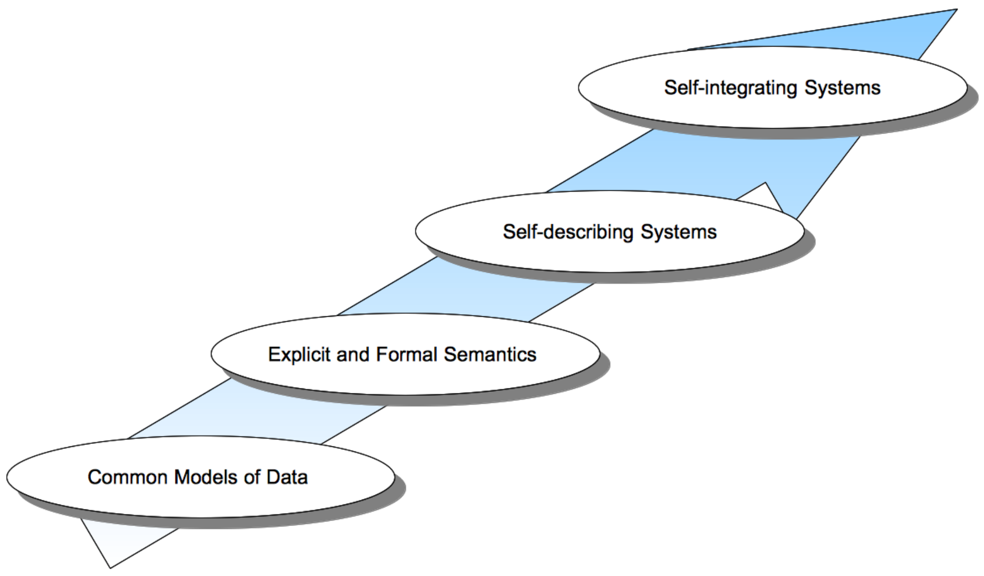


Figure 1.3: Arrow of Progress, modified from [56]

systems are intelligent enough to be able to ask others for a description of their interfaces and, on the basis of the information thus acquired, adjust their own interfaces to be able to exchange information.

However, because of the clear separation of design domains and the lack of integrated knowledge representation, the utilisation of XML tends to stay inside the domain boundaries. The boundaries of design domains are, historically, the boundaries of each design system. Modern design systems could utilise XML as a generic representation language; however, its use is limited to communication inside specified software suites or generating cartoon-type design documents. But they cannot deliver the meaning which might exist inside the design system itself outside that system. So far there have been very few design system vendors who have adopted XML-based representations as a means of facilitating loose integration with other systems.

From the product knowledge representation perspective there exist several feasible approaches for proposing a partial solution for the first problem area introduced in the previous subchapter. Scientifically-oriented interest groups, in cooperation with academia and industry, have produced numerous different internationally recognised standards, de facto standards and software suite-specific international models. For product knowledge representation there are the well-known product knowledge-sharing formats such as the

Standard for the Exchange of Product Model Data (STEP) and its extensions.

For process and system descriptions there are also different standards addressing the needs of those domains, such as Process Specification Language (PSL) and Core Manufacturing Simulation Data (CMSD). However, these standards - as well as they are defined - can be seen as *'islands of standards'*, since there are no models for representing all of these domains under one architecture. In most of the implementations, there are overlaps between standardised models, but again they are not interconnected. A second note has to be made; the standards have evolved over the years and have become very complex sets of extensions, dedicated to one viewpoint and serving mainly that specified viewpoint. Different implementations of standards linked to each other exist, but very often these form a framework for an expert system, where the actual reasoning and design rationale are embedded into the model itself. Additionally, the specified yet detailed models are often too specialised for one domain to be used elsewhere.

However, the extensions seem to ruin the whole idea of standards, because with each extension from a different implementer from a different point of view, the standard loses its role and strength. New standards or the combination of existing standards for exchanging both the geometric and non-geometric product and process definition data are desperately needed. In order for the standard to be efficient and reach its goals, it has to be respected and followed by as many users as possible. Still, for one model to rise above others does not seem to be possible because of the impressive number of different models [31].

Chapter 2

Research Objective

2.1 Formulation of the Research Objective

The common ground for the business problems discussed in Chapter 1 is the lack of meaningful, sufficiently rigorous, and formalised knowledge representation (KR) that is capable of expressing the details of the design as well as serving as a link and explanation for external material. It is understood that the KR that is developed cannot include all the design information of all models, but it must be able to provide metadata on those entities which cannot be included into the inner structure of the KR. These kinds of entities include proprietary formats that are not feasible to open on the geometric feature level, and offline or online programs related to the models.

In order to provide such a KR, this thesis formalises a generic knowledge representation that defines the possible connection between product, process, and system. The structure of the intended knowledge representation (KR) is formed on the basis of:

1. the requirements set by the knowledge management challenge between different design tools, and

2. the requirements set by the adaptive and complex production environment, illustrated in Figure 2.1.

It is important for a generic, scalable, and yet expressive KR that it provides the necessary number of relations between classes. The requirements for a generic model comes from the semantic interoperability challenges between and within domains. Since the development is also intended for human interpretation, the class structure needs to be relatively easy to understand and

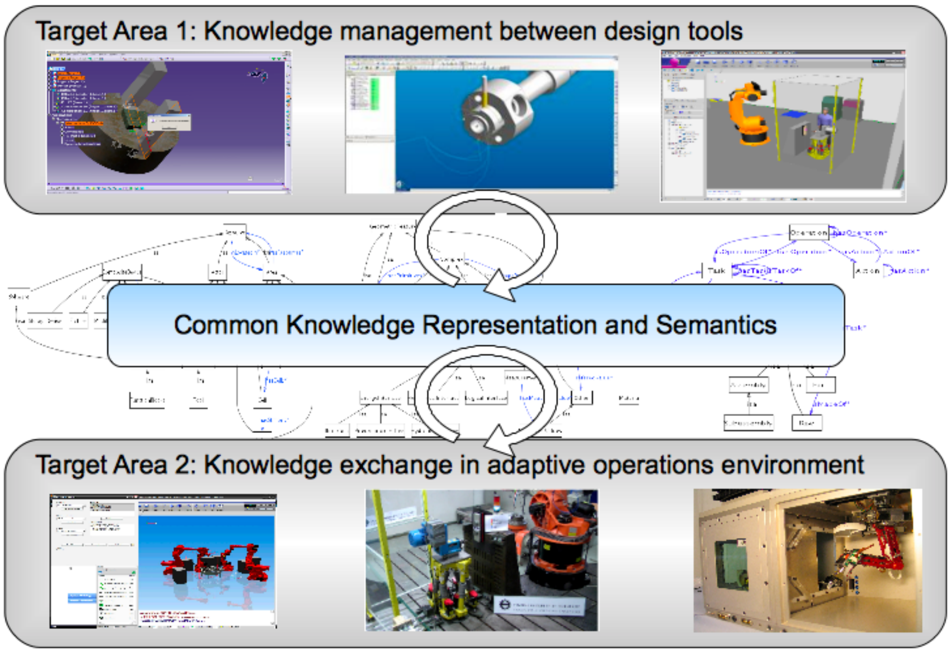

Target Area 1: Knowledge management between design tools
Common Knowledge Representation and Semantics
Target Area 2: Knowledge exchange in adaptive operations environment

Figure 2.1: Definition of the two main problem areas that share common needs

to support the design language of the engineers. The KR does not provide reasoning on the basis of the content. The surrounding system, be it the design environment or adaptive production system, will focus on the reasoning at different levels of abstraction, while the KR will remain neutral for these reasoning procedures. This allows the KR to be used as a common interface between different systems.

The intended knowledge representation must provide a semantic meaning for the terms 'product', 'process', and 'system'. In this thesis the term 'product' is defined as a representation of a physical object that has one or many parts assembled together and forms a stable set of part(s). The term 'feature' is any given geometric or non-geometric product, part, or resource property that has a meaning for design, assembly, or manufacturing purposes, and can be saved into a computerised system. The process is any activity related to the manufacture or assembly of the product. The term 'system' describes the set of resources needed to carry out the manufacturing or assembly processes. The term 'model', whether combined with the aforementioned terms or used as a single term, is a computerised representation of a product, process, or system. The thesis considers term 'knowledge' to be most complex form of content with context that can be saved into a computerised system.

The research objectives of the thesis are the following.

1. To develop and create a method for representing the product structure and the corresponding process representation.
2. To extend the representation into the product feature level and represent the processes that occur on the feature level.
3. To develop a model for representing product-, process-, and system-specific knowledge on the basis of the relations between the product level and the feature level.
4. To formalise the model thus developed into the form of a knowledge representation that allows knowledge inference to be applied.

2.2 Hypothesis

The proposed hypothesis is as follows.

The integration of the product, process, and system domains using a formal knowledge representation increases the knowledge content of a model by including the contextual description of the environment and its temporal aspects, as well as allowing inference to be applied on the model across the traditionally separated domains.

Formal knowledge representation such as ontology will structure the product, process, and system knowledge in such a way that inference with the captured knowledge becomes feasible.The use and re-use of the acquired knowledge will become more efficient as a result of the reduced number of re-design steps and unnecessary data conversions needed. The model's validity is increased by reducing the manual data transfer between tools and this enables the model to preserve its validity.

2.3 Research Methodology

The thesis starts with the assumption of the existence of relationships between the three design domains: the product, process, and system design domains as Rampersad [54] has proposed. The characteristics of the product pre-describe the set of processes needed to manufacture and/or assemble the product. The description of the product defines the constraints for the suitable processes. The processes pre-define the system requirements and constrain the set of systems capable of carrying out the processes that are needed. The cross-domain integration is deepened with the inclusion of semantic feature-based modelling methods adopted from [3, 18, 69] and theory of assembly-features from van Holland [28].

The research strategy starts with addressing the research problem. The second step is to summarise the relevant background material from the research efforts made in past. Since the scope of the thesis lies in the between of product design domain's and process planning domain's interfaces, and it aims to solve a knowledge management problem, the literature review is knitted tightly around corresponding fields. The main focus is on the formulation of the knowledge representation and in order to prevent the unnecessary expansion of the thesis the most important and most relevant references are collected. However, it is recognised that many other approaches exist, which are not discussed in this thesis.

The methodology for creating the knowledge representation starts from the definition of connectivity between the domains. The developed Knowledge Representation Formulation (KRF) methodology consist of five steps.

- Step 1 - Establishing Correct Requirements: this step aims to set out the requirements for the knowledge representation in the chosen domain.

- Step 2 - Definition of Connection Between Domains: the second step analyses the connectivity between the domains and proposes appropriate relations.

- Step 3 - Formulation Conceptual Model: the third step will gather the connectivities proposed in Step 2 in one conceptual model.

- Step 4 - Creation of a Knowledge Representation: in the fourth step the conceptual model is formalised.

- Step 5 - Evaluation of the KR on the basis of the Set Requirements: the fifth step evaluates the soundness and expressiveness of the knowledge representation that has been developed according to the requirements set.

Since no suitable method exists for creating connections between product and process models or feature and process models, two methods were developed. For defining the connectivity between products and the corresponding processes, a graph-based method, the Assembly State Decomposition Model (ASDM), was established. For representing feature-level information and the corresponding processes, a Manufacturing State Decomposition Model (MSDM) was created. To represent detailed product information, a feature-based modelling approach was used to add deeper meaning to product models. The third step of the knowledge representation formulation methodology utilised the Unified Modelling Language (UML) to define the connections between domains. The fourth step was to generate an ontology in the Web Ontology Language based on the Description Logics (OWL DL) to serve as a knowledge representation, and the final step was to evaluate the knowledge representation thus formed against the set requirements.

Other ongoing research projects at Tampere University of Technology (TUT) have provided a feasible platform to test and evaluate the knowledge representation that was created. In order to implement the KR a Knowledge Base (KB) was formed. The KB environment, along with the KR, is utilised in three different case studies. From the perspective of the thesis the goal is to evaluate the expressiveness and soundness of the knowledge representation that was formed. The first case study evaluates the inclusion of features and the connection to the processes and system. The second case study tests the role of the KR as a standard knowledge interface between several commercial and academy-built design tools. The final case study evaluates the feasibility of the KR in a live production environment, where the KR provides the required information on products and processes for an adaptive holonic manufacturing system and saves the operational values of the product that are realised as the product history. The conclusion will provide a summary of the results of the used methodology and the case studies.

2.4 Limitations

As the focus of this thesis touches on multiple different research areas, some constraints need to be expressed. First, the thesis represents the KR as a feasible solution that will meet the requirements in the three cases. The aim of this thesis is not to optimise the knowledge representation, but to make it work. Second, the thesis does not take into account the actual product design phases or the production control side. Third, the aim of this thesis is not to create a new standard or an extension to old ones for representing product, process, or system information or a combination of those.

The focus is on demonstrating the existence of relationships between these domains and providing a formal KR for that. The fourth note to be made is that the literature review will only summarise the past research activities relating directly to the solution presented here and to the similar solutions that other researchers have proposed. It should be noted that multiple other implementations and methods exist, but those will surely be discussed in other theses.

Most of the knowledge representations in the manufacturing domain focus on specific areas, such as product knowledge representations, process descriptions, or hardware-level knowledge representations. The research work done in this thesis adds input to the holistic approach, while most of the previous approaches are combined and developed further to fit the needs of the problem areas, such as production control and process planning. Naturally, this approach does not intend to solve all of the problems that arise while dealing with complex systems, but to provide solid foundations for future developments.

Chapter 3

Structure of the thesis

The structure of the thesis is as follows. The literature review presented in Chapter 4 covers the main research efforts in the fields of manufacturing knowledge representation, formal models, and knowledge-based approaches, which all play an important role in the formulation of the Knowledge Representation. As the starting point of the thesis lies in the interface between product design and the planning of manufacturing operations, the theory of product concept design itself is left aside.

Chapter 5, Proposed Methodology, introduces the Knowledge Representation Formulation (KRF) methodology for generating and evaluating a suitable cross-domain knowledge representation. Chapter 5 also introduces intermediate methods for creating relationships between two domains on different levels of abstraction. The Assembly State Decomposition Model (ASDM), Manufacturing State Decomposition Model (MSDM), and the Product-Process-System (PPS) model are introduced and, finally, the knowledge representation, the Core Ontology, is formed and evaluated.

Chapter 6 - Implementation - is dedicated to a brief description of the implementation of a Knowledge Base (KB) environment. The implementation of the KB is not part of the contribution of the thesis, but it provides a feasible test environment for the knowledge representation.

Chapter 7 will introduce the proof of concept in the form of three separate case studies that all utilise the same knowledge representation and implementation. These cases have been demonstrated during national and international projects. The cases are introduced from the knowledge representation perspective and aim to test the KR that was formed, the Core Ontology, from different points of view.

The conclusion and evaluation of the research done for this thesis is given in Chapter 8 - Conclusions. Chapter 9, Contributions, will summarise the contributions to science and research projects that this thesis has to offer. Chapter 10, Future Work, will outline new implementation ideas and plans where the Core Ontology could be used. Additionally, the future work will discuss some extensions to the Core Ontology that might be implemented later on.

Chapter 4

Generic Literature Review

4.1 Introduction to Literature

The structure of the literature review is formed in such a manner that in the first part the product, process, and system domain[1] integration method is introduced. The literature offers much unstructured information concerning the separate product, process, and system domains, referred to as design domains later on in the text, but the connection between these domains has hardly been discussed. For this reason there is insufficient insight concerning the relations between the various design domains and levels in each domain.

According to Rampersad [54], a solid understanding of the interaction between product, process, and system design is of prime importance in the analysis and design of manufacturing and assembly systems, as well as product development and process planning. This integration method, defined by Rampersad [54], outlines the basic connections between the domains from the assembly system design perspective. The integration method is based on an idea proposed by Rampersad [54] and extended by Lohse [45]. The method in itself does not define the connections below the product/part level to processes or systems.

The second part of the literature review introduces the beyond-the-geometry information modelling. The methods for adding the geometric and non-geometric information to the product and process models can take place with feature-based modelling and analysis. Several existing theorems will be

[1]The product answers the question of what, the process aims to answer the question of how, and the system domain defines where and by whom.

briefly introduced in this part of the literature review.

The third part of the literature review will introduce a selection of graph-based assembly process planning theories and process and resource description models. None of these models or theories alone provides the holistic knowledge representation that is needed to answer the problem defined in this thesis, but they provide a solid background from their areas.

The final part of the literature review will introduce domain-neutral knowledge modelling and reasoning methods, which provide a partly theoretical and partly technical background for solving the technical problem of this thesis. This part of the literature review briefly introduces the languages and technologies used in the formulation of the KR. In the last part of the literature review a few examples of existing knowledge representations, technologies, and objectives are introduced.

4.2 Overview of the Connectivity between Product Models, Process Structure, and System Requirements

The literature in the assembly-related domain is often ambiguous, since the word *'assembly'* is used to describe the static product, as well as the assembly process. To avoid any misinterpretation of the term, an assembly is *"a static combination of parts forming a (sub)product"*. The activity of combining the parts together is referred to as an assembly process, also defined by literature references as the *"putting together of components to make a product"* [45]. A system is *"a group of interacting elements forming a complex whole or entity"*.

According to Lohse's definition [45], an assembly system can therefore be defined as a group of interacting elements composed to put together components in order to form a product. From this definition it is possible to derive that an assembly system involves three distinct aspects: the product being assembled, the process of assembling the product, and the actual physical system that carries out that process.

In his research Rampersad [54] divided the overall assembly system requirements into three categories; *product*, *assembly process*, and *assembly system*,

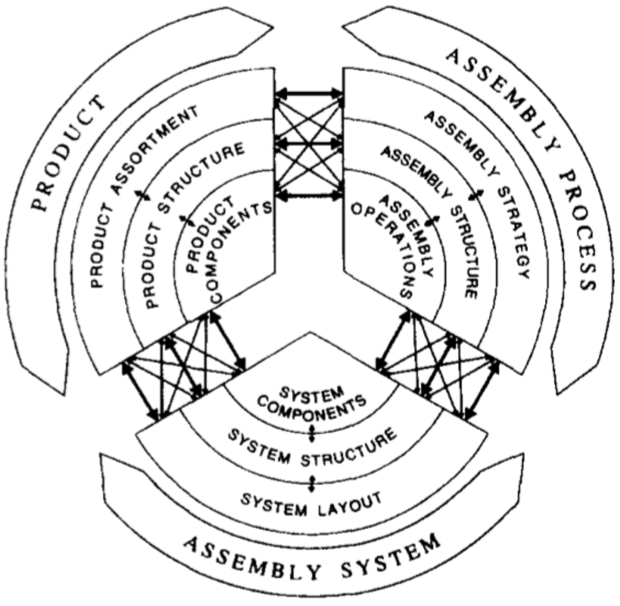


Figure 4.1: Integrated Assembly Model from [54]

and formed the Integrated Assembly Model. Each consists of three elements, which are set into three levels of abstraction (i.e. levels of complexity). The levels are illustrated in Figure 4.1. This classification corresponds to the steps in the system design process.

The classification corresponds to the steps that are taken during the design of an assembly system. The relationships between the variables are indicated by means of arrows. The thick arrows designate a stronger relationship than the thin arrows. As is apparent from the figure, the interactions between the variables on the same level of abstraction are the strongest, as well as those between the various elements per variable [54].

According to Rampersad [54], a product family entails *a collection of product variants which show far-reaching similarities in their characteristics.* As a result of this, the present generation of automated/semi-automated assembly systems is, in general, not suited to the assembly of strongly differing product families.

In his Integrated Assembly Model the product variable consisted of the three elements:

- *Product Assortment*: all the product variants to be assembled

- *Product Structure*: classification of the product sub-assemblies and components, as well as the representation of the relationships between them; the product structure defines the relationships between the parts
- *Product Components*: parts of a (sub)assembly or a product.

The Assembly Process section of the model contains the following:

- *Assembly Strategy*: the high-level choices made from alternative methods in order to increase the controllability of the assembly process.
- *Assembly Structure*: the sequence of the individual assembly operations and the relationships between them, aimed at bringing together product parts as composite units.
- *Assembly Operations*: a collection of individual assembly operations, including feeding, handling, composing, checking, adjusting, and special processes. Each of these operations is subdivided into sub-operations [54].

The third section of the model, Assembly System, contains the following layers:

- *System Layout*: the arranged positioning of concrete system components in the space within the assembly system: the location of the components and the relationships between them are determined in detail for this purpose. The system layout results from the system structure.
- *System Structure*: the collection of system components that are mutually related to each other. The location of the system components is determined globally for this purpose.
- *System Components*: the combination of the sub-systems of the assembly system that fulfil functions needed in the system [54].

Rampersad [54] concentrated on the theoretical model to show the connectivity among these three domains. Lohse [45] continued developing the integrated assembly model by concentrating on the connectivity of assembly processes and available assembly systems. However, the connection between products and processes is still in the stage of theoretical development and product knowledge is not used as a basis for assembly process or systems reasoning.

4.3 Product Models

The following subchapter will introduce selected ways to model the elementary relations inside the product models by utilising a feature-based modelling method to represent the models. They will form the theoretical foundation for the required KR.

4.3.1 Feature-based Modelling for Definition of Elementary Product Information

The feature-based model was developed in order to fill the gap between detailed geometry information expressed in CAD files, the elementary relations, expressed in engineering Bill of Material (eBOM)s and manufacturing Bill of Material (mBOM)s, and abstract functional information (other design documentation). Features include both the geometric and functional characteristics of the product data. By including features into the product model, the model can be represented on a higher level of abstraction than just a pure geometric model [18, 28].

The CAD, CAM, and CAx systems usually lack a complete specification of feature semantics or they have ill-defined semantics. Consequently, those options do not link functions to the final geometry in a proper way. Design systems do not utilise or save the necessary information on how certain geometric solutions were used. There are extensions available for CAD and CAM systems that do store information about features into the product model. But they fail to adequately maintain the meaning of the features throughout the modelling process or fail to transport the feature information to the downstream to process or system design software [3, 18, 69].

Bidarra and Bronsvoort [3] introduced a semantic feature-based modelling and defined this approach as being a declarative feature modelling approach that has characteristics such as:

- each feature must possess a well-defined meaning or semantics;
- semantics are classified in special feature classes, which are structured descriptions of all the properties of a given feature, defining a certain template for all its instances;
- users can define their own feature classes, e.g. by inheriting them from

an existing feature class and adding the desired constraints to them, and

- the whole modelling process is uniformly carried out in terms of features and their entities, as well as constraints.

In semantic feature modelling, all the properties of features, such as geometric parameters, values, and validity conditions, are declared by constraints. The main advantage is the freedom in the type of constraints that can be specified and therefore edited and maintained with ease. In addition, the use of various constraints for validity conditions in generic feature classes allows the specification of many semantic aspects for the instances of each class [3].

As a summary, the assembly features are defined in four different ways:

- elementary relations between components;
- elementary relations between components extended with some assembly information;
- a collection of elementary relations and matching form features, and
- an association between two form features present in different parts.

There are several definitions of assembly features, ranging from definitions of mating conditions to characteristics and requirements defining assembly processes. DeFazio [21] defined an assembly feature as any geometric or non-geometric attribute of a discrete part relating to mating conditions and whose presence or dimensions are relevant to the function, manufacture, engineering analysis, and use of the product or part. Sondhi and Turner [62] stated that the assembly features can be used for the specification of relationships on a higher level of abstraction. They proposed that assembly features as a higher-level interface could capture assembly relationships at the functional level, thus removing from the designer the burden of identifying the underlying elementary relationships. Shah [61] and his colleagues defined an assembly feature as an association between two form features in different parts. Deneux [18] continued by defining an assembly feature as a generic solution referring to two groups of parts that need to be related by a relationship so as to solve a design problem.

Sung [19] introduced the notion of assembly features that are composed of three adjacency relationships: Contact Adjacency, Internal Spatial Adjacency, and External Spatial Adjacency. Contact Adjacency is similar to the connection features used in the research done by Bronsvoort and Van Holland [28]. Internal Spatial Adjacency is similar to the handling features used by Bronsvoort and Van Holland [28, 29]. External Spatial Adjacency shows spatially opposing faces separated by empty space [19, 28, 29].

4.3.2 Product models for data exchange

Once the modelling of features became a standard characteristic of modern CAD systems, a new need arose. In order to share the features found between different modelling tools, several standard representations were formed. One of the most common formats that promised to share feature-level information was STEP (Standard for Exchange of Product data), which is the International Organisation for Standardisation (ISO) standard aimed at neutral product data exchange.

The implementable data specification of STEP is represented by Application Protocols (APs). The most widely-used APs are AP203 and AP214, for exchanging CAD files, and AP239, for product life-cycle support (PLCS). Originally STEP was developed in EXPRESS as a network of concepts. According to Krima et. al. [35], since EXPRESS is not based on formal semantics, the quality checking of these models is difficult. In order to overcome the constraints of the old STEP format and EXPRESS, an OWL-DL-based representation of STEP, ontoSTEP, was developed.

During the years different organisations saw that the core of STEP was not enough for their purposes and soon several different extensions were introduced. Unfortunately this led to a situation where the extensions were not embraced by the CAD system vendors, and thus only shallow product data could be transferred between different systems. ontoSTEP naturally cannot include more information than STEP models, but it can translate those into a more open format to be shared among different users.

Several new techniques exist for capturing *"beyond geometry"*-level product information. Those new developments include the Core Product Model (CPM), illustrated in Figure 4.2, designed at the National Institute of Standards and Technology (NIST). The CPM aims to provide a basic product model that is not tied to any vendor software and is open and non-proprietary,

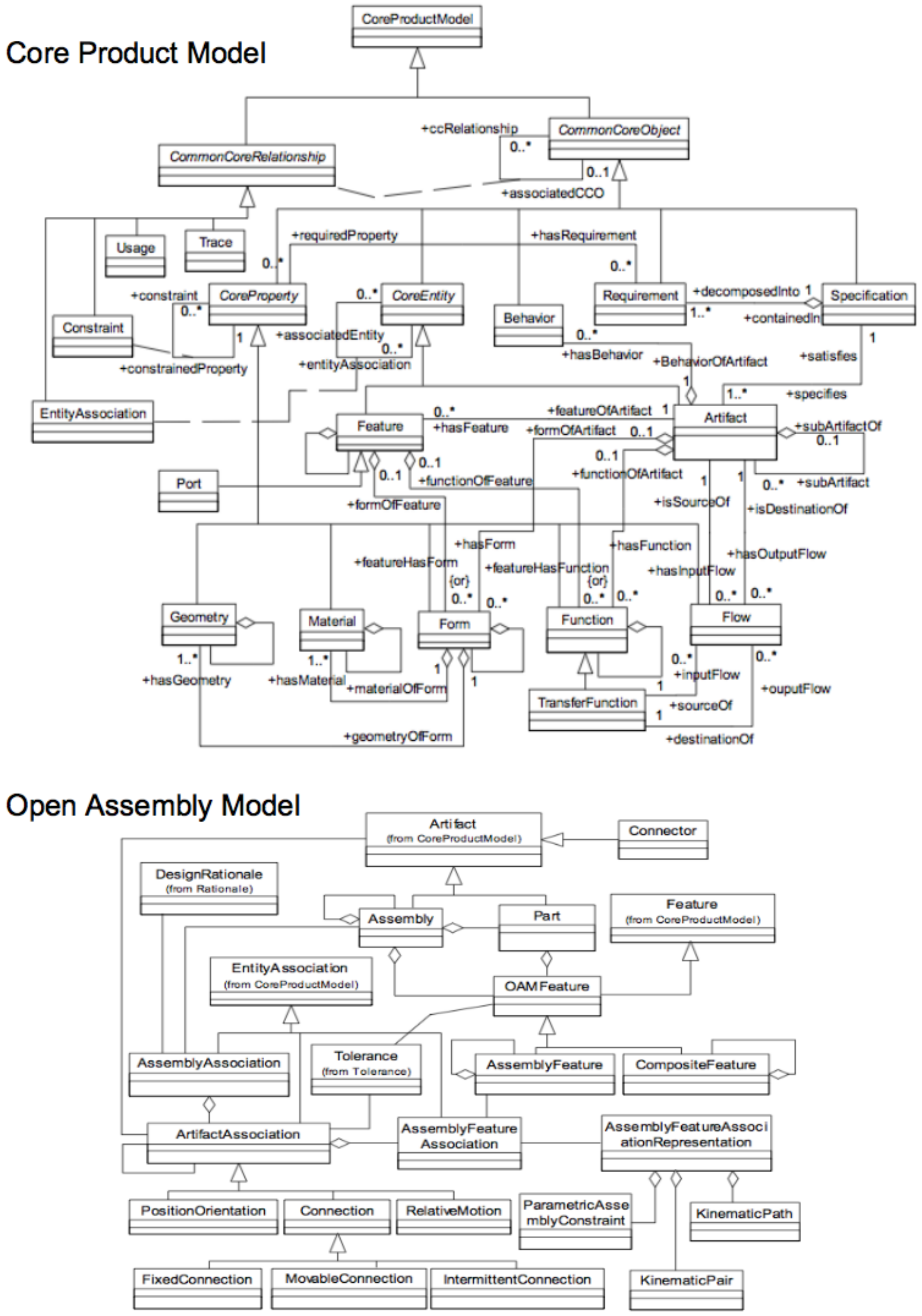


Figure 4.2: UML diagram of the CPM and OAM, modified from [53]

generic, independent of any product development process, and capable of capturing the engineering context that is most commonly shared in product development activities [22, 53].

While the CPM concentrated on single products, the Open Assembly Model (OAM) introduced the function, form, and behaviour of the assembly and defines both a system-level conceptual model and the associated hierarchical relationships. The main idea of the OAM is to provide a standard representation and exchange protocol for assembly and system-level tolerance information. Figure 4.2 shows the main schema of the OAM below the description of CPM. For the data structure representation, the OAM uses data structures adopted from STEP [35, 53]. Figure 4.2 shows the structure of the OAM [22, 53].

The OAM, together with the definitions of the CPM, gives a neutral yet detailed combination to represent product-specific knowledge defined on the feature level. However, the models lack process-related information, which could be tied to the product features.

4.3.3 Assembly Process Requirements through Assembly Features

Before production can begin, the product must first enter the engineering phase, which is a stage between modelling and producing the product. In this phase, the production and assembly process plans are described. The key for automated assembly process planning lies in the use of product model information for assembly analysis and assembly process planning.

In the design and modelling of the assembly process, features can also be seen as information carriers for assembly-specific information. This assembly-specific information includes, for example, degrees of freedom, interfaces between parts, subassemblies, and assemblies, and fit information. Assembly-specific feature information focuses on the relations between the components [28]. Feature-based assembly follows on naturally from feature-based modelling and can be used to add assembly-specific details to the models.

Assembly features, as discussed here, were originally only used to make the modelling effort easier. The information stored in the models was not used for any deeper analysis of manufacturing or assembly process planning. All

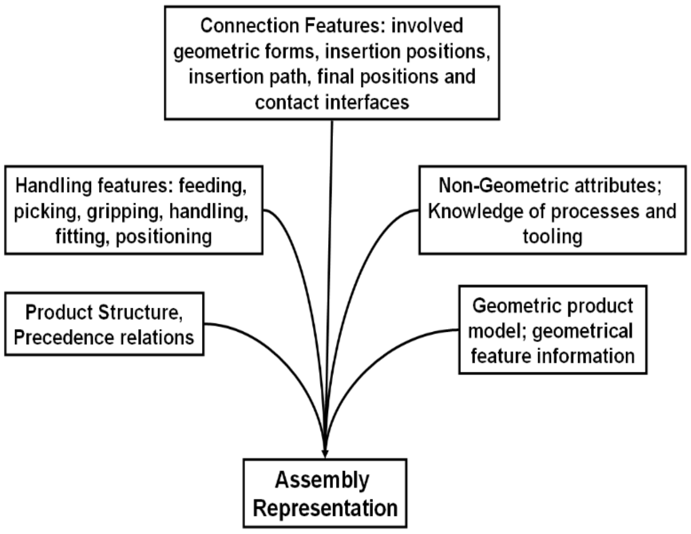

Connection Features: involved geometric forms, insertion positions, insertion path, final positions and contact interfaces
Handling features: feeding, picking, gripping, handling, fitting, positioning
Non-Geometric attributes; Knowledge of processes and tooling
Product Structure, Precedence relations
Geometric product model; geometrical feature information
Assembly Representation

Figure 4.3: Assembly process-related features [42, 65]

of these definitions focus mainly on the relationships between two or more components. Understanding the relationships between components is essential, but is not enough for assembly process representation [28, 69].

As mentioned by several authors [28, 42, 65], product knowledge is the combination of product-specific information, such as functionality, colour, and product variants, and the corresponding product model. This knowledge includes geometric and non-geometric information. In the design and modelling of manufacturing processes, features can be seen as the foundation elements, which can be used for the analysis and knowledge acquisition of the product.

Focusing on, for example, the assembly, the geometric features include all the geometric information of the product. Geometric features are, for example, pads, pockets, holes, chamfers, and rounding. Non-geometric information is, for example, tolerances, material, density, and surface roughness [28]. Figure 4.3 summarises the information required for assembly representation and emphasises the connection between the product information and process requirements.

Assembly features are made during design and manufacturing, so they are, or correspond to, manufacturing features; however, not all manufacturing features become assembly features. Furthermore, assembly features carry different design intent and information in their object data and methods [28,29,61].

Bronsvoort [29] and van Holland [28] defined assembly process features as *features with significance for assembly processes* and stated that they are subdivided into connection features and handling features. van Holland stated that the assembly-process features link information about the assembly process itself and the geometry, that is, both abstract and detailed geometric information together. Therefore, they are used to fill the gap between abstract and geometric models.

4.4 Process and System Models

The following subchapter will introduce different methods to tackle process information and process flow descriptions. System models as proposed by Rampersad [54] do not exist, but there are a few models for describing the connection of resources to processes. The taxonomic representations of manufacturing systems are not included in this literature review, since those are too strongly related to a specific physical resource to be generalised.

The assembly sequence is the most basic requirement of the assembly plan for the product. Traditionally it has played a key role in determining the important characteristics of the assembly tasks and of the finished product. In general, the process of generating the optimal assembly sequences are illustrated as a graph [72].

The theory of assembly sequence planning was formed at the beginning of the 1980s. Many of the early assembly reasoning systems were interactive ones, querying the user for geometric reasoning information and generating assembly sequences from the answers. Bourjault (1984) launched the field of algorithmic determination of all feasible assembly sequences for a rigid mechanical item [17, 36, 72].

As a continuation of Borjault's work, Lee [33] expanded the research field by introducing the mating conditions of components. Their methodology was divided into two steps: each component in an assembly is located at a specific vertex of the hierarchical tree, and an assembly procedure is generated from the hierarchical tree with the help of inference checking [17, 33, 36].

De Mello and Sanderson [17] developed their approach for analysing the assembly-sequence plan further via AND/OR graph representation. The AND/OR graph provides a compact representation of assembly plans and is equivalent to a directed graph of assembly states. The precedence relations that capture the domain-specific ordering constraints between connections and assembly states and connections were developed.

The connection-state precedence relations require some independence assumptions among assembly operations. They can be generated by enumerating state sequences using the AND/OR graph and simplified by using standard Boolean simplification routines. The connection-connection precedence relationships required more restrictive assumptions and could be generated more easily from the AND/OR graph, but they were more difficult to simplify. The precedence relationship provided an implicit representation of assembly sequences when they are used locally to test for a feasible next step in sequence generation, but the AND/OR graph is an explicit representation of complete and correct sequences.

Later, De Mello and Sanderson [17] introduced relational model graphs for defining assembly sequences. The principal focus in their research back in the '90s was the formalisation of assembly sequence plan representations, the proof of equivalence of different representations, and the implementation of algorithms to generate and transform them to plan representations. During the '90s a lot of research was conducted concerning the development of heuristic assembly sequence planning methodologies and planners. Yokota and Brough added the hierarchical object representation into the field of assembly sequence planning [72].

Zhao and Masood [72] stated that the determination of feasible choices for the assembly sequence can be a difficult process, for two reasons. First, the number of valid sequences can be large, even for a small part count: and it can rise staggeringly with increasing part counts. Secondly, seemingly minor design changes can modify the available choices of assembly sequences drastically.

The assembly sequence planning provides the order all of the possible solutions for the sequence of operations. It is a reasoning step. For representing the results in a form of language there exists several options.

For modelling the optimal process steps and connection of processes and resources, several different representations have been developed. Most of these models are implemented as stand-alone tools or ad hoc implementations, though a few standard representations do exist. One of those standardised process models is Process Specification Language (PSL). PSL facilitates

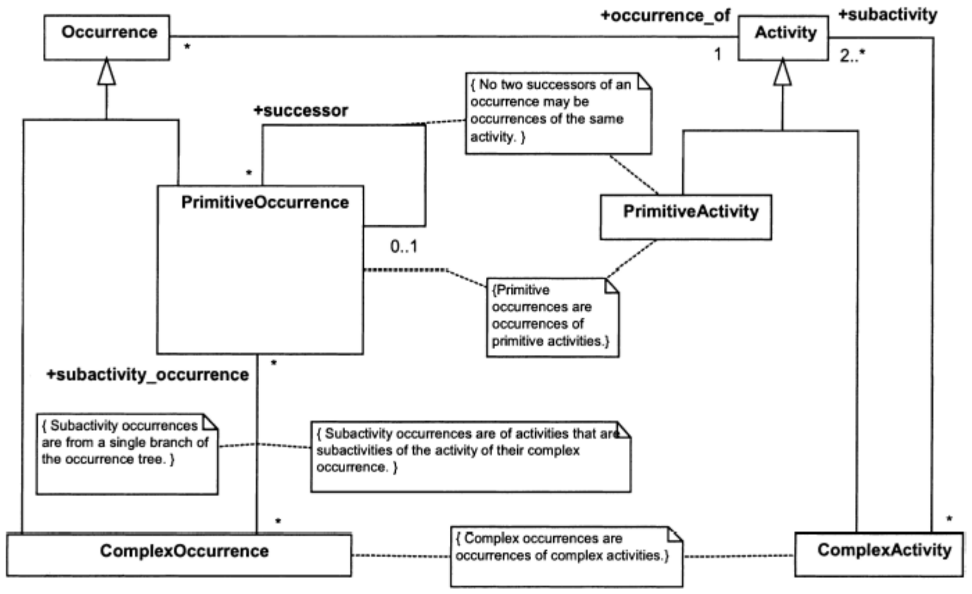

Occurrence
+occurrence_of
Activity
+subactivity
*
1
2..*
+successor
{ No two successors of an occurrence may be occurrences of the same activity. }
*
PrimitiveOccurrence
PrimitiveActivity
0..1
{Primitive occurrences are occurrences of primitive activities.}
+subactivity_occurrence
*
{ Subactivity occurrences are from a single branch of the occurrence tree. }
{ Subactivity occurrences are of activities that are subactivities of the activity of their complex occurrence. }
*
*
ComplexOccurrence
{ Complex occurrences are occurrences of complex activities.}
ComplexActivity

Figure 4.4: Basic concepts of PSL: Activity, Occurrence, and Successor [4]

a complete exchange of process information among manufacturing systems such as scheduling, process modelling, process planning, production planning, process simulation, project management, workflow, and business process reengineering [4, 27].

PSL consists of a core ontology of basic objects that exist in the domain, a partially ordered set of extensions that axiomatise additional primitive process concepts, and a multitude of definitional extensions that provide a rich terminology for describing process knowledge. The PSL ontology in Figure 4.4 is a set of theories used by first-order logic queries [4, 27].

In addition to PSL, another generic process and system representation method, Core Manufacturing Simulations Data (CMSD), was also developed. CMSD is intended to be a transfer language for conveying data from different design systems in a simulation environment. It describes the entities in the manufacturing domain on a high level and the relationships between them that are necessary to create manufacturing simulations. CMSD facilitates the exchange of information between simulation and other manufacturing software applications. The major categories in CMSD are organisation, calendar, resource, skill definition, setup definition, part, BOM, inventory, process plan, work schedule, revision, distribution definition, reference, and unit defaults

[32, 59].

The CMSD model is divided into two representations, one in UML and the other, which is identical, as a series of XML schemas. The UML version is intended for humans to understand the complex interrelationships and inheritance, while the XML one is a machine-interpretable format. The layout portion of the CMSD specification is not intended to be a new CAD format. CMSD operates on a higher level of process definition and simulation, where high-fidelity geometric representation of the manufacturing entities is not utilised [59].

4.5 Knowledge

The previous sub-chapters briefly introduced the field of product information, assembly process, and sequence planning. The information discussed was on a very concrete level and led to actual engineering solutions. However, in order to cover the necessary literature field the next chapter will introduce a more philosophical and IT-oriented view of the background needed for this thesis. The following subchapter will give a brief introduction to the world of knowledge formulation and reasoning that can be used for Computer-Aided decision making. At the beginning of this part a range of different knowledge-based systems (KBS) and higher-level paradigms using systems are introduced. Once the field is covered, the literature review continues with examples of knowledge modelling methods, languages used for formalising the knowledge representations, and concepts for reasoning the knowledge. The aim is to familiarise the reader with the different options of knowledge modelling which could have been used in solving the technical problem of this thesis.

It has been recognised for over 50 years that the information rate will grow rapidly without end, and yet our brains will remain in roughly the same state of development as they were when cavemen communicated with scowls and barks. Of course, technology has solved many problems regarding knowledge warehousing, but the real problem relating to the meaning of data and knowledge retrieval and reasoning unfortunately remains unsolved [14].

To make hidden knowledge accessible to a computer, knowledge-based and object-oriented systems are built around declarative languages whose form of expression is closer to human languages. Such systems help programmers and knowledge engineers reflect on *'the treasures contained in the knowledge'* and express it in a form that both humans and computers can understand [63].

According to Sowa [63] knowledge representation is a multidisciplinary subject that applies theories and techniques from three other fields: logic, ontology, and computation. *Without logic, a knowledge representation is vague, with no criteria for determining whether statements are redundant or contradictory. Without an ontology, the terms and symbols are badly defined, confused, and confusing. And without computable models, the logic and ontology cannot be implemented in computer programs.* Knowledge representation is the application of logic and ontology to the task of constructing computable models for some domain [63].

The definition proposed by Awad and Ghaziri [1] stated that knowledge can be divided into three levels: data, information, and knowledge. Data are defined as unstructured facts, which in IT terms are usually considered as just raw bits, bytes, values, or characters. Information is structured data and attributes which can be communicated, but which may only have their meaning locked inside proprietary software. Knowledge is seen as information that has meaning in more than only one type of software and can be used to achieve some results. Examples of these are an information model describing any computerised model or issues or 3D visualisation models, regardless of the information content of the model.

According to Davis [16], a knowledge representation is most fundamentally a surrogate, a substitute for the thing itself. The aim of a knowledge representation is to provide as good a mock-up of reality as is feasible. Depending on the language and inference method chosen, the surrogate is more or less accurate. Viewing representations as surrogates leads naturally to two important questions: what it is a surrogate for and how close the surrogate is to the real thing.

For knowledge representation to be accessible, surrounding technical solutions have been built. The indented knowledge is planned to be captured in knowledge bases rather than conventional databases. Knowledge-based systems utilise the knowledge saved in the system. The principal difference between a knowledge-based system (KBS) and a conventional program lies in their structure. In a conventional program, domain knowledge is intimately intertwined with software for controlling the application of that knowledge. In a knowledge-based system, the two roles are explicitly separated. In the simplest case there are two modules; the knowledge module is called the knowledge base and the control module is called the inference engine. In more complex systems, the inference engine itself may be a knowledge-based system containing meta-knowledge, i.e., knowledge of how to apply the domain knowledge [30].

Today the KBS systems are structured in such a manner that the problems of integrating the data are evident.

- Information archives are document-based. For a collective gathering of facts, a document-centric view is too coarse to be useful.
- Typical document management systems rely almost exclusively on information retrieval techniques that are inaccurate.
- Implications can only be made transparent if background knowledge is used, but systems today rarely support background, i.e. contextual, knowledge.
- Different people might contribute knowledge.
- Different people might require different views of the same basic piece of information [64].

A KB system that covers such knowledge about the outside world should:

- support the collective gathering of information on the level of facts rather than documents;
- integrate the gathering task smoothly into the common research process;
- allow one to combine facts intelligently;
- check new facts against the available background knowledge;
- allow multiple-view access to the knowledge through a single entry portal, and
- allow to route derived facts back into the common workplace environment [64].

KBs serve multiple roles. It can be said that they are repositories of shared knowledge. Interoperability and reuse of components and declarative knowledge are crucial to the further development of knowledge-based software. Unfortunately, it is hard to get components to interoperate and even harder to reuse other people's work. These difficulties are often a result of incompatibilities in the knowledge models, the precise definitions of declarative knowledge structures, assumed by the various components [25].

4.5.1 Knowledge Representations

Logic and Formal Structures

Even though the previous subchapters have concentrated on describing the content, for this thesis it is necessary also to look at model-independent reasoning. The technical verification of this thesis uses a Description Logic (DL) inference engine for processing the knowledge inside the Knowledge Base (KB) that is implemented. In order to provide a wider view of the possibilities, other reasoning methods and knowledge representation types are briefly described as well. It has to be noted that while reasoning relates to the scope of the thesis, the contribution is rather small, yet important, and in the form of technical requirements.

Logic-based Representation

Logic-based representation relies on sound mathematical foundations. Logical reasoning methods are generally divided into three classes: deduction, induction, and abduction. In a Peircean logic system, the logic of abduction and deduction contribute to our conceptual understanding of a phenomenon, while the logic of induction adds quantitative details to ones conceptual knowledge. Although Peirce justified the validity of induction as a self-corrective process, he asserted that neither induction nor deduction can help to unveil the internal structure of meaning [71].

According to Yu [71], during the stage of abduction, the goal is to explore the data, find a pattern, and suggest a plausible hypothesis; deduction involves refining the hypothesis on the basis of other plausible premises, and induction is the empirical substantiation. In other words abduction is more a form of critical thinking than formal logic, while the deductive reasoning moves from a general premise to a more specific conclusion, and inductive reasoning moves from specific premises to a general conclusion both based on formalism.

These methods of reasoning will produce different kinds of results. In a deductive argument with valid reasoning the conclusion contains no more information than is contained in the premises. Therefore, deductive reasoning does not increase one's knowledge base, and so is said to be non-ampliative. A logic allows the axiomatisation of the domain information and the drawing of conclusions from that information [2, 63].

Propositional Logic

The simplest form of logic is propositional logic, involving the manipulation of propositions. A proposition is a statement that can have two values: TRUE or FALSE. Propositional variables (and propositions) can be connected by logical operators or logical connectives. The commonly used logical connectives are: AND, OR, NOT, IMPLIES, and EQUIVALENCE. Another form of logic, predicate calculus, extends propositional logic in such a way that a wide range of real-world knowledge can be represented [2, 63].

Predicate Logic - First-Order Logics

The system of first-order logic (FOL) is the most widely studied today, because of its applicability to the foundations of mathematics and because of its desirable proof-theoretic properties. First-order logic is a predicate logic and is distinguished from propositional logic by its use of quantifiers; each interpretation of FOL includes a domain of discourse over which the quantifiers range. A predicate resembles a function that returns either True or False.

As an example: by considering the following sentences: *"Robot is a Device"*, *"Lathe is a Device"*. In propositional logic these are treated as two unrelated propositions, denoted, for example, by p and q. In first-order logic, however, the sentences can be expressed in a more parallel manner using the predicate Device(a), which asserts that the object represented by *"a"* is a device. Thus if *"a"* represents Robot then Device(a) asserts the first proposition, p; if a represents Lathe then Device(a) asserts the second proposition, q. A key aspect of first-order logic is visible here: the string *"Device"* is a syntactic entity which is given semantic meaning by declaring that Device(a) holds exactly when *a* is a device. An assignment of semantic meaning is called an interpretation. First-order logic allows reasoning about properties that are shared by many objects, through the use of variables [63].

Description Logics

Description Logics (DLs) is the most recent name for a family of knowledge-processing formalisms that represent the knowledge of an application domain by defining the relevant concepts of the domain (its terminology), and then using these concepts to specify the properties of objects and individuals occurring in the domain (the world description) [2].

Because Description Logics are a KR formalism, it is expected that a KR system always provides an answer in reasonable time and these procedures should return both positive and negative answers. The guarantee of an answer in finite time need not imply that the answer is given in reasonable time; investigating the computational complexity of a given DL with decidable inference problems is an important issue. The decidability and complexity of the inference problems depend on the expressive power of the DL at hand. On the one hand, very expressive DLs are likely to have inference problems of high complexity, or they may even be undecidable. On the other hand, very weak DLs (with efficient reasoning procedures) may not be sufficiently expressive to represent the important concepts of a given application [2].

DL-based inference engines are implementable in a wide range of applications and can provide reasoning with database schemas and queries. DL inference engines are perhaps best known as the basis for widely used ontology languages such as OIL (Ontology Inference Language), DAML (Darpa Agent Modelling Language)+OIL, and OWL (Web Ontology Language). In general, DL as a concept provides formal underpinning for these languages and also as an inference engine for computational services for ontology management tools and applications [67].

There are several methods to describe the knowledge for exchange and/or strictly for knowledge inference. These include Rule-based Representation, Semantic Networks, and Frame-based representations. According to Hopgood [30], in most cases the rules are shallow by nature, representing a shallow amount of knowledge, namely information or data. Rule-based representation has been used in various expert systems. But since the interest of this thesis is in the domain of knowledge representation, the focus is more on representing knowledge rather than going straight to inference with it.

Ontologies for Formalising Knowledge

The efforts to realise the semantic web have been accelerating the research on the development of ontologies, since ontologies are considered as catalysts for knowledge sharing and mediation in heterogenous environments. One of the first modern definitions of an ontology was given by Neches et. al. [47], who defined an ontology as follows: *"an ontology defines the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining terms and relations to define extensions to the vocabulary"*. This descriptive definition identifies basic terms and relations between terms, identifies rules to combine terms, and provides the definitions of such terms and relations. According to Neches's definition, an ontology includes not only the terms that are explicitly defined in it, but also the knowledge that can be inferred from it [46].

A few years later, Gruber [26] defined an ontology as *"an explicit specification of a conceptualisation"*. This definition became the most quoted in literature and by the ontology community. On the basis of Gruber's definition, many definitions of what an ontology is were proposed. Borst [6] modified Gruber's definition slightly : "ontologies are defined as a formal specification of a shared conceptualisation". Gruber's and Borst's definitions were merged and explained by Studer et. al. [64] as follows: *"Conceptualisation refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon."* Fernandez et al. [46] compared and divided the types of ontologies in the following manner:

- *Knowledge Representation Ontology*: the representation primitives for a given knowledge paradigm
- *General/Common Ontology*: a generic type of ontology, which can be used/re-used across different domains
- *Upper-level Ontology*: defines a set of concepts for a framework, which can be specialised for different domains
- *Domain Ontology*: a domain-specific ontology, where the domain is well-known by the community using it
- *Task Ontology*: is a specialisation of a upper-level ontology, but can still be used as a generic level across the domains
- *Domain Task Ontology*: defines a set of tasks in a chosen domain

- *Method Ontology*: provides a problem-solving or reasoning methods for achieving given tasks
- *Application Ontology*: defines an application specific ontology

Ontologies, in general, could be modelled utilising different knowledge modelling techniques and they could be implemented using several languages. Depending on the implementation language, ontologies are classified as:

- highly informal (expressed in natural language)
- semi-informal (expressed in a structured form of natural language)
- semi-formal (expressed in an artificial and formally defined language)
- rigorously formal (expressed with precise terms and formal semantics) [51]

4.5.2 Tools and Languages Sharing the Knowledge

The relationship between the knowledge modelling components (concepts, roles, etc.) and the knowledge representation (KR) techniques are used to formally include concepts of First-Order Logic (FOL), Description Logics (DL), Frames, and conceptual graphs and the languages utilised to construct ontologies within a given KR technique. In other words, an ontology can be built according to a specific KR technique and it could be implemented in several languages used by the specific KR technique [51].

KIF, GFP, and Ontolingua

Previously ontologies were mainly built using AI modelling techniques based on frames and first-order logic and expressed with modelling tools such as Ontolingua and Knowledge Interchange Format (KIF). A few years later the Generic Frame Protocol for manipulating knowledge expressed in an implicit representation formalism called the GFP Knowledge Model was developed. The GFP Knowledge Model supports an object-oriented representation of knowledge and provides a set of representational constructs commonly found in Frame Representation Systems (FRSs). In order to provide a precise and succinct description of the GFP Knowledge Model, the Knowledge Interchange Format (KIF) was used as a formal specification language [9, 51].

The original Ontolingua, as [26] defined it, was designed to support the design and specification of ontologies with a clear logical semantics. In order to accomplish this, Gruber extended the KIF definition sublanguage to provide additional idioms that occurs in ontologies and added a *Frame Ontology* to enable ontologies to be specified in a pseudo-object-oriented style using relations and functions such as *Class*, *Subclass-of*, *Slot*, *Slot-Value-Type*, *Slot-cardinality* and *Facet* [23]. The development done by Fikes et al. [23] in 1997, extended Ontolingua to allow its users to state explicit inclusion relationships between ontologies and implicitly created axioms.

Web Ontology Language

W3C standard for representing an ontology in 2004. OWL is intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans. An OWL ontology may include descriptions of classes, properties, and their instances. Given such an ontology, the formal semantics of OWL specify how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms. OWL is a revision of the DAML+OIL web ontology language, which is built upon the RDF(S). OWL ontologies are written either in XML or with the triples notation for RDF. OWL is divided into three layers: OWL Lite, OWL DL, and OWL Full [51, 70].

- OWL Lite is used to create only class taxonomies and simple constraints.
- OWL DL as already stated, provides maximum expressiveness, while ensuring that all valid conclusions can be inferred (computational completeness) and that inferences are deterministic.
- OWL Full includes the complete vocabulary, thus providing more flexibility. However, OWL Full provides no guarantee of computational completeness and determinism [70].

4.5.3 Knowledge-Based Systems

In the early '80s, medical expert system applications dominated the scene, primarily because of the diagnostic nature of these applications and the relative ease of developing such systems [44]. By the end of the '80s the problems the expert systems were being given had become more complex. It was also the time when the development of systems for commercial sectors began. Hopgood [30] concluded that an expert system is a knowledge-based system that acts as a specialist consultant; it is often proposed that an expert system must offer certain capabilities that mirror those of a human consultant. In particular, it is often claimed that an expert system must be capable of justifying its current line of inquiry and explaining its reasoning in arriving at a conclusion.

Unfortunately, the initial attempts frequently met with limited success. The reasons for the initial failures were, first, the fact that the early applications of expert systems over-challenged the technology, leading to poor results. The second reason for their failure was the sheer size and complexity of the problems they were expected to deal with; completing these in reasonable time was impossible. The limitations of such an expert system are easily exposed by presenting it with a situation outside its narrow area of expertise. When confronted with a set of data with no explicit rules, the system cannot respond or, worse still, may give wrong answers. Other important deficiency of inference of expert systems is the shalloweness of the reasoning procedure and results that can be derived from it. Because the knowledge bypasses the causal links between an observation and a deduction, the system has no understanding of its knowledge. For this reason the expert systems were mainly used for diagnosis, interpretation, and prescription [30, 44].

According to Hopgood [30], in achieving modest success research in the field of artificial intelligence, together with other branches of computer science, has resulted in the development of several useful computing tools. These tools or methods have a range of potential applications and those can be roughly divided into knowledge-based systems, computational intelligence, and hybrid systems. Caglyan [7] and Hopgood [30] divided knowledge-based systems into categories; expert and rule-based systems, object-oriented, and frame-based systems, and intelligent agents. Computational intelligence includes neural networks, genetic algorithms, and other optimisation algorithms. Techniques for handling uncertainty, such as fuzzy logic, fit into both categories. The following subchapters will outline different agent- and holon-based approaches used in the field of manufacturing for to reason with knowledge.

Agent-based Systems: ONTOMAS and inference

In his research Lohse [45] focused on the connection between assembly processes and modular assembly systems. The challenge in his work was to create a common environment where domain experts can collaborate effectively while taking advantage of the best practices of their diverse domains. The approach taken in this research started by taking advantage of the higher levels of standardisation inherent in the modular assembly system paradigm, which is considered to be one of the fundamental enabling factors in achieving a high level of adaptation. In order to address the adaptation by modularisation, a new ontology-based framework was developed. The ONTOMAS (Ontology for the design of Modular Assembly Systems) framework is based on engineering ontology principles structuring the domain using formalisms for aggregation, topology, taxonomies, and system theory principles.

Lohse [45] divided the domains according to Rampersad [54] into three domains: the product, process, and system domains. The product (and project) domain ontology is mainly focused on capturing the user requirements of an assembly system. The product definition stands at the core of the user requirements. The required relationships, so-called liaisons, between the component parts are of specific interest for the definition of an assembly system. The product model is created from the assembly system perspective and does not include feature-level information. The main focus of the research was the connection between processes and modular assembly systems. Since the domain was limited to the modular systems there was no need to utilise detailed product knowledge as a starting point.

The assembly process domain ontology defines the core of the assembly system requirements. The process model is used to define the temporal order in which the individual components of the product can be put together through activities and their temporal relationships. The processes were divided into three semantic layers: tasks, operations and actions. The assembly system domain defines the resource-related concepts that are used, or can be used, to facilitate assembly processes. The central concept is the equipment which can be connected to form system solutions. Fixed hierarchical levels were defined and incorporated into an equipment classification on the meta-level [45].

Lohse proposed a hierarchical taxonomy for providing a structure for required assembly equipment classification. The proposed assembly equipment taxonomy is based in the first instance on their functional capabilities and in the second instance on their implementation principle, which is expressed

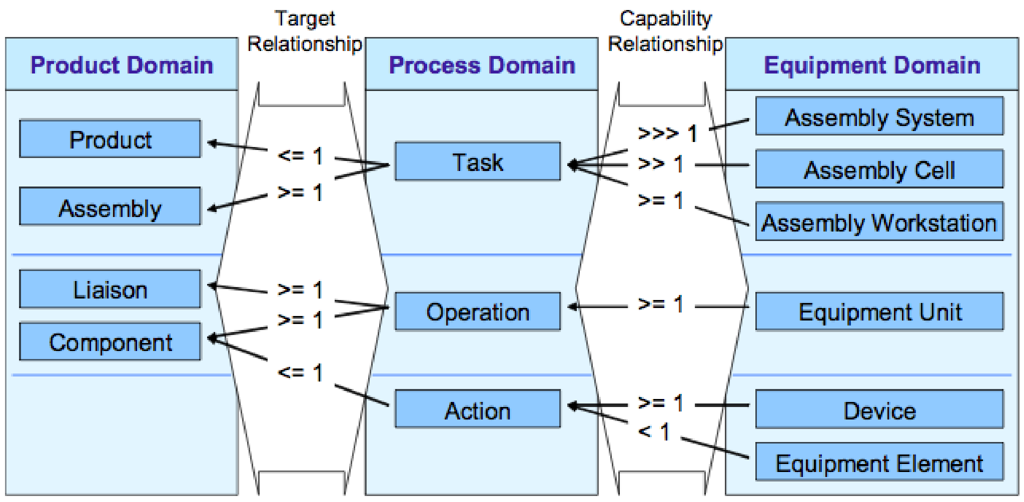


Figure 4.5: ONTOMAS product, process, and system domains [45]

through their behaviour. The equipment taxonomy was a high-level taxonomy that aimed more at structuring the domain in the general sense and not so much at providing very detailed classifications for the different equipment domains, which should be left to the domain experts in any case. The advantage with a taxonomic approach is that more specific domain concepts can always be included by extending existing more generic concepts.

For structuring the defined knowledge into three levels a CommonKADS methodology [52] was used. The three levels were Task Knowledge, Domain Knowledge, and Inference Knowledge. The Framework Programme 6 project EUPASS extended the knowledge levels from CommonKADS to fit to the theoretical integrated assembly model created by Rampersad [54] and developed further by Lohse [45].

- *The Task Knowledge* level defines the design tasks required for achieving specific design goals and the actors involved. Task knowledge is described on different levels of abstraction that define a hierarchy from general tasks to more specific tasks. The lowest level includes all the tasks which cannot be divided meaningfully into subtasks. Those are entirely built for members of the inference knowledge level. Each task has one or more task methods that define their sub-activities and in which order they need to be performed. Actors are then mapped into sufficient tasks [45, 52].
- *The Domain Knowledge* level defines all the concepts, relationships,

attributes, and rules which are used by the inference level activities. Concepts are defined as a set of attributes. Concepts are defined as a set of attributes. Relationships are either concepts in their own right or are defined through the attributes of other non-relationship concepts. Rules are defined through their antecedents, consequences and the causality between them. The constraints of the model are specified as first-order-logic axioms [45, 52].

- *The Inference Knowledge* level defines the inferences, decisions, and communication acts required for performing the design tasks on the task knowledge level. Inferences are defined through the type of their dynamic input and output knowledge and a set of rules that are used to infer the output from the input. Decisions are specialised inferences that have dynamic input knowledge and infer a yes-or-no decision on the basis of a set of static rules, defined by the decision criteria [45, 52].

Agent-based Systems: P2 Ontology and inference

It was stated in the Framework Programme 6 project PABADIS'PROMISE [51] that the increased number of combinations and the consequent complexity of the manufacturing system highlight the role of the prime importance of the communication between the different entities, such as agents, and creates the necessity of thoroughly structuring the information which will be exchanged. Accordingly, the information has to be structured in such a way that first, all agents can share the same understanding, and second, the data necessary for an agent to perform its mission can be tailored to fit its limited storage capacity.

The interoperable integration of enterprise-level systems with manufacturing-level systems is not an easy task, mainly because of the plethora of different systems, tools, terminologies, and business logics. The prevailing MES solutions are too industry-specific and process-centric. Standardisation seems to play a crucial role within this environment. Since a lot of data flow exists between the enterprise and the production systems and applications, considerable difficulties arise during their supervision and control, mainly as a result of origin and content ambiguity. If the production data is not valid, the performance of manufacturing systems becomes weak [51].

The approach taken in this project was to form the P2 model and P2 ontology to be formal and to provide unambiguous definition of all the components and of their interactions with each other in an enterprise/industrial environment in order to establish a common language for exchanging and describing all the complex information that is related to the lower levels of an industry. The PABADIS'PROMISE project resulted an ontology, the P2 Ontology, for describing manufacturing resources, processes and corresponding products. The P2 Ontology represents the knowledge of a domain in a formal way that can be machine-understandable (XML/RDF format).

In a way the PABADIS'PROMISE resulted in a reference architecture that utilised standards such as IEC/ISO 62264 with the IEC 61499 Function Block and the ISO 10303 STEP in conjunction with the P2 Ontology. This approach provided a generic solution to the vertical interoperability problem from the enterprise level to the manufacturing level.

The focus of the project was on factory control and production scheduling in a multi-agent environment. They identified similar challenges to those mentioned in the introduction to this work. However, in their approach the product data conversions were performed semi-automatically, while the production planning was agent-based.

Holonic Manufacturing System: ADACOR

In order to improve agility and flexibility, nowadays distributed approaches are used in developing manufacturing control applications. These are built upon autonomous and cooperative entities, such as those based on multiagent and holonic systems. A Holonic Manufacturing System (HMS) translates to the manufacturing world the concepts developed by Koestler [34] for living organisms and social organisations. The HMS paradigm addresses the agile reaction to disturbances at the shop floor level in volatile environments and it is built upon a set of autonomous and cooperative holons, each one being a representation of a manufacturing component, i.e., a physical resource such as a robot, CNC centre and conveyor, or a logical entity, such as orders.

In their research Borgo et. al. [5] defined and developed the foundations for a core ontology for manufacturing. The aim of their approach was to consider aspects of the manufacturing domain in order to build a core ontology for this domain that guarantees: (1) integration with a well-organised and accepted foundational ontology; (2) accessibility to agents in the manufacturing domain, and (3) suitability for product and process modelling, as well

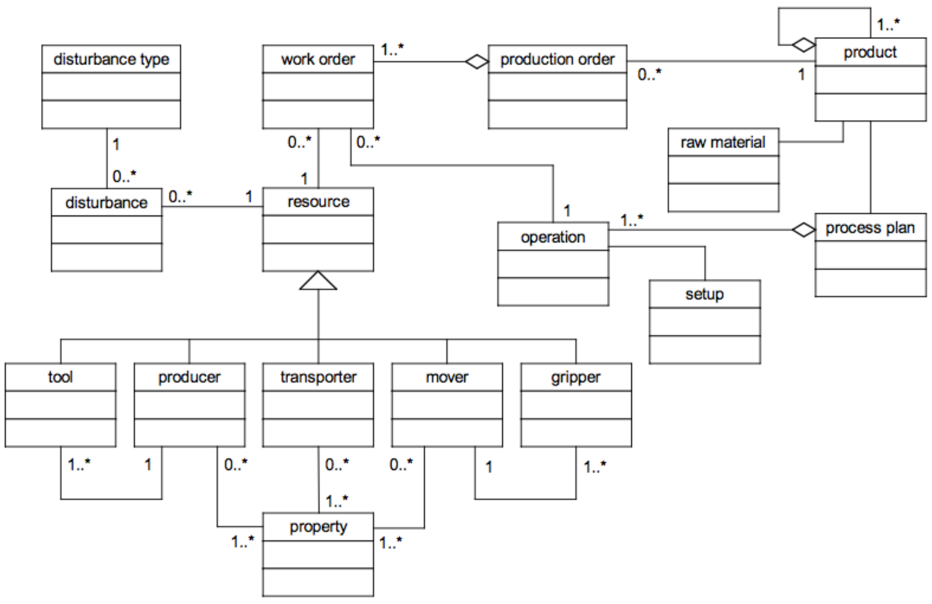


Figure 4.6: Manufacturing Ontology in the ADACOR Architecture [5]

as for information sharing, exchange, and retrieval.

According to Borgo [5], the adoption of an established foundation ontology, DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering), was used to improve the consistency of the overall system. Their research resulted in an ontology named ADACOR (ADAptive holonic COntrol aRchitecture for distributed manufacturing systems) for a multi-agent-based production paradigm. The ADACOR ontology defined its own proprietary manufacturing ontology, expressed in an object-oriented frame-based manner. The ontology includes the basic classes for representing processes, resources, and orders in a manufacturing environment; see Figure 4.6.

Holonic Manufacturing System - DiMS

As mentioned previously, manufacturing systems in use are under constant pressure to improve and re-organise. The need for improvement and change comes from the business, product development, and manufacturing domains and is generally related to new value offerings, capacity adjustments, cost efficiency, or technology. At the same time, innovative ideas and new concepts are emerging and new technologies are being developed. Fast adaptation to the opportunities arising from innovative business models, products and pro-

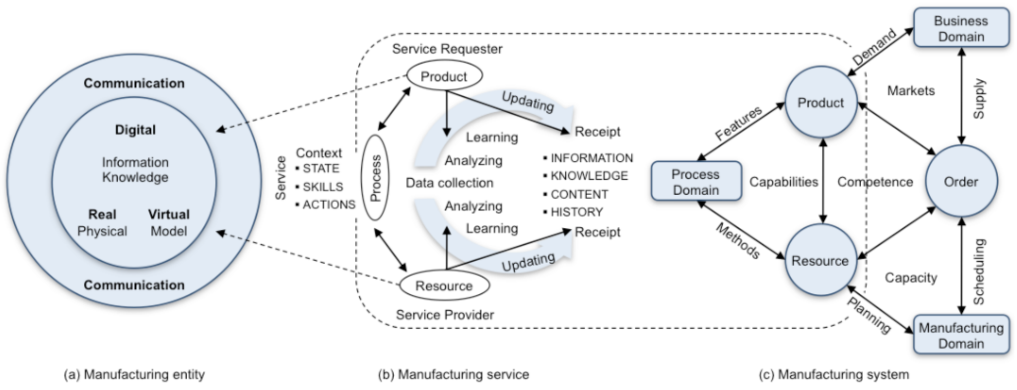


Figure 4.7: Structure of Manufacturing Entities, Services, and Systems, adapted from Lanz et. al, [40]

duction models, and technologies and their synergies is the key competitive issue.

Emerging new manufacturing paradigms such as Bionic Manufacturing Systems (BMS), Holonic Manufacturing Systems (HMS) and Fractal Manufacturing Systems (FrMS) have recognised the distributed and evolutionary nature of manufacturing processes within an industrial context [68]. Manufacturing systems can be seen from this viewpoint as emerging through an evolutionary synthesis process simultaneously with business and product models, rather than being planned according to a known set of parameters. Nylund and Salminen [48, 49] have developed a Distributed Manufacturing System concept (DiMS) as a synthesis of new ideas. DiMS is based on a holonic architecture.The core of self-configuring manufacturing system, such as DiMS, is in its internal service negotiation process between the Business, R&D, and Manufacturing domains, as illustrated on the right-hand-side in Figure 4.7.

As the concept operates on the basis of the information embedded in services, it is also vital to ensure the existence and quality of the services offered. As service holons are communicating entities in a constantly evolving self-organising system, their formal structure has to support emergent properties, i.e. to have internal change logic [48, 49].

In one hand the holons are representations of actual physical devices that have been enhanced with intelligent reasoning and communication interfaces, on the other hand the order holons are purely software agents setting things

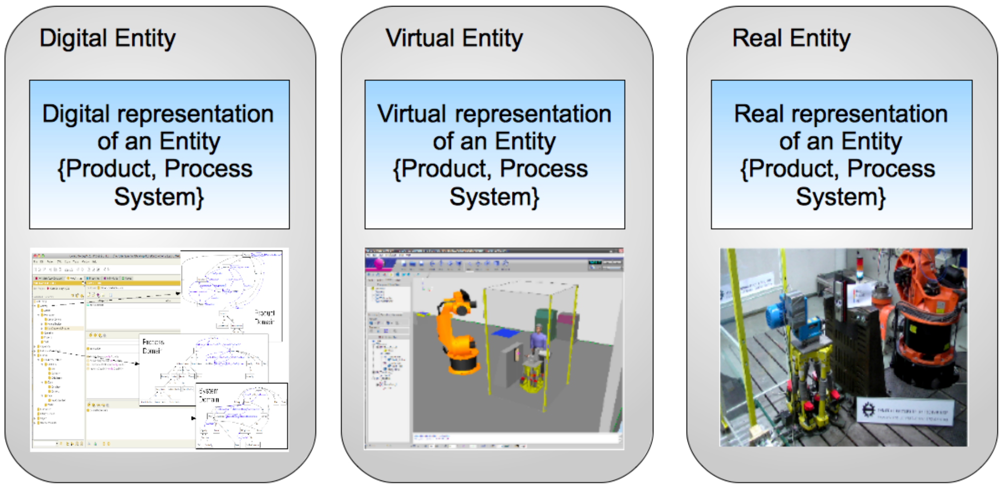

Digital Entity
Digital representation of an Entity {Product, Process System}
Virtual Entity
Virtual representation of an Entity {Product, Process System}
Real Entity
Real representation of an Entity {Product, Process System}

Figure 4.8: Knowledge representations of an entity, modified from Nylund [49] and Lanz [41]

into motion. An order consists of a fractal holarchy [2] representing sales and product information and equivalent test manufacturing model. Operative manufacturing takes place as order holons manufactures itself and aftermarket information remains as order holon gets history holon status [48, 49].

The DiMS concept aims to eliminate waste - of time, material, or resources - by validating the digital model in the simulation environment and treating it as a hypothesis, which will be proven true or false in the real manufacturing environment. The manufacturing plan itself is also considered to be in the state of hypothesis when the manufacturing context is changed. The hypothesis approach forces the manufacturing system to re-evaluate its performance and focus on continuous improvement, since the context is considered to be dynamic. In order to represent the product-process system entities' knowledge content, the model must be extendable for the digital representation of an entity and its simulation and a physical representation of it, as illustrated in Figure 4.8.

The *Digital Entity* is the representation of the digital information of the entity [48]. Figure 4.8 illustrates the different domains used to describe the contents and the context of the entity. The *Virtual Entity* is a category for the validation of the knowledge of the Digital Entity by different levels of

[2]Holarchy is a group of holons connected together in order to provide required capability or a combination of a capability.

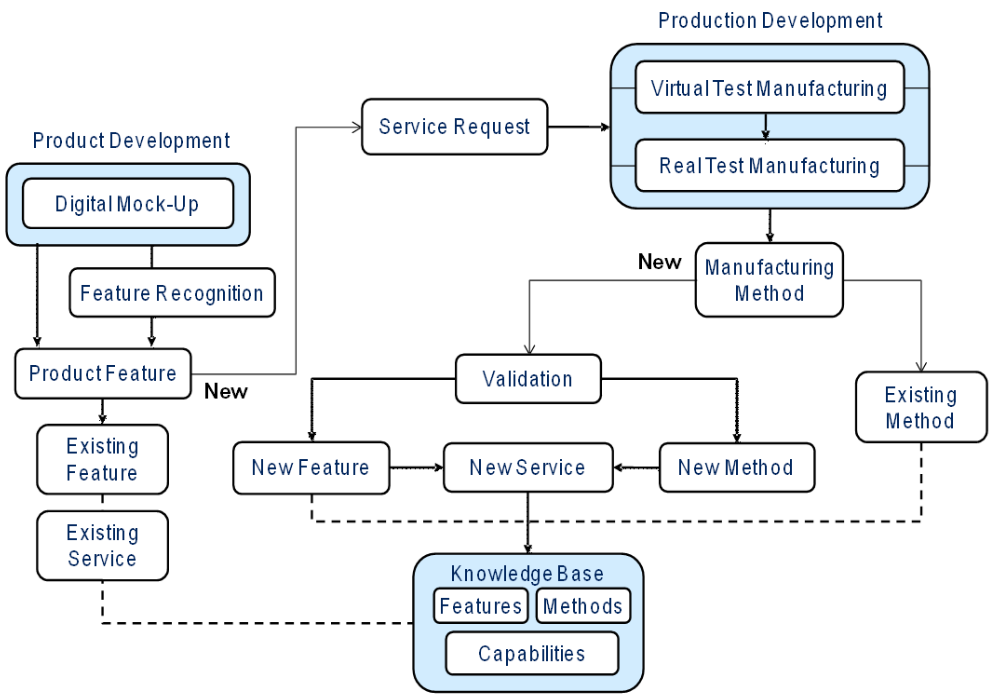


Figure 4.9: Connections and Relations in the DiMS framework, modified from [49]

simulations. The simulation is used, for example, for obtaining possible manufacturing scenarios, the verification of assembly movements, the reachability of the robots and devices that are used, and the validation of planned processing times. The best option, defined by the user, is sent back to KB as a validated process and with the resource information of the digital model. The simulation model is connected to the Digital Entity via references. Figure 4.9 illustrates the connections and relationships inside the DiMS framework.

Summary of the Literature Review

The structure of the literature review is formed in such a manner that a product, process, and system domain integration method is introduced first. The integration method is based on the idea proposed by [54] and implemented with modifications by Lohse [45]. The method in itself does not define the below-product/part-level connections to processes or systems. For representing these there exist the feature-based modelling and analysis methods from van der Net [69] and van Holland [28], which were introduced in the second part of the literature review.

Third part of the literature review introduced graph-based assembly process planning theories from Zhao [72] and Sandersson and de Mello [17], and process and resource description models as explained by Gruninger [27] and Bock [4] and Riddick [59].

The fourth part of the literature review gave an introduction to the formulation of knowledge representation and logics. Ontological modelling was also described briefly, and finally, knowledge-based production paradigms - expert systems, agent-based systems [45, 51], and holon-based [5, 49] manufacturing paradigms and their implementations focusing on the knowledge representations used - were introduced.

As can be seen from the literature review, most of the solutions that have been used in various domains have only been implemented in somewhat narrow areas. For example, formal knowledge representations or standards representing connections between the product, process, and system domains do not exist. The KRs represented in the final part of the literature review all focus on describing the manufacturing domain, i.e. how the processes can be linked to the resources. However, the input knowledge is missing. The characteristics of a product impose requirements on the overall system capabilities that fulfil the process needs. Currently no model exists that could combine product characteristics with the capabilities of a system component.

Chapter 5

Proposed Methodology

Chapter Five will describe the core of the thesis. In this chapter the initial problem is formulated into a business problem and in order to propose a solution the business problem is formulated into a technical problem. Figure 1.2 illustrated the current situation, with a distributed design, process planning, and production scheme with highly informal knowledge exchange. The thesis proposes a methodology and a model to solve the technical problem concerning the product, process, and system information modelling, integration, and exchange. The result of the methodology is the knowledge representation. The steps for the Knowledge Representation Formulation (KRF) Methodology are the following:

- Step 1 - Establishing Correct Requirements
- Step 2 - Definition of Connection Between Domains
- Step 3 - Formulation of a Conceptual Model
- Step 4 - Creation of a Knowledge Representation
- Step 5 - Evaluation of the KR on basis of the Set Requirements

The chapter will first introduce definition criteria for establishing the correct requirements. The second step of the methodology is to establish relationships between the product and process domains. Then the content information of the product models and the resource descriptions are deepened with a feature-based modelling approach. The feature-based modelling will provide the necessary content description for the virtual models. The third step formulates the connections under one concept model. The fourth step is to formulate the knowledge representation and the fifth step will evaluate the soundness and expressiveness of the knowledge representation that is developed on the basis of the first step. In each of the steps there will first be a description of the method and goals of the step and then of the actual implementation of the step.

5.1 Step 1 - Establishing Correct Requirements

The first step in the methodology presented in this chapter deals with the definition of correct requirements. Defining requirements to establish specifications is the first step in the development of a knowledge-based system. Unfortunately, in many cases, the definition of semantics and requirements based on commonly understood semantics is not done well enough. This causes problems when ambiguities in requirements surface later in the lifecycle, and more time and money is spent on fixing these ambiguities. Therefore, it is necessary for the requirements to be established in a systematic way in order to ensure their accuracy and completeness [66].

The difficulty arises from the fact that establishing requirements is a tough abstraction problem and often the preplanned implementation gets mixed with the requirements. In addition, it requires people with both communication and technical skills to do the definition in the first place. As requirements are often weak about what a system should not do, this poses potential problems in the development of dependable systems, where these requirements are necessary to ensure that the system does not enter an undefined state [66].

According to Tran [66], the first step is to form a common understanding. There is no point in trying to establish exact specifications if the designers and end users cannot agree on what the requirements are. A simple case is where ambiguousness in the design specification allows multiple different interpretations. For example, the requirement states that there is a need to create a means that would transport a small patch from location A to location B. Possible interpretations of this requirement include building a conveyor, an automated guided vehicle, a forklift or a crane, among other possibilities. Although each of these transportation devices satisfies the requirement, they are certainly very different.

Ambiguity can be caused by missing requirements, ambiguous words, or elements that have been introduced. The above requirement does not state how fast the patch should be transported from location A to location B. Taking a forklift would certainly be cheaper than building and operating a conveyor or crane. There are also missing requirements, such as the meaning of *"a small patch size"* in the above requirement, which is an example of ambiguous words. What exactly does *"small"* imply? A few products in a tray, a few trays, or a set of trays grouped together? The requirement states that a means should be created, not a transportation device designed. This is an example of elements that have been introduced when an incorrect meaning has slipped into the discussion. It is important to eliminate or at least reduce ambiguities as early as possible, because their cost increases along the progress [66].

The requirements for the knowledge representation are:

- the model will represent a selected domain, which is manufacturing and assembly process modelling;
- semantics are defined in such a way that the meaning of each structure in the knowledge representation is clear and there is no ambiguity in terminology;
- the precision of terms and definitions is in agreement with the properties of the structures used;
- the proposed knowledge representation is interpretable by humans and machines;
- the proposed knowledge representation is suitable for reasoning, and
- the proposed knowledge representation must be suitable for use in a dynamically adaptive operating environment where the product characteristics impose requirements on the resource capabilities.

5.2 Step 2 - Definition of Connection Between Domains

Step 2 of the methodology will deal with the definition of the connections between the product, process, and system domains. As defined in Chapter 3, the term *'product'* represents the digital and virtual (visualisation) model of a product. *'Process'* describes the actions taken in order to manufacture or assemble it. The term *'feature'* describes any geometric or non-geometric properties of a part, product or resource. The domain system is dedicated to describing physical resources that include the operators, machines, areas of action, and software blocks connected to the machines.

The rise of virtual factory theorems and requirements for more accurate digital representations of both products and process capabilities have created the need for a product-process definition. There have been ideological efforts to combine product assembly requirements with process planning. The efforts have been more methodological than actual implementations. However, despite the efforts, the meaning of the product-specific features has been lost on the assembly and manufacturing process side. The reasoning of the technical solutions about the assembly or manufacturing processes has, in most cases, been too specialised for broader use. The other drawback has been the sole concentration on the connection between processes and system instead of the connection between the product requirements and needed manufacturing capabilities.

Feature-based Modeling and Analysis

As a starting point, the definitions used for feature-based modelling and feature classification must be clarified. A feature is any geometric or functional element or property of an object that is useful in understanding its function, behaviour, or performance. Features combine geometric and non-geometric information together. In mechanical engineering, different geometric features can be distinguished: protrusions, slots, ribs, and pockets are typical examples of frequently used and recognised ones [36].

- Geometric Product Modelling: features are elements used in generating, analysing, or evaluating designs
- Design of Manufacturing: features represent shapes and technological attributes associated with manufacturing operations and tools
- Design of Assembly Processes: features represent shapes and technological attributes associated with assembly tasks, processes, operations, and tools

Feature-based modelling and analysis is used to add meaning to the product data. Features can be either geometric, such as edges, faces, holes, pockets, slots, ribs, and pads, or non-geometric product information, such as tolerances, material, density, handling requirements, and process descriptions. As

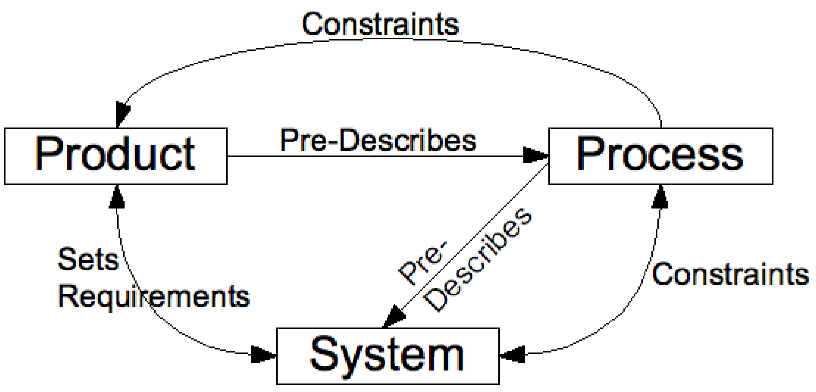


Figure 5.1: Product, Process, and System Connectivity Graph

explained by Lanz in [36], the product model will include features from both of the categories.

Product, Process, and System Connectivity

In order to combine the product characteristics with the assembly process needs, the Assembly-State Decomposition Model (ASDM) was developed. The ASD model, as presented by Lanz et al [36, 42], continues from the AND/OR graph theory by concentrating on the description of the minimal number of assembly processes required. The ASDM acts as a missing link between feature-based assembly process planning and assembly graph theory. However, the ASDM does not describe activities at the feature level. For describing manufacturing-related activities at the geometric feature level, the Manufacturing State Decomposition Model (MSDM) was created. The MSDM continues as a stand-alone model separate from the ASDM.

Figure 5.1 shows the conceptualisation of the connectivity between the different domains. It can be seen that the characteristics of a product predescribe a set of processes needed to manufacture and/or assemble the product. The description of the product pre-describes the processes. The processes pre-describe the system requirements and constrain the set of systems. The system domain includes the defined resources and functions related to specific resource types, such as scripts or function blocks. A direct mapping between the three different domains allows an immediate action in any of the processes on the basis of the requirements of the other two. The characteristics of products and systems are defined via geometric and non-geometric features, which are described through a domain ontology.

The terminology used for modelling with the ASDM is as follows:

- Assembly - an assembly consists of a number of components. A component can be either a part or a sub-assembly.

- Assembly Action - a single action, such as translation, rotation, or the application of a force that can change the state of the targeted object (product, assembly, part) - see Figure 5.2.

- Assembly task - combined and more complicated movements such as move along a trajectory, grasp, release, and join - also in Figure 5.2.

- Assembly operation - a series of operations, such as insert, screw, clue and press-fit, that contributes to the formation of the product by bringing together parts and assemblies - see Figure 5.2.

- Assembly phase - an assembly phase is defined as the assembly state in which different assembly operations can be performed on the initial configuration of the product (stable assembly, initial sub-assembly, or base part) with the condition that only one operation can be performed upon one part.

- Base part - the base part is the first part in the assembly sequence. The base part influences the main assembly direction. Together with the fixture it determines stability properties.

- Mating conditions - a mating condition is the requirement of a part for accepting other parts. The mating condition can be either geometric or functional. It defines the way parts have to be assembled and hence defines the assembly processes - see Figure 5.6

- Part - a part is the elementary component of an assembly.

- Stable assembly - there are several types of stability. The part can be analysed for stability in the feeding process and in the feeding position, and for stability of the grasp during the move or mount trajectory. The partial assembly can be analysed for stability in the assembly position, during the mount process, and during transport.

- Sub-assembly - a stable assembly that contains two or more parts.

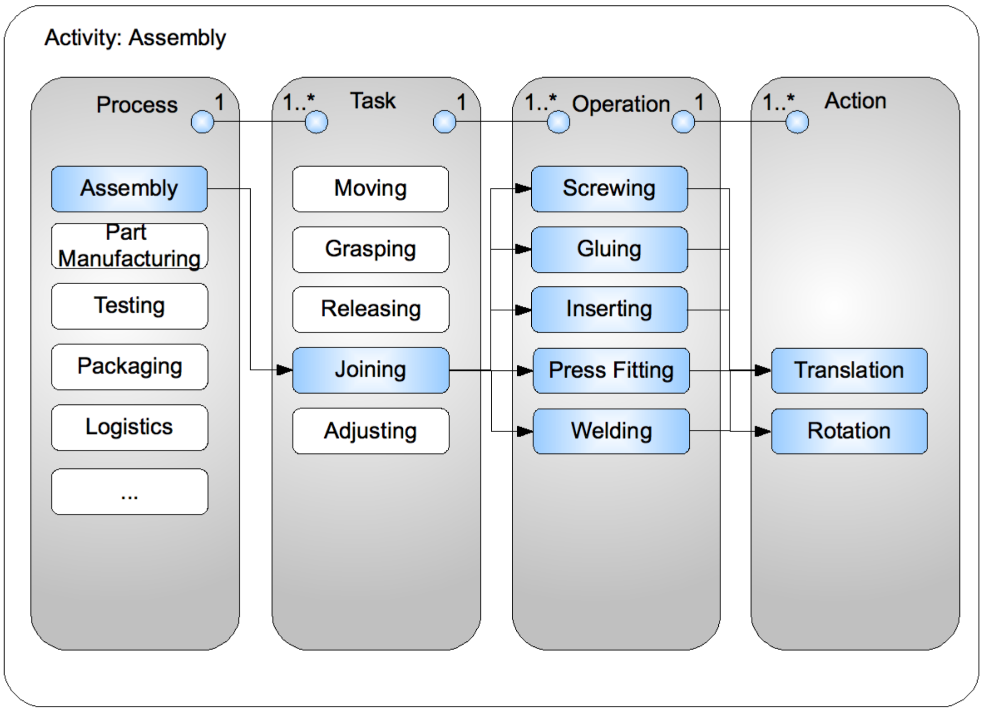


Figure 5.2: Assembly Activity Levels in the form of a taxonomy

- Transition - a transition is the event of changing from one assembly phase to the next one [36, 42].

The diagrams that compose the product model are as follows:

- Direct connection between parts - this is represented by an arrow pointing in the assembly direction. In Figure 5.3, part 1 is assembled directly to part 2.

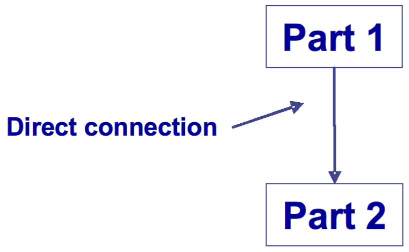


Figure 5.3: ASD model - Direct Connection

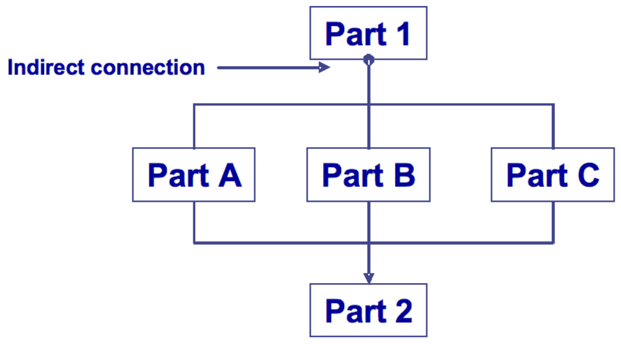


Figure 5.4: ASD model - Indirect Connection

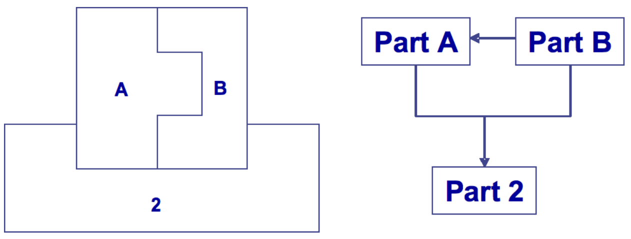


Figure 5.5: ASD model - Sub-assembly

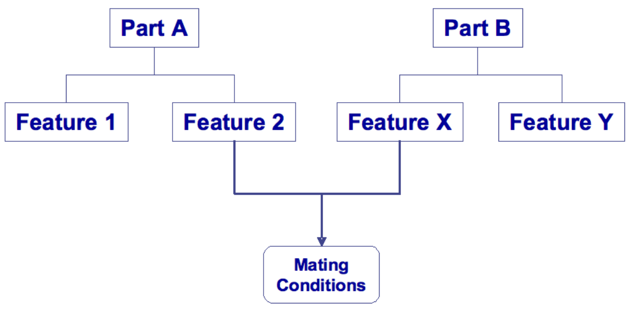


Figure 5.6: ASD model - Mating Conditions

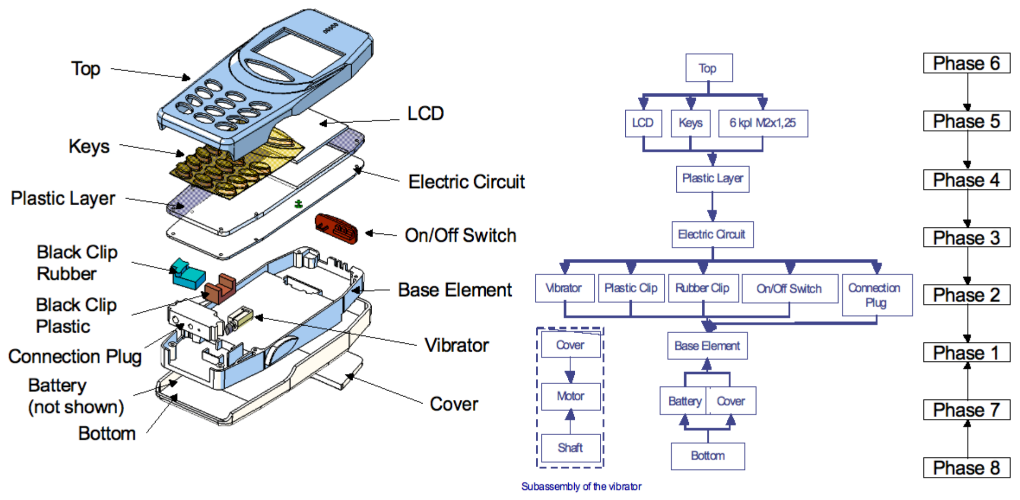


Figure 5.7: Assembly State Decomposition model of the product view and a case product

- Indirect connection between parts - this is represented by a point embedded in the part block. In the figure 5.4, part 1 is directly assembled onto part 2; however, precedence constraints state that parts A, B, and C should be assembled prior to part 1.

- Sub-assemblies - these are represented by a direct connection between parts in the same assembly phase; see Figure 5.5. The assembly direction of the sub-assembly can be defined in the same manner as with any other assembly. A connection between parts can be bi-directional when a base part is not properly defined. This offers some flexibility, but increases the difficulty of the decision-making process in the assembly planning.

The physical connection between parts is defined through feature recognition. Features are defined as any geometric or functional element or property of an object that is useful for understanding its function, behaviour, or performance. Figure 5.6 shows a schematic representation of the connection between two parts on the basis of feature information and mating conditions.

A case product that utilises the rules of the ASDM is shown in Figure 5.7.

On the basis of the assembly modelling methodology and the mating con-

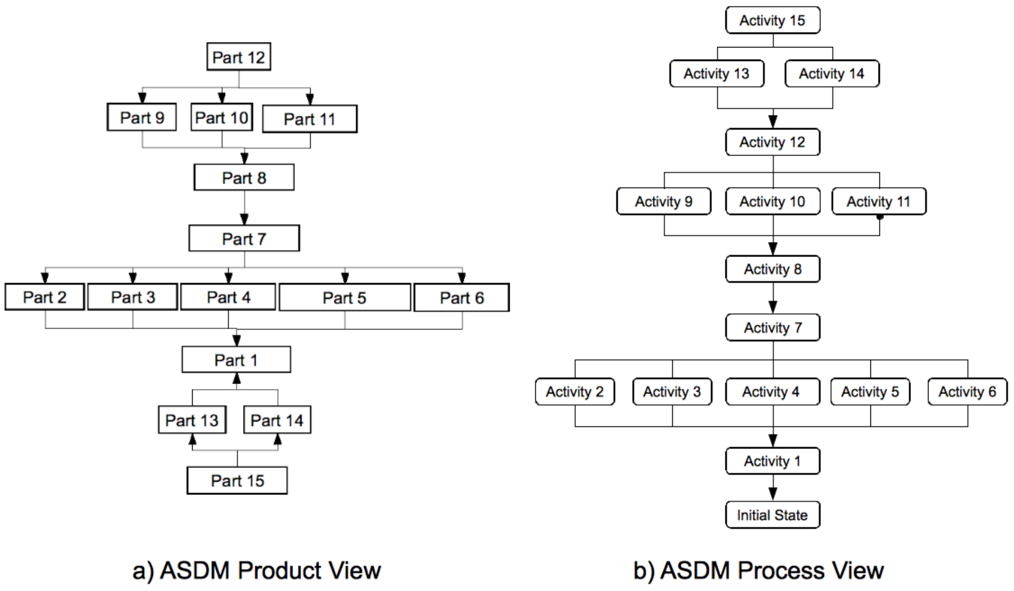


Figure 5.8: ASDM product and corresponding activity graph from the case product

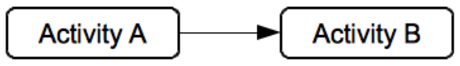


Figure 5.9: ASD model of the Activity View - Prior Activity

dition analysis performed in the product model, an activity view of the assembly process can be formed. Figure 5.8 shows the product view on the left-hand side of the figure and the process view of the same case product on the right-hand side of the figure. The process representation also includes special diagrams to represent specific situations in assembly. They are as follows:

- Prior Activity - a prior activity is represented with an arrow from the side of one activity to the other, representing the directional flow of the activities. The current case product shows this diagram for the vibrator assembly, and is a direct connection between process 2 and the prior activity box; see Figure 5.9.

- Repetitive Activity - when an activity is repetitive and can be performed in the same assembly phase, such as screwing six screws into a

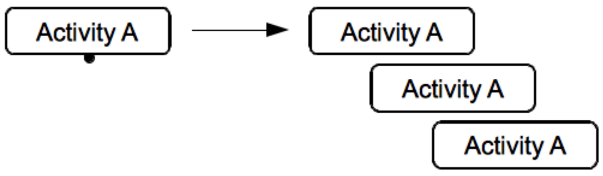


Figure 5.10: ASD model of the Activity View - Repetitive Activity

plate [36, 42], see Figure 5.10.

For the Manufacturing State Decomposition Model (MSDM), which is a sister model to the ASD model, the activities are connected to a geometric feature level as illustrated in Figure 5.11. In this case the primitive features, such as a round hole and rectangle, are combined into more advanced feature sets such as a rounded hole. Rounded holes consist of two primitive types: rectangles and round holes. In the case of sheet metal, the tools are, for example, a round hole and a rectangle. By combining these primitive features the feature can be nibbled or cut straight away with a laser. Figure 5.11 shows repetitive activities on the manufacturing feature set level. However, the feature sets such as the rounded hole can also be divided into single primitive features, each connected to a single punch operation.

- Part - the base part upon which the features are inflicted
- Primitive Feature - a rectangular pocket or round hole
- Manufacturing Feature Set - a combination of primitive geometric features and non-geometric features
- Manufacturing Task - a combined and more complicated movement such as drilling, machining, punching, and laser cutting
- Manufacturing Operation - a series of operations, such as hole cutting or edge cutting
- Manufacturing Action - a single action, such as translation or rotation
- Manufacturing Phase - a phase is defined as the state in which different manufacturing operations can be performed on the initial part with the condition that one or more operations are performed upon one manufacturing feature.

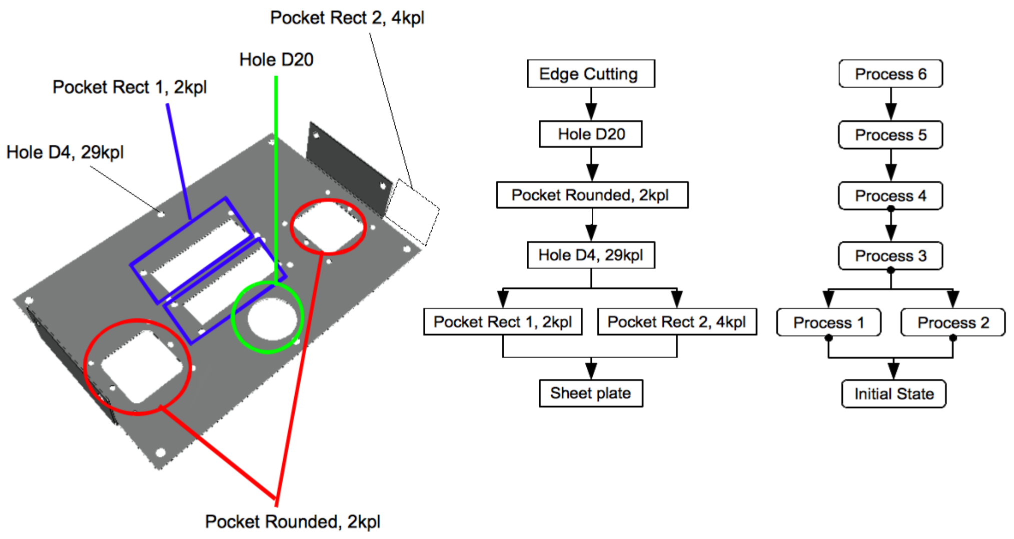


Figure 5.11: Manufacturing State Decomposition Model of the part view and a case product

- Transition - a transition is the event of changing from one manufacturing phase to the next one [36, 42].

For the manufacturing state decomposition model (MSDM), the process descriptions are the same as for the ASDM; however, they are, naturally, on a different conceptual level. The ASDM provides an understanding of product structure, rules, constraints, and assembly-specific information in relation to the product model. The MSDM provides the connection between the manufacturing feature level and processes connected to single features or sets of features. In the case of the ASDM, the definition of activities defines the requirements for the system. However, in the case of discrete part manufacturing, the requirements are clearer. For example, a rounded hole in a sheet metal product, which is a manufacturing feature set, requires cutting. The machine types that are capable of completing the required actions are a laser or punching machine. The type of material used and its thickness, represented as non-geometric features, define the requirements even more closely. So, in other words, the product and system are connected through processes.

5.3 Step 3 - Generation of a Conceptual Model

Step 3 will focus on creating a conceptual model based on the ASDM and MSDM. The goal is to connect these two different models under one generic Product-Process-System model.

There are several obstacles or barriers that make knowledge capture and management rather difficult. One is the vast amount of heterogeneous information being captured. The other one is the description of the information. Traditional methods of capturing knowledge from information fail when applied to large, highly complex, and interconnected bodies of semantic information [58]. The formal practice of information modelling as a discipline supports the wider field of systems analysis, which encompasses both information and activity modelling. Ray and Wallace [58] defined two main principles of modelling such information systems as early as in 1995.

- First requirement: the most important step in the modelling of information structure is *consensus building*. The modelling method must support human communication and interpretation of the resulting model.
- Second requirement: for large and complex models the representation must be machine-interpretable and thus valid.

Starting with the theoretical background, a product model is understood as a representation of product-specific information which describes a particular product or a product family. A product model is a representation of product information. Product information describes a particular product or a product family. The computerised representation of product knowledge requires a formal representation, which defines the form and types of content, and a corresponding product model, which contains information specific to a particular product.

The product model typically contains a combination of single parts, where different instances can be of the same type. Some of the parts are not always directly assembled into the main product; the existence of sub-assemblies is allowed. Single parts and sub-assemblies are expected to be stable, and they can be assembled into the main assembly [28, 36].

The traditional modelling inside the CAD environment is restricted mainly to the product model. This means that process and system information cannot be added into the CAD file. However, the ontological representation, introduced later on, instead of restricting the product-process connection and additional knowledge, allows the possibility of adding process and system information and provides a means for semantic reasoning and knowledge discovery.

In the conceptual PPS (Product-Process-System) knowledge model, introduced in [37, 38], the assembly-specific feature information is divided into four semantic layers: *Features, Feature Sets, Assembly Features* and *Assembly Feature Sets*. A feature can be a certain face or an edge. Feature sets are more complex combinations of recognised and meaningful faces. The assembly features include the assembly process requirements [36, 38]. Consequently, the assembly feature sets are complex feature sets defining a set of assembly requirements. This hierarchical, feature-class representation can help in the definition of the structure of a knowledge representation in relation to process and system knowledge. The feature sets that are recognised are used as a basis for the reasoning and parsing of the knowledge.

The conceptual PPS knowledge model, illustrated in Figure 5.12, defines how the *Process Model, Reasoning Machine, Reasoning Results and* Product Model are integrated. At the conceptual level the product-specific information is connected into the processes through the two classes: *Reasoning Machine and* Reasoning Results [38].

The main focus for the knowledge representation is the product and the actions that occur upon it. This viewpoint is called a product-centric view. Another viewpoint for the model is the resource-centric viewpoint, illustrated in Figure 5.13. The knowledge for the resource can be very vague and it can be temporally based on data acquisition and aggregation from a physical machine via performance logs or it can have a very detailed description of its characteristics and a corresponding visualisation. The resource-centric view utilises the Object domain to represent the structural aspects of a resource. The geometric and non-geometric features, as well as the life-cycle states, also need to be modelled for each resource.

The characteristics of the normal Product-centric viewpoint are:

- System is defined as resources connected to product through activity classes.

- Activities describe what happens between parts or for forming a feature.

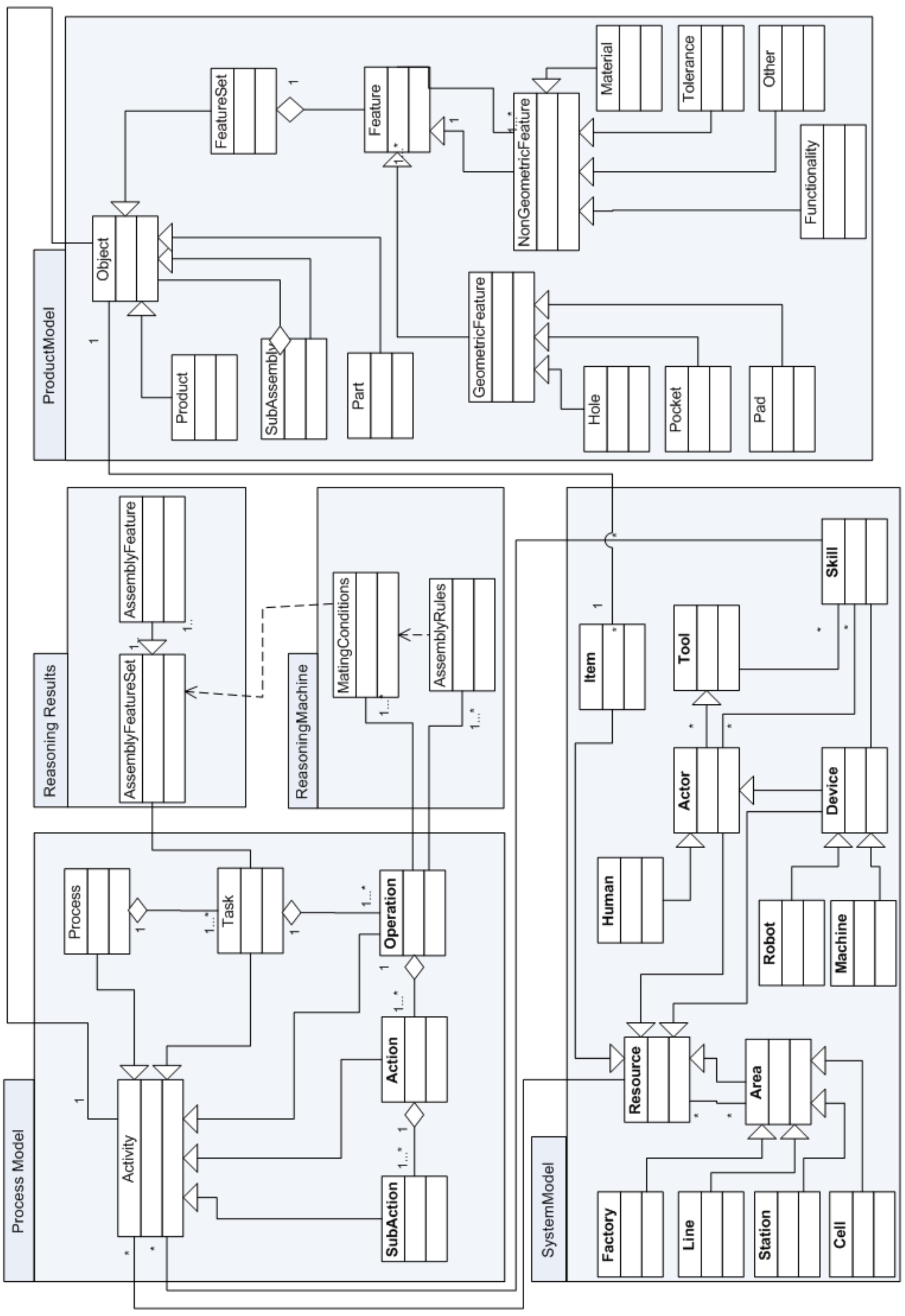


Figure 5.12: Product-Process-System Model

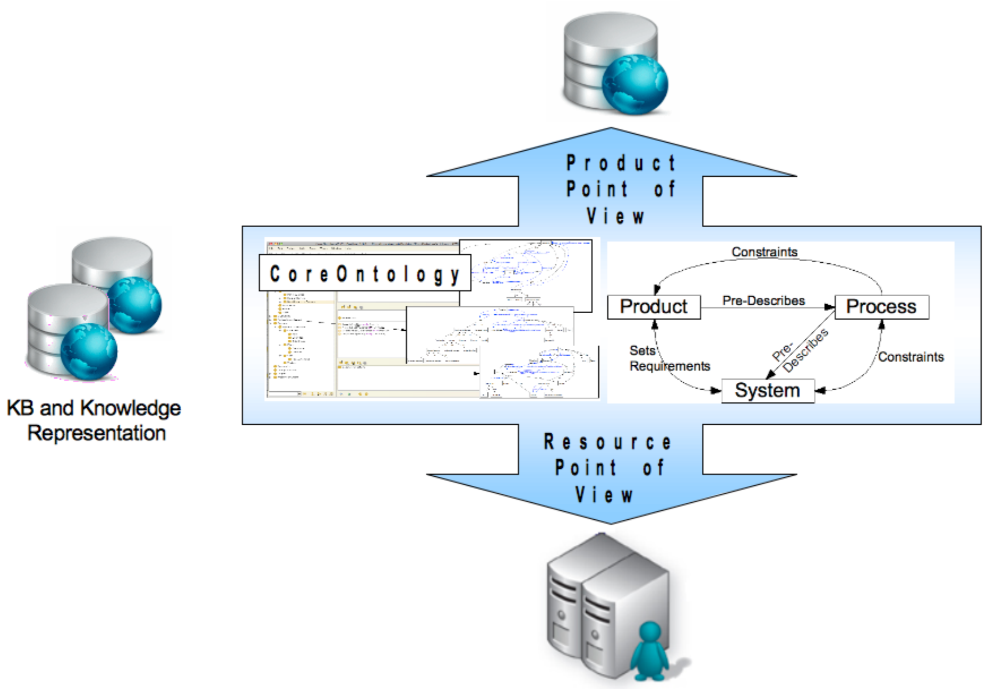


Figure 5.13: Product and Resource viewpoints

- Product is described as planned and/or manufactured.

The characteristics of the Resource-centric view are:

- The characteristics of Resources are described through the product and resource domains of the ontology.
- Activities describe what happens between parts inside of a system forming a set of functionalities.
- Groups of functionalities form the capability of a resource.
- Resource is capable of performing a task or activity multiple times.
- System is described as an initial state, current state, and any state (prediction based on the operational log).

5.4 Step 4 - Creation of a Knowledge Representation

This step will formalise the conceptual model into a Knowledge Representation (KR) that structures the relevant knowledge. The relations on different levels of abstraction are taken into account. In order to be a useful solution, the KR must represent the Product-Process-System model introduced in the previous step; i.e., it must be able to capture the connections between the product, process, and system domains.

The following section will define the necessary conditions between the classes. Since the technical problem is more a classification problem than an inference problem, only the most important axioms are modelled here. However, as the KR is designed to be the foundation element of an intelligent manufacturing environment it must include semantics for inference. Other notes must also be made. There is no time index in the KR, since in this domain it is rarely modelled or considered to exist. For example, the stages of modification of a part in relation to time are not modelled, i.e. part A is part A during the modification and it is not modelled as raw material evolving into the physical part A.

The connection between the *Object, Activity*, and *Resource* layers is shown in Figure 5.14. The Object class contains three subclasses: *Product, SubAssembly*, and *Part*. The Part is *partOf* SubAssembly and *partOf* Product.

The class Part is the smallest entity under the super-class Object, where the x represents a concept in the class Part and y any other concept, and the Axiom for the class Part is following:

$$\forall x(Partx \rightarrow \neg\exists y(y < x)) \tag{5.1}$$

The class Product is the largest entity under the super class Object, where the x represents a concept in the class Product and y any other concept, and the Axiom for the Class Product is the following:

$$\forall x(Productx \rightarrow \neg\exists y(x < y)) \tag{5.2}$$

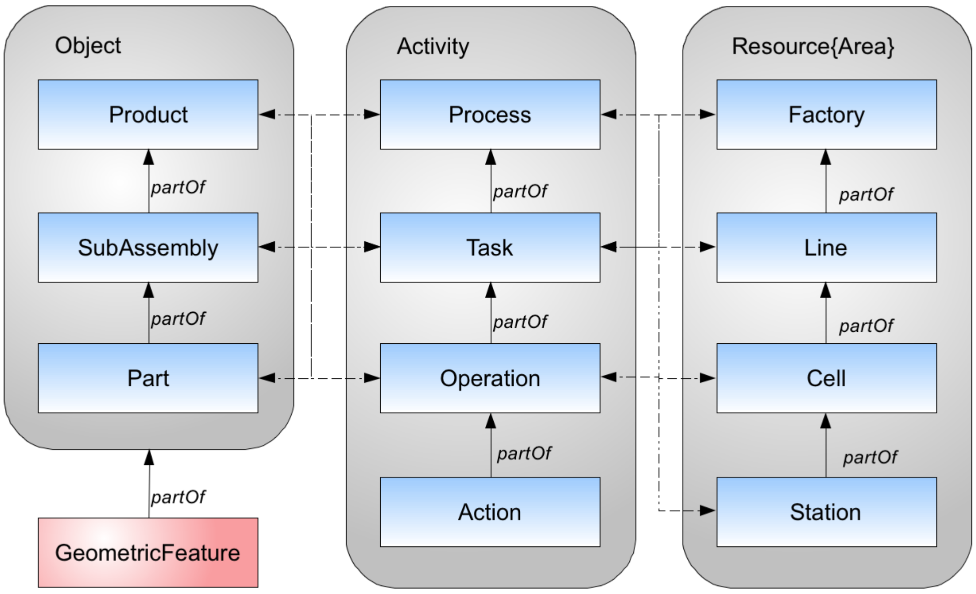


Figure 5.14: Connection of the Object, Activity, and Resource (Area) levels

Since Parts and SubAssemblies can exist in a SubAssembly, where the x represents a concept in the class SubAssembly and y any other concept, the axioms for the class SubAssembly are:

$$\forall x(SubAssemblyx \rightarrow \exists y(y < x)) \tag{5.3}$$

$$\forall x(SubAssemblyx \rightarrow \exists y(x < y)) \tag{5.4}$$

For all Object there must be defined instance from the classes Part or Sub-Assembly or Product:

$$\forall x(Objectx \leftrightarrow Partx \vee SubAssemblyx \vee Productx) \tag{5.5}$$

On the basis of the axioms, it is defined that there cannot be a Part that is a Subassembly or a Product that is a SubAssembly:

$$\neg\exists x(Partx \wedge SubAssemblyx) \tag{5.6}$$

where the x represents a concept in the corresponding class.

$$\neg\exists x(Productx \wedge SubAssemblyx) \tag{5.7}$$

where the x represents a concept in the corresponding class.

The *GeometricFeature* class has a connection through the class *Feature* to the class *Object*. On the conceptual level a part (model) can be divided into smaller units, such as edges and faces. However, since features can be connected to Products and SubAssemblies as well, and to Resources, the class is not part of the Object class. The GeometricFeature class is a dependent entity that cannot exist without the Object class, i.e. a GeometricFeature is nothing in itself.

$$\forall x\exists y(GeometricFeaturex \rightarrow Objecty) \tag{5.8}$$

where the x represents a concept in the class GeometricFeature and y a concept in the class Object.

Similar structures exist for Activities as for Objects. The Process class is the highest class in the Activity. Under Tasks are *partOf* Processes; Operations are *partOf* Tasks and Actions are *partOf* Tasks. The following axioms describe these relations.

The Action class is the lowest class in Activity:

$$\forall x(Actionx \rightarrow \neg\exists y(y < x)) \tag{5.9}$$

The Process class is the highest class in Activity:

$$\forall x(Processx \rightarrow \neg\exists y(x < y)) \tag{5.10}$$

For all instances of Activity there must be a defined instance from the Process or Task or Operation or Activity classes:

$$\forall x(Activityx \leftrightarrow Processx \lor Taskx \lor Operationx \lor Activityx) \qquad (5.11)$$

For Resource(Area) entities the axioms are following:

The Station class is the lowest class in Resource(Area), where the x represents a concept in the class Station and y any other concept:

$$\forall x(Stationx \rightarrow \neg\exists y(y < x)) \qquad (5.12)$$

The Factory class is the highest class in Area, where the x represents a concept in the class Factory and y any other concept:

$$\forall x(Factoryx \rightarrow \neg\exists y(x < y)) \qquad (5.13)$$

For all instances of an Area there must be a defined instance from the Factory or Line or Cell or Station classes, where x represents any concept from the corresponding class:

$$\forall x(Areax \leftrightarrow Factoryx \lor Linex \lor Cellx \lor Stationx) \qquad (5.14)$$

The rest of the Resource class structures are illustrated in Figure 5.15. The axiom for an Actor is as follows, where x represents any concept from the corresponding class:

$$\forall x(Actorx \leftrightarrow Devicex \lor Humanx \lor Softwarex) \qquad (5.15)$$

Figure 5.16 illustrates the structure of the Device classification. The axiom for the Device class is as follows, where x represents a concept from the corresponding class:

$$\forall x(Devicex \leftrightarrow ComplexDevicex \lor SimpleDevicex) \qquad (5.16)$$

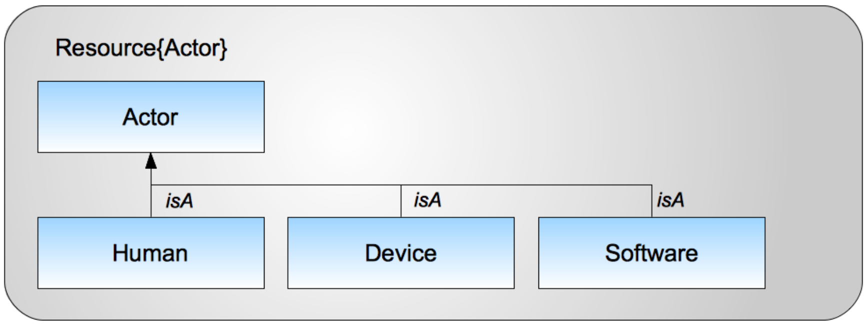


Figure 5.15: Resource(Actor) classes

These classes are purely for classifying what exists on a factory floor. For example, a robot unit as a Complex Device consists of ComplexDevice(Robot) and ComplexDevice(Gripper) and SimpleDevice(Tool), where x represents any concept from the corresponding class, is as follows:

$$ComplexDevice(Robotunit)$$
$$=$$
$$Robot(x) + Gripper(x) + GripperFinger(x) + GripperFinger(y)$$

The *GeometricFeature* class is different from the other hierarchical class structures since it includes *isA* and *partOf* relationships. As defined above, the Vertex class is the most atomic structure in this class structure. An edge is formed when there are two vertices forming a stable structure. Since these classes come from the visualisation, the edge is defined with at least two vertices.

For creating a face at least three vertices that are connected with three edges must exist. *Face-Extended* is defined only for the needs of the feature recognition tool. The Face-Extendeds are faces that can include holes and pockets. The Face-Extendeds can remain in a datum or they can be described as mathematical surfaces.The *Complex* class is reserved for the recognised features, such as primitive features and undefined features. The primitive features are considered to be *Cylinder, Box*, and *Triangle.* If a form that is not recognised as primitive is found, it will be saved to the *OddShape* class.

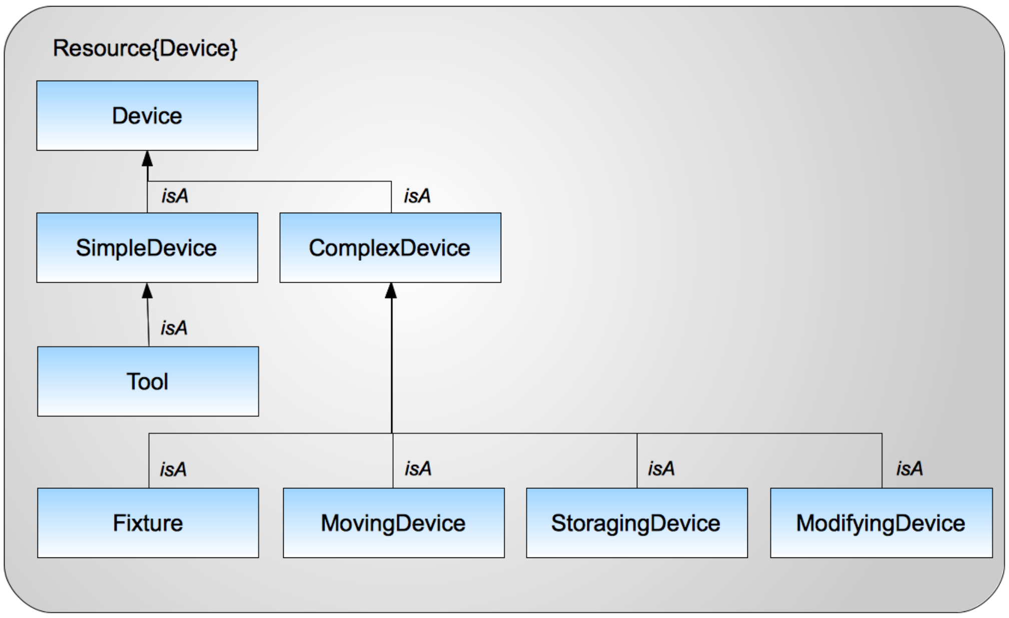


Figure 5.16: Resource(Device) classes

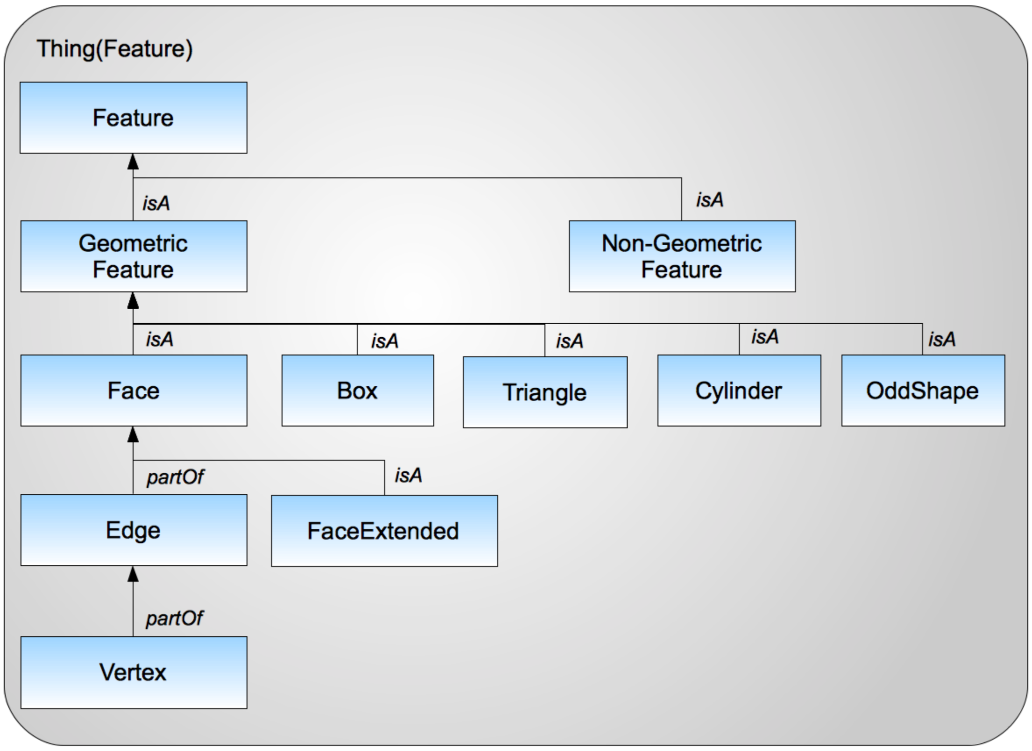


Figure 5.17: Feature classes

Figure 5.17 illustrates the structure of the Feature class.

The *Vertex* class is the smallest entity under the *Geometric Feature* class and the axiom for the Vertex class is as follows:

$$\forall x(Vertexx \rightarrow \neg\exists y(y < x)) \tag{5.17}$$

where the x represents a concept in the class Vertex and y any other concept.

The Face class is the largest entity under the super-class GeometricFeature, where the x represents a concept in the class Face and y any other concept, and the axiom for the Class Face is as follows:

$$\forall x(Facex \rightarrow \neg\exists y(x < y)) \tag{5.18}$$

Since Edges and Vertices can exist in an Edge, where the x represents a concept in the class Edge and y any other concept, the axioms for an Edge are such as:

$$\forall x(Edgex \rightarrow \exists y(y < x)) \tag{5.19}$$

$$\forall x(Edgex \rightarrow \exists y(x < y)) \tag{5.20}$$

For all instances of a GeometricFeature there must be a defined instance from the Edge or Vertex or Face or Complex classes:

$$\forall x(GeometricFeaturex \leftrightarrow Vertexx \vee Edgex \vee Facex \vee Complex) \tag{5.21}$$

where the x represents a concept in the corresponding class.

The knowledge representation aims to formalise and capture the meaning of entities. Previously, ontological modelling aimed at providing a knowledge representation, which allows inferences to be to be applied upon the content. However, in recent years, according to Gruber [26] and, later on, Cech [8],the ontology design actions should also take into account the interoperability criteria, extending the design requirements as follows:

- *Clarity*: the ontology should effectively communicate the intended meaning of defined terms. Definitions should be objective and documented with natural language.

- *Coherence*:the ontology should approve only inferences that are consistent with the definitions.

- *Extendability*: the ontology should be designed to anticipate the uses of the shared vocabulary. The user should be able to define new terms for special uses based on the existing vocabulary in such a way that the re-visioning of the ontology is not needed.

- *Minimal encoding bias*: the conceptualisation should be specified at the knowledge level without depending on a particular symbol-level encoding. Encoding bias should be minimised, because knowledge-sharing agents may be implemented in different representation systems an styles of representation.

- *Minimal ontological commitment*: the ontology should require the minimal ontological commitment sufficient to support the intended knowledge-sharing activities. The ontology should make as few claims as possible about the world being modelled.

Several types of ontology languages exist, each having slightly different knowledge representation expressivenesses and reasoning mechanisms. The selection of an ontology language is not an easy task. As stated previously in [51], artificial intelligence-based languages such as KIF and Ontolingua and ontology markup languages (RDF(S), DAML+OIL, OWL) are better suited to representing and implementing ontologies than UML and ER diagrams.

For exchanging ontologies between applications, languages based on XML are easily read and managed since standard libraries for the treatment of XML are available. Among the above-named markup languages, OWL DL is one candidate for ontology expression since it provides expressiveness while ensuring that all valid conclusions can be inferred and that inferences are deterministic [51]. However, the light weight ontology language such as OWL DL, is still a compromise between expressiveness and implementation, since it does not provide all of the needed logical expressions in its vocabulary. For example a basic concept such as the *partOf* relation does not exist in OWL as in FOL.

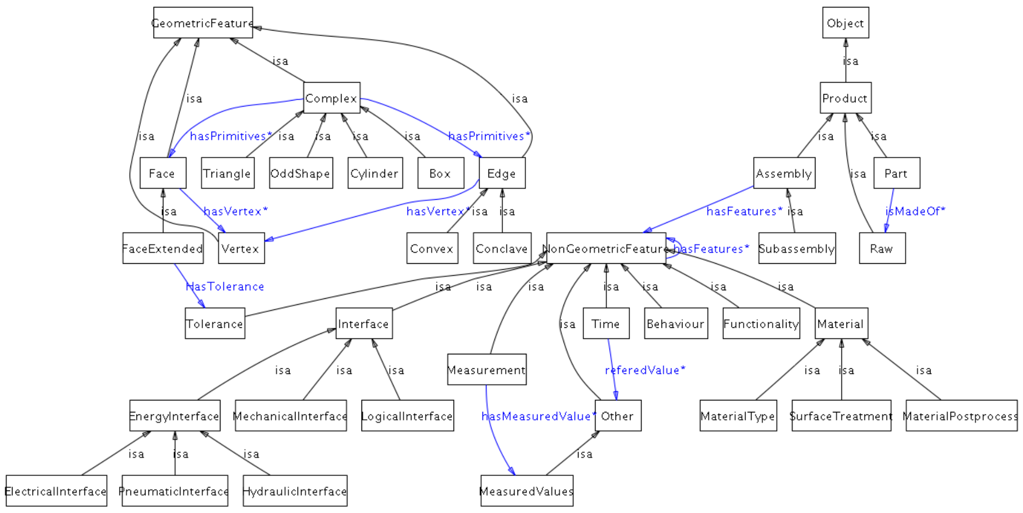


Figure 5.18: Product Definition in the Core Ontology

The purpose of the ontology is to provide a generic description of three different domains that are traditionally separated (but in most cases overlapping) and combine the common terminology under one domain ontology. The three domains are introduced separately for the sake of the reader's ease of understanding. The connections between the domains are explained in each section. For modelling the ontology an ontology modelling tool, Protégé 3.4. [24], was used.

5.4.1 Core Ontology - Product Definition

The product section[1] in the Core Ontology, introduced in Figure 5.18, is used for describing and modelling the product-specific information inside the knowledge base. The product ontology supports variable types of product structures with rich meta-information. Additional information, if needed, is presented as non-geometric features of the object and life-cycle-related information.

[1]The term *'product'* is used here for describing this particular ontology domain. However, in order to solve technical modelling issues, the superclass of this domain is Object. This allows a part to be modelled without defining its place in a subassembly or in a product.

Definition for the Classes

In the product ontology, illustrated in Figure 5.18, the Part and Sub-Assembly classes all have a connection to the Product classes, as illustrated in Table 5.1.

Table 5.1: Core Ontology: Class Product

Product		
modelUrlt	-	String
hasFailureMode	Instance(s)	FailureMode
refersToActivity	Instance(s)	Activity
providesSkill	Instance(s)	-
hasAssembly	Instance(s)	-

Table 5.2: Core Ontology: Class SubAssembly

SubAssembly		
hasFeatures	Instance(s)	NongeometricFeature
hasParts	Instance(s)	-

Table 5.3: Core Ontology: Class Part

Part		
hasTask	Instance(s)	Task
isMadeOf	Instance(s)	Raw

In order to represent the required detailed information of the product, the *Feature* class was established. The Feature class has two sub-classes: *Geometric Feature* and *Non-Geometric Feature.* The Product class is separated from the Feature class, but has a connection through it to the Geometric and Non-Geometric Feature classes. This structure will allow more freedom to describe the different detail levels of the current product compared with a traditional strictly hierarchical presentation of product structures.

If there is a need to add deeper meaning to the product or the actual meaningful features to the part/product model, it is done via Geometric Features. As the product model is exported from one CAD system to another CAD or simulation system, most of the geometric features are lost. In the worst-case scenario, even the boundary representation information is lost. An example

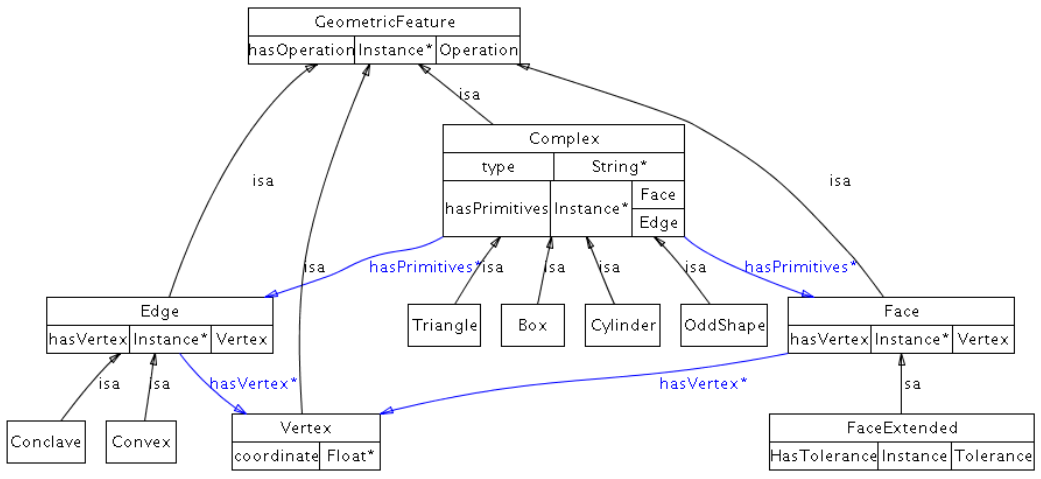


Figure 5.19: Geometric Feature Class and its sub-classes

of this information loss is a round hole when virtual reality modelling language (VRML) is utilised as a visualisation language. While it renders it as a round one, a closer inspection might reveal that instead of a cylindrical hole there is a cloud of points. Depending on the capabilities of system B, it may or may not understand that this particular cloud of points is actually a representation of a round hole.

The feature recognition tool that was utilised, explained in more detail in Section 7, Case 1, Pro-FMA, parses the product model and re-creates the lost features. The value of the diameter of a hole is saved under the Non-

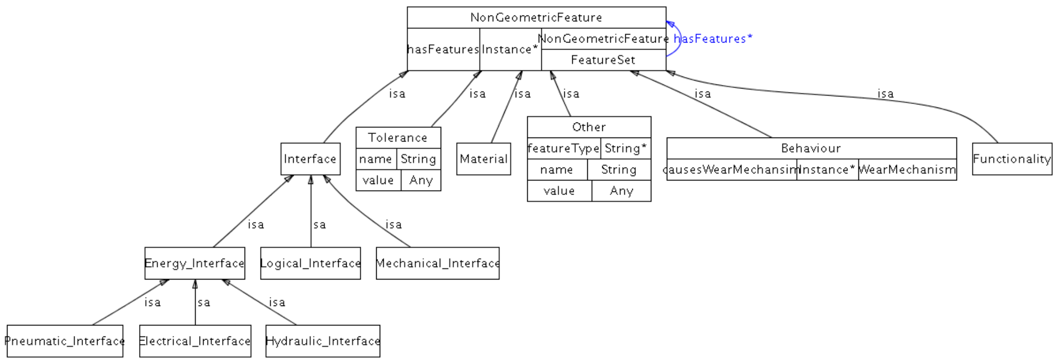


Figure 5.20: Non-geometric Feature Class and its sub-classes

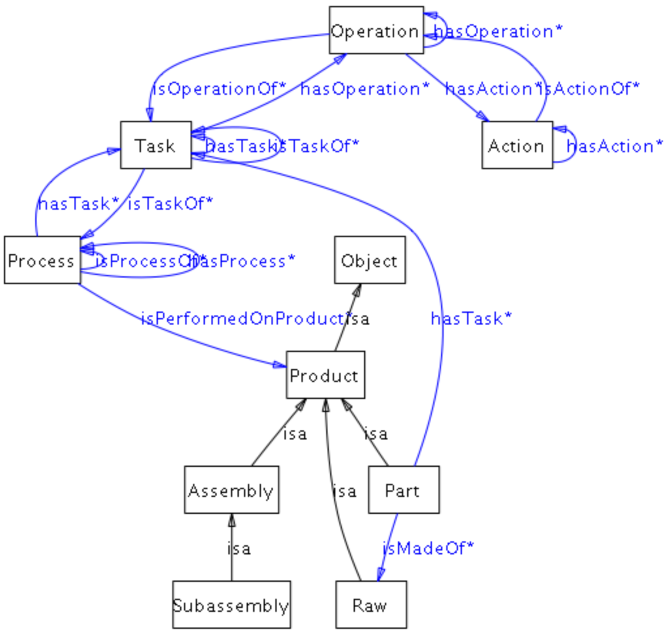


Figure 5.21: Process Definition in the Core Ontology

Geometric Feature class.

The product ontology also contains information about the life-cycle phases of the product. The life-cycle phase information will define whether or not the part or the product is allowed to be used in an assembly or machining simulation. The product ontology is connected to the system ontology through processes but also through device classes. The structure of a certain device can partly be described in detail with the product ontology.

5.4.2 Core Ontology - Process Definition

Manufacturing and assembly processes [2] are described with a process ontology, see Figure 5.21. The process ontology is the key operator when combining the product and the system knowledge.

The process ontology is the key operator when the product and the system

[2]The term *'process'* is used here for describing this particular ontology domain. However, in order to solve technical modelling issues, the superclass of this domain is Activity. This allows operations and tasks to be modelled without defining the term's superclass.

knowledge are being combined. The process ontology defines the activities required to realise the product. However, during the conceptual process modelling the resources might not be known. The activity class is the root class in this domain. Processes are defined, for instance, as as *Part Manufacturing, Assembly, Testing* and *Packaging.*

In the case of the assembly process, the tasks are such as *move, retrieve, release*, and *join.* If the assembly task belongs to the joining class then the operations can be, for example, *screw, glue, insert* and *press-fit.* The Action class is reserved for the most basic functions, which are *rotate* and *translate* and combinations thereof, as defined earlier by Rampersad [54] and further modified by Lohse [38, 43, 45].

Compared to the architecture represented in Figure 4.6, the ADACOR architecture, the approach in this process ontology is different. The idea in this thesis' approach is not to express the classes with details such as class Setup in ADACOR. The setup actions in the Core Ontology are described through Task classes, where the content describes the actions taken into account. The idea with the Core Ontology is to describe actions and connected resources in such a way that the resource capabilities describe more than the action itself.

The process ontology, illustrated in Figure 5.21, provides shallow definitions of the types of interaction between actors and products.

Definition for the Classes

Activity, in Table 5.4, is the superclass for the process definitions.

Table 5.4: Core Ontology: Class Activity

Activity		
isPerformedOnProduct	Instance(s)	Product
requiresCapability	Instance(s)	Capability
ActivitySeqPrevious	Instance(s)	Activity
Activity SeqNext	Instance(s)	Activity
requiresResource	Instance(s)	Resource

The Process class, illustrated in Table 5.5, is the actual definition for the processes used in the realisation of a product. As previously defined, the processes have four main classes, as presented in Figure 5.2, which form a hierarchical structure. The processes are higher-level activities such as part manufacturing, assembly, packaging, and testing. It has to be noted that this division is not the result of pure reasoning but more like a guideline for understanding the levels of content.

Table 5.5: Core Ontology: Class Process

Process		
isPerformedOnProduct	Instance (s)	Product
hasProcess	Instance(s)	Process
hasTask	Instance(s)	Task
isProcess	Instance(s)	Process

Below the Process class is the class for *Tasks*. Task are defined as being parts of the process, such as drilling, milling for part manufacturing, and, for example, joining for assembly processes.

Table 5.6: Core Ontology: Class Task

Task		
hasTask	Instance(s)	Task
hasOperation	Instance(s)	Operation
isTaskOf	Instance(s)	Process
isTaskOf	Instance(s)	Task

The class for *Operations* is a subset of the Task class. The operations are for assembly processes such as snap-fitting, press-fitting, screwing, gluing, and welding or for manufacturing-side operations such as laser cutting or punching.

Table 5.7: Core Ontology: Class Operation

Operation		
hasOperation	Instance(s)	Operation
isAction	Instance(s)	Action
isOperationOf	Instance(s)	Task

The lowest-level class is called as *Action*. This class is reserved for the movements of the devices.

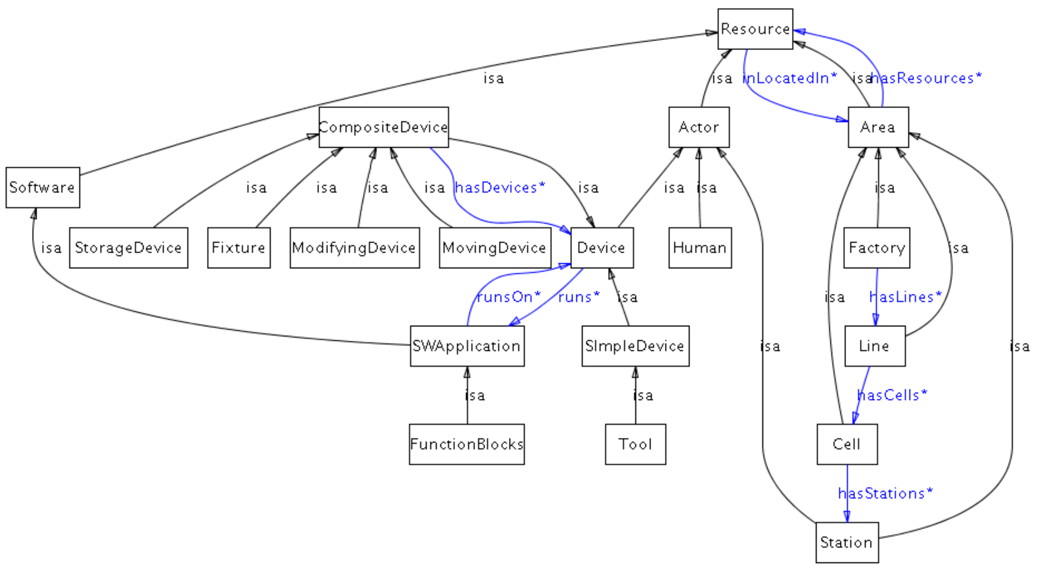


Figure 5.22: System Definition in the Core Ontology

5.4.3 Core Ontology - System Definition

The third ontology section describes the resources as illustrated in Figure 5.22. The system ontology is used to describe the manufacturing environment and its characteristics. It defines the resources assigned to the processes. It includes the life-cycles of the production equipment and system-specific programs. The life-cycles are device-specific parameters defining the characteristics of the device, such as the mean time between repairs and the mean time between failures. The system ontology defines the capabilities of an actor; see Table 5.11, where a device has a functionality, but a human has a skill or skills. The production equipment, devices, and the related software are partly defined via the product ontology, since the product ontology provides a means for defining non-geometric properties.

The Resource class, in Table 5.9, is the superclass for all of the different resources.

Table 5.8: Core Ontology: Class Action

Action		
hasAction	Instance(s)	Action
isActionOf	Instance(s)	Operation

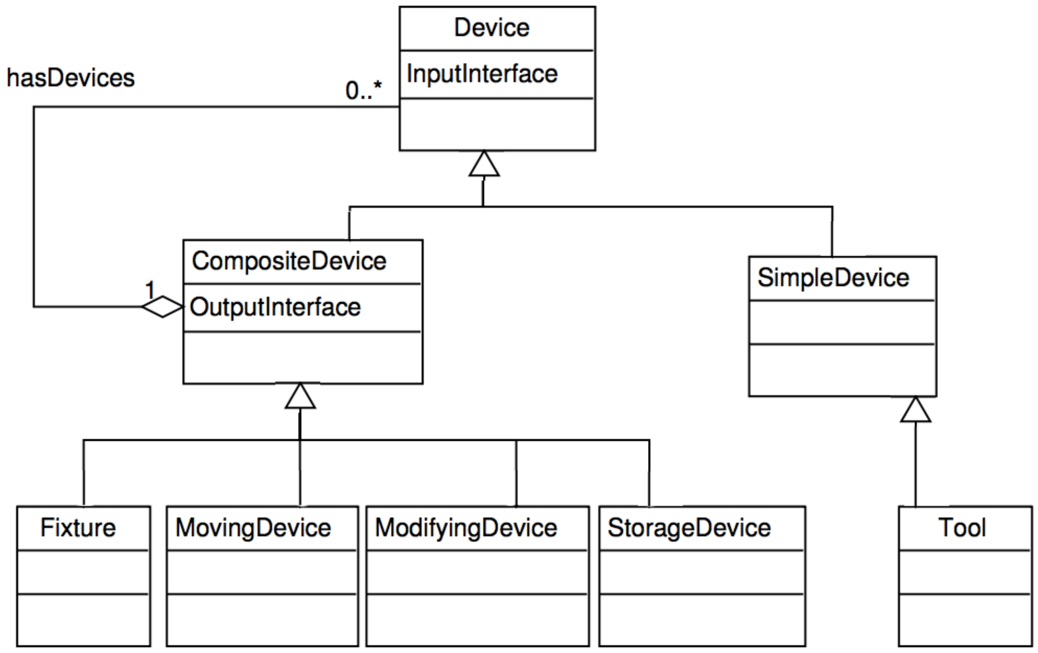


Figure 5.23: Structure of the Device Class

For the system ontology, the Area class represents the production facility. Under the area there are classes for describing the production facility on four levels of detail: Factory, Line, Cell, and Station. These classes are not necessarily the only candidates for classes, but were left in the structure because the classification follows the way in which resources are expressed in natural language in industry.

For describing the type of resource, the Actor class was inserted into the Core

Table 5.9: Core Ontology: Class Resource

Resource		
hasFunctionality	Instance(s)	-
hasLifecyclePhases	Instance(s)	LifecyclePhase
isLocatedIn	Instance(s)	Area
isInLifecyclePhase	Instance	Area

Table 5.10: Core Ontology: Class Area

Area		
hasResources	Instance(s)	Resource

Ontology. The Actor class, in Table 5.11, has three semantically different actor types. In the current stage of the development an actor is a Human, a Software, or a Device.The class device includes all of the tools, devices, in Table 5.12 and the software applications needed to operate the device.

The devices are divided into *SimpleDevices* and *CompositeDevices*, illustrated in Figure 5.23. The Composite Device class is further divided into *Fixture, Modifying Device, Storage Device*, and *Moving Device.* The *Simple Device* class has a subclass, Tool, that is the most atomic entity of a physical device resource. These sub-layers were implemented simply for human interpretation and the classification of different device categories on the basis of the primarily intended functionality of each device combination.

Again, comparing the chosen KR to previously expressed knowledge representations such as ADACOR - Figure 4.6 - and ONTOMAS - Figure 4.5 - approach taken here does not try to classify the devices on the basis of a traditional hierarchy. The main aim is to describe the resources in Figure 5.23 as devices, based on the capabilities of single or combined devices. For example, the gripper is described by its capability to grasp and hold on the class level and with more details via individual properties. The combination of capabilities is similar to the approach taken in Lohse's [45] ONTOMAS model, i.e. each capability has inputs and outputs.

Table 5.11: Core Ontology: Class Actor

Actor		
actorType	String	-
hasCapability	Instance(s)	Capability

Table 5.12: Core Ontology: Class Device

Device		
providesSkill	Instance(s)	-
runs	Instance(s)	WSApplication

5.5 Step 5 - Evaluation of the KR on the basis of the Set Requirements

The requirements for the knowledge representation were defined in the first step:

- the semantics are defined in such a way that the meaning of each structure in the knowledge representation is clear and there is no ambiguity in terminology;
- the terms and definitions are precise;
- the proposed knowledge representation is interpretable by both humans and machines;
- the proposed knowledge representation is suitable for reasoning, and
- the proposed knowledge representation must be suitable for use in a dynamically adaptive operating environment where the product characteristics set the requirements for the resource capabilities.

The domain is defined as being in the domain of part manufacturing and product assembly process modelling. Continuous processes and the production of raw material were excluded from this modelling method. The conceptual Product-Process-System model did experience some ambiguities with the terminology. However, the ambiguities were eliminated when the ontology was being modelled. The definitions are unambiguous and the necessary number of relations between the classes is defined. The ontology itself is generic enough for modelling both assembly processes and manufacturing operations.

The ASDM and MSDM product, feature, and activity graphs can be modelled into the ontology with precedence rules. The product and processes can be modelled on different levels of abstraction. However, the guideline, which is provided in the form of a taxonomy, may not in itself provide enough guidance for instance modelling; nevertheless, it is assumed that the designer possesses basic knowledge in the field of production engineering.

The KR is human- and machine-interpretable. For human interpretation natural language explanations are provided as notations for each class. The ontologies were built utilising the Protégé 3.4 OWL-DL tool [24], illustrated in Figure 5.24. OWL-DL is based on the description logics and allows the definition of the domain concepts mostly according to a predefined formalism. OWL-DL provides enough expressiveness for most of the axioms, while retaining computational completeness and decidability, which facilitate the reasoning procedures for consistency validation. However, since the OWL lacks some definitions, such as *partOf* relation, some shortcuts in the implementation phase were necessary. It is understood that OWL DL might be too light-weight for defining more formal ontologies.

Graphviz [20], as an addition to the Protégé OWL DL ontology tool, allows the rendering of ontology graphs that can help human operators to validate the structure for it to represent the desired view of the domains. The structure allows simple queries to be tested inside Protégé.

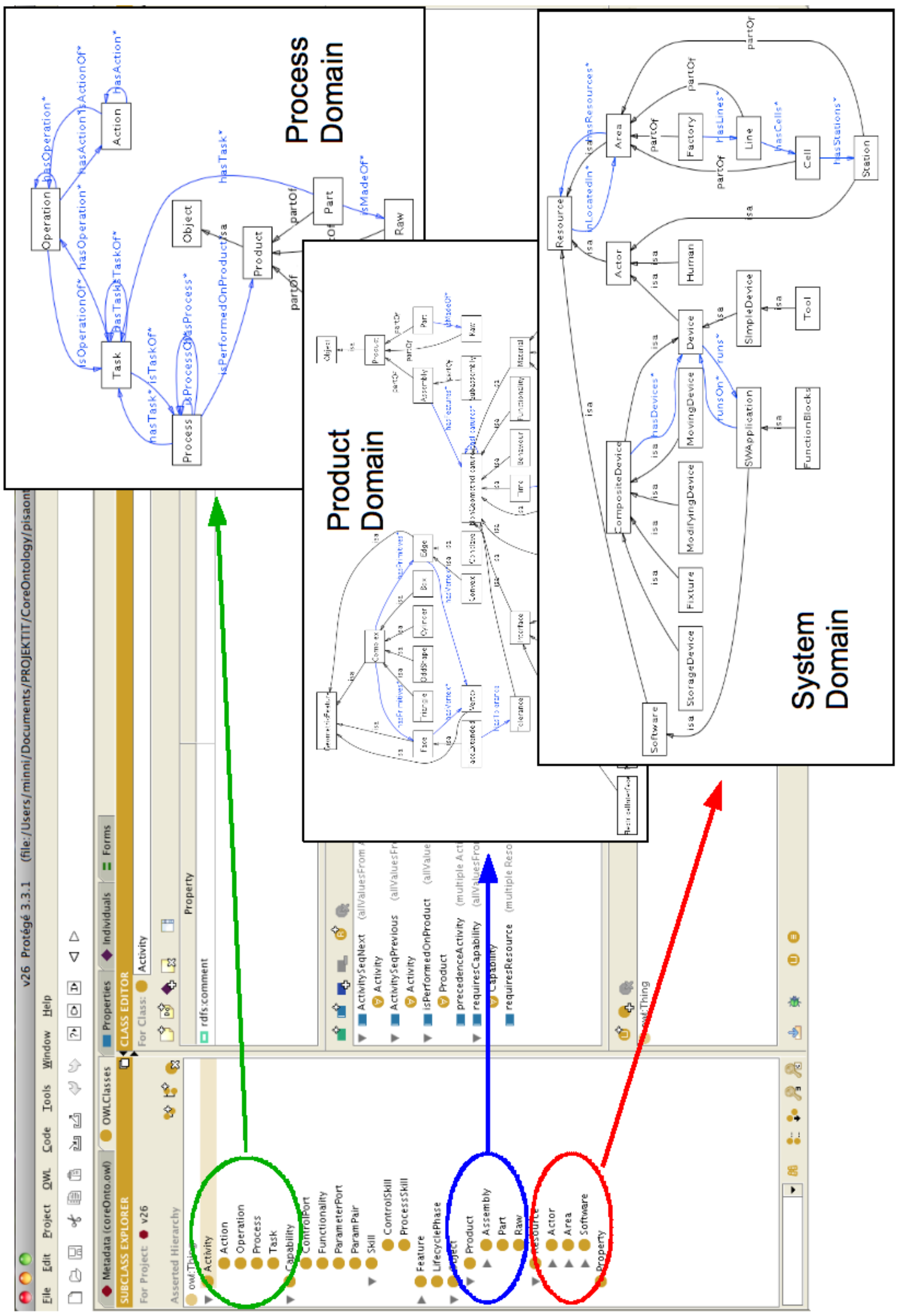


Figure 5.24: A view of Protégé and the product, process, and system domains

Chapter 6

Implementation

This chapter will introduce the implementation of an environment where the Core Ontology is implemented. The aim of the chapter is to introduce technical solutions for building a knowledge base environment where the reasoning and queries are provided upon the knowledge representation that was developed. The Knowledge Base (KB) serves a platform in which the Core Ontology that was developed and its structure and relations can be validated. Chapter Seven will validate the ontology and knowledge base with real case implementations from ongoing projects [1].

6.1 Knowledge Base

According to Grosso et al. [25], a knowledge management system should be a neutral system which is independent of any specific applications. This is indispensable in order to fulfil the objective of seamless interaction between different design support systems. The KB system encapsulates all the specific parts of a control system in order to provide a neutral interface with the application software. The KB system software, which provides the functionality needed to support the openness for the application software, is located on top of the platform. Knowledge acquisition is also accessed by humans, as part of a job that does not require previous experience of knowledge-based systems [25].

The KB has three main goals. The first goal is to act, in a simple case, as

[1] The implementation of the KB environment is part of ongoing academic research projects developed in the Department of Production Engineering at Tampere University of Technology, see [37, 38].

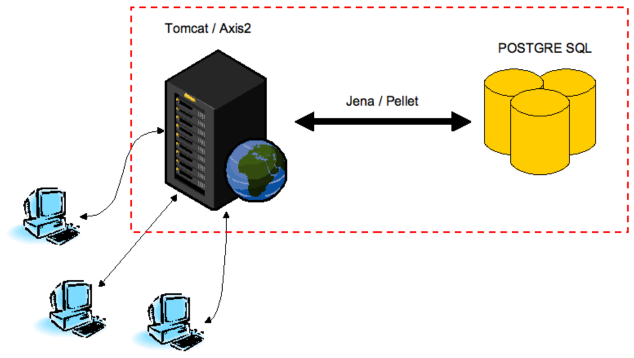


Figure 6.1: The KB and the clients

a PDM (Product Data Management) vault. The second goal is to provide a standardised interface between different design support system clients, as illustrated in Figure 6.1. The clients are able to share the information more efficiently when the native formats are mapped into the proposed knowledge model, the Core Ontology. For the clients the only requirement is to provide the interface to the KB. As mentioned in the introduction, the clients contribute their native knowledge to the common knowledge model while revising and validating the overlapping knowledge from the surrounding systems connected to the KB.

The third goal for the KB is to serve as a globally accessible data storage unit for the design plans and virtual manufacturing. This differs from a PDM system in that once the production has been completed the production control unit will save the real product-specific process values into the digital representation of the product, i.e. the product model; see Figure 6.2.

The conceptual architecture of the knowledge base consists of four access layers, illustrated in Figure 6.3. These layers are intended to create controllable groups, or software modules, according to the responsibilities of each.

The three knowledge domains defined by ontologies, the product, process, and system domains, are implemented into the KB environment. This structure is flexible enough to accept the inclusion of other knowledge domains, such as logistics, supply chain management, order management, and others.

The storage of information for each concept or entity is implemented with XML-based files: XML for data management and X3D (eXtensive 3D, a successor to VRML) for visualisation.This allows the knowledge base to be

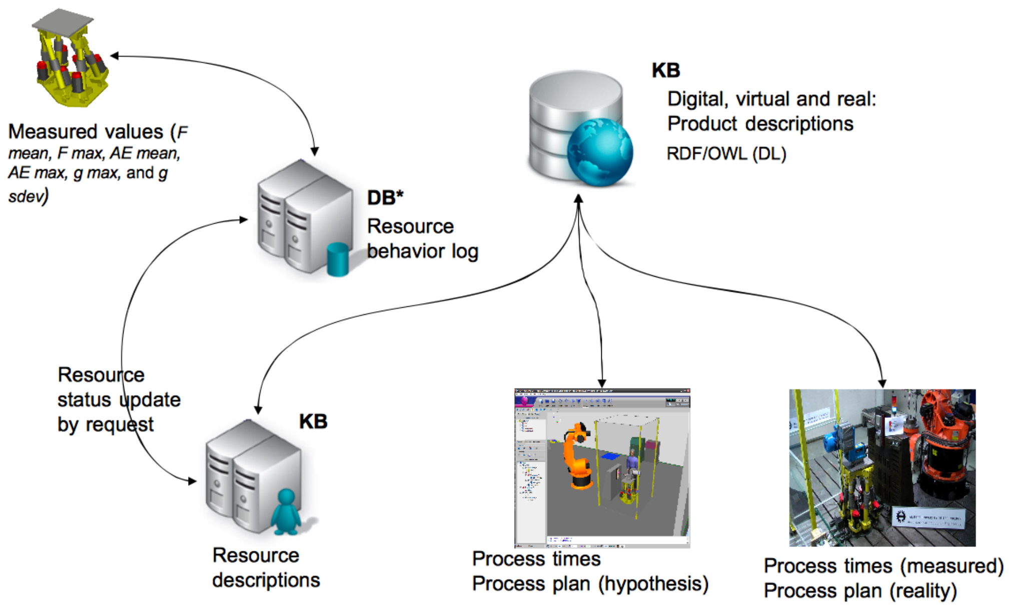

Measured values (F mean, F max, AE mean, AE max, g max, and g sdev)
DB*
Resource behavior log
KB
Digital, virtual and real:
Product descriptions
RDF/OWL (DL)
Resource status update by request
KB
Resource descriptions
Process times
Process plan (hypothesis)
Process times (measured)
Process plan (reality)

Figure 6.2: The KB and the surrounding manufacturing systems

distributed in case there is a need for the collaborative design of products and systems.

The *Service Interface* in the architecture description - Figure 6.3 - provides the necessary interfaces that allow the clients access to the knowledge stored in the KB. The services provided in this layer are *Ontology Services, Ontology Query Services*, and *XML Services*. These focus on the retrieval of information from the KB. The *Ontology Querying Services* were divided into three groups - *Product, Process*, and *System services* - to reflect the ideological structure of the Core Ontology, illustrated in Figure 6.4. The fourth group in the Ontology Querying Services is called the *Model Service Group*, which is intended for file-saving and retrieval operations in the KB environment.

The originally planned X3D and VRML services were planned as a group of services which would handle the specific formats for integrating these into the KB. However, the deeper analysis performed during the development showed that the X3D and VRML content also needed to be included into the Core Ontology. This inclusion was performed through feature-based analysis with Pro-FMA, introduced in the chapter on Cases, which parses the VRML files into X3D and adds basic meaning and values into the surfaces and features that are found and a reference to the corresponding visualisation file (X3D). The Model Service Group handles the references, wether or not those files

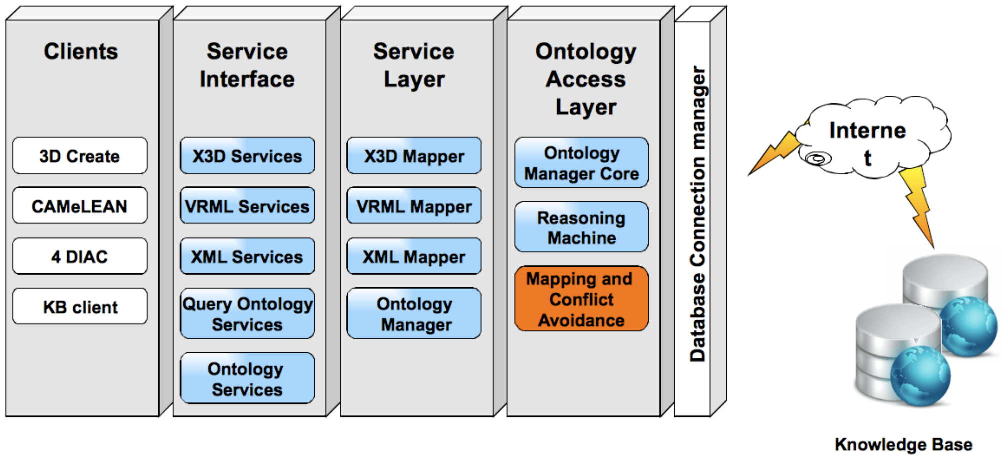


Figure 6.3: Conceptual structure of the KB

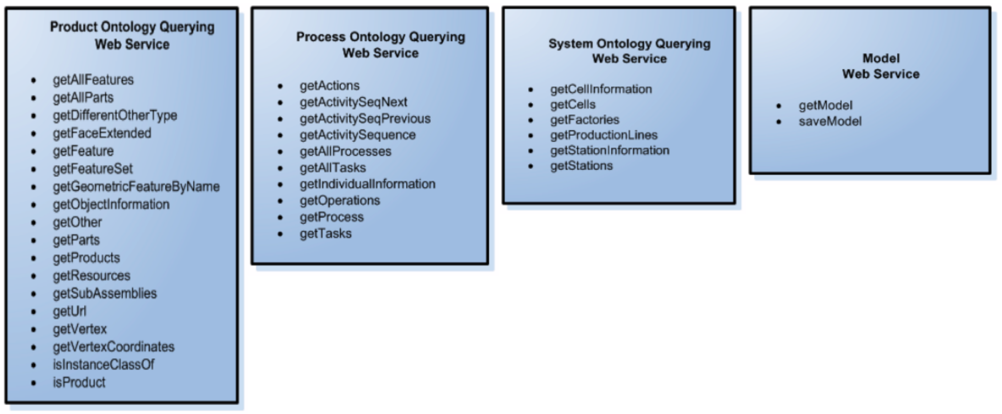


Figure 6.4: Ontology Querying Services for product, process and system knowledge

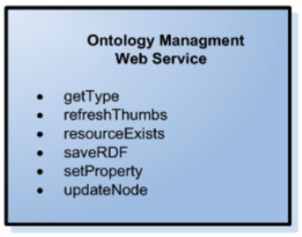


Figure 6.5: Ontology Management Services

have deeper meaning inside the Core Ontology.

The *Service Layer* was originally dedicated to serve as an access layer between the mappers for the different formats in the KB and Ontology Manager. The mapper is the part of the KB that presets the data to the application that requested it in the format required by that application. The current implementation has two mappers, an XML mapper and a JSON (Java Script Object Notation) mapper. The purpose of these mappers is to facilitate the exchange of data between different web applications. The JSON mapper was found to be especially good for the web applications developed in JavaScript, since the KB stores the data in triplets; each of the triplets has a subject, verb, and object. The *Ontology Manager* is responsible for the modification of the content of the Core Ontology, i.e. the triplet values.

The examples of the services used for management are as follows:

- value update of triplet(s);
- deletion of triplet(s);
- insertion of triplet(s);
- querying if a resource exists, and
- querying if the type of an individual exist.

One of the most important services in this group is the function that allows the information to be exported from another application into the KB. By using this web service, a client application can save data directly to the KB with only one format-based restriction; it must obey the RDF/XML specification proposed by W3C.

The *Ontology Access Layer* is the final layer in this architecture. Its main purpose is to facilitate the communication between the architecture and the Structured Query Language (SQL) database. It provides reasoning and conflict avoidance. The reasoning is performed in two different ways in the KB. The most fundamental reasoning is provided through the Core Ontology. The classes have properties and relationships to the surrounding classes, thus forming a basic set of relations and restrictions between stored contents. For example, the *Product* classis related to the subclass *Part* and the relation is a "Product has Part(s)". Every time new data stored in the KB, new instances are created and relationships between contents are created. Since the number of classes and properties and the relationships between them is constant and not modifiable by the users, the KB can define what the relationship between "separate" contents is. This is defined as the most basic level of reasoning in this architecture.

The second type of reasoning provided by the KB is reasoning based on a dedicated reasoner, Pellet [11]. Pellet allows the creation of rules for defining additional relations, or constraints, between classes and facilitates the generation of more complex queries with a relatively simple request. Currently there are only a few queries implemented with Pellet, but the chapter on Future Work will identify more to come.

The third function of this layer is to provide conflict avoidance. The original plan of the KB environment was to design it as a distributed system, where the multiple partners would access the system from different places at the same time. This would naturally have required protocols for conflict avoidance. Because of the use case scenarios provided by the Piirre2.0 - Chapter 7, Case 1- and PiSA - Chapter 7, Case 2 - projects, the distribution of the KB environment was categorised as a low priority and left as a "possible distant future developments" plan. Currently, the conflict avoidance module is focused on the optimum use of the framework in order to speed up the queries and reliability of the KB.

The technical implementation of the KB is a combination of several existing technologies:

- Apache 2 web services engine
- Jena semantic web framework
- Pellet reasoner
- Postgre (SQL) database

- Core Ontology created with Protégé OWL DL ontology tool

6.2 Product, Process, and System Knowledge Acquisition

The query process starts when a client application requests certain information from the KB. In order to access the KB, a Web Services interface must be available through an internet connection. At first a client sends a request to the Web Service interface; see Figure 6.6. If the request is valid, the web service then calls an instance of the Ontology Manager to handle the request. If the request is invalid, a response indicating a communication error or failure is sent to the client. The Ontology Manager creates a query on the basis of the request that has been received and passes it forward to the *Reasoning Machine.*

The Reasoning Machine interacts with the ontology model, which is stored in the Postgre database, in order to find results that match the query. After checking the ontology the Reasoning Machine returns all the results found to the Ontology Manager. The Ontology Manager then decomposes the returned set of results and copies each result to an array. When all the results are included in the array, the array is sent to the *Format Mapper.*

The Format Mapper gets the array of results and converts it to the desired format, XML or JSON, depending on the request. This converted data are returned to the Web Services, such as WDSL (Web Services Description Language), which, finally, is in charge of sending back the response to the client.

6.2.1 An Interface to the Core Ontology and Access to the KB

The knowledge stored in the KB can be accessed through designated middleware using interfaces. Figure 6.7lists the get() functions that are defined at the moment. For the clients who wish to utilise this environment, an automatically updated list of available services is provided through a website.

The common interface of the KB for all clients has the function of providing communication between the knowledge base and the client. The clients need to provide their own parser for saving or querying the KB. An example

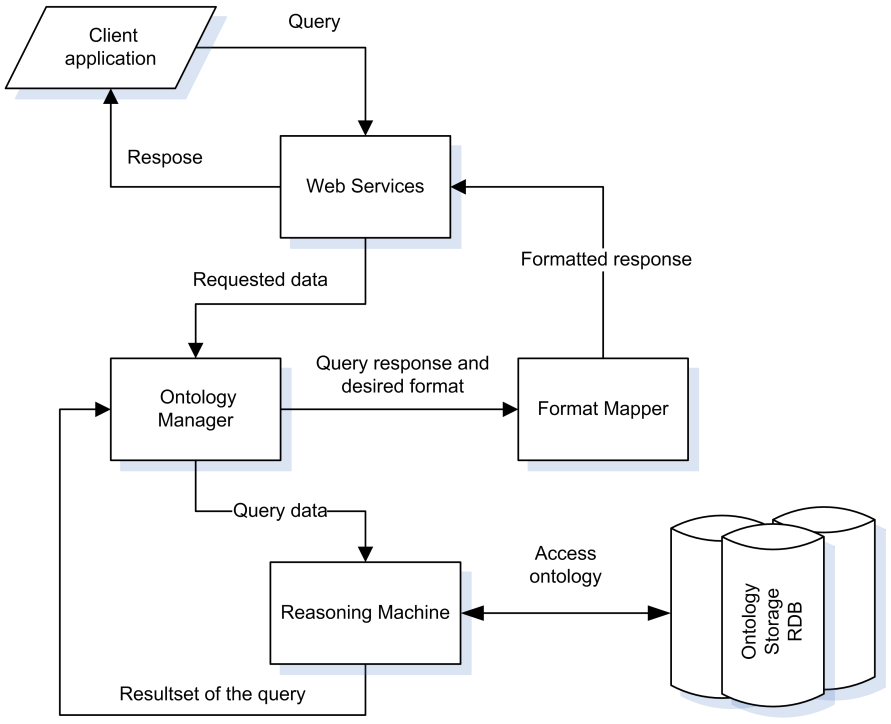
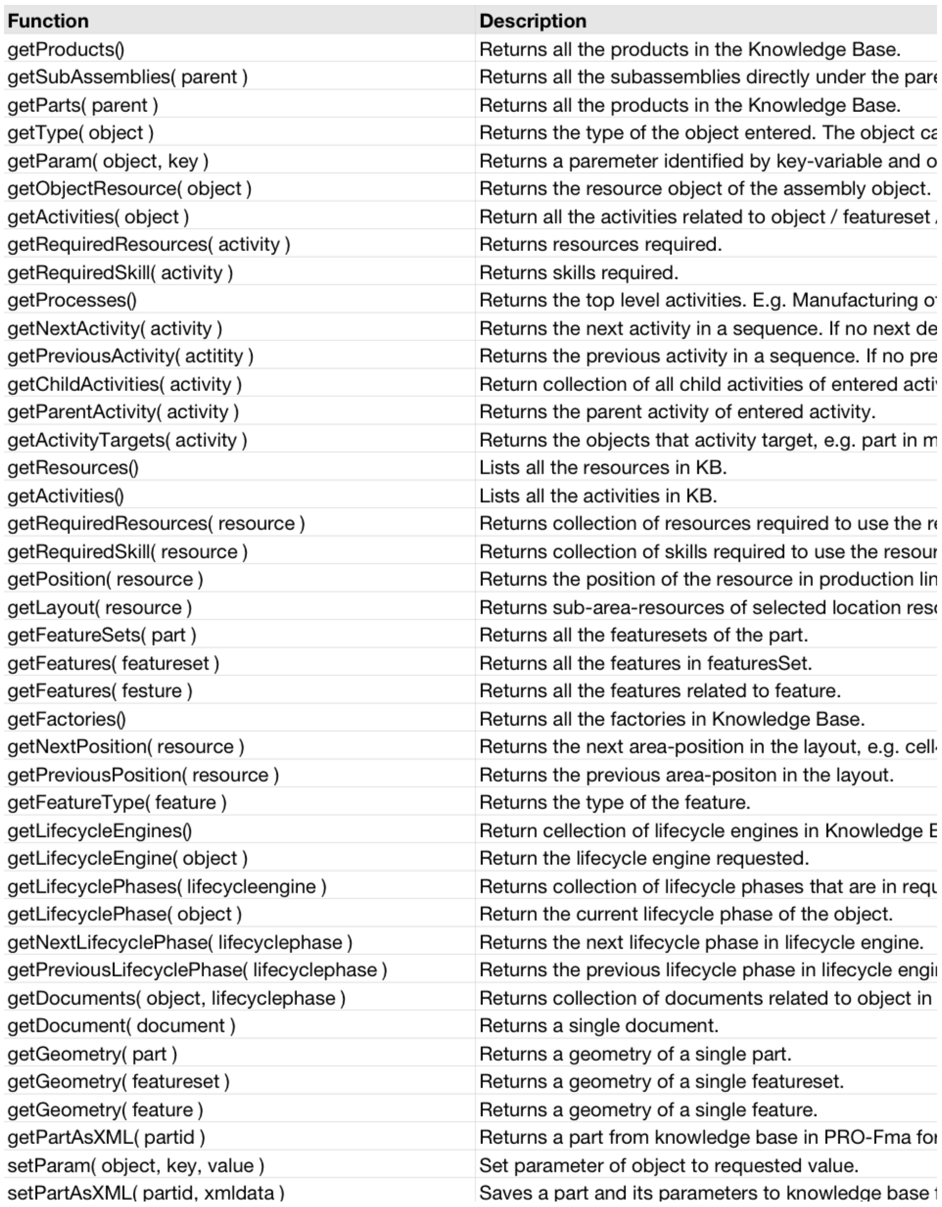


Figure 6.6: Illustration of the query process

of this could be an interface reading XML files sent through HTTP or an interface providing output as native binary format and taking Web Service requests as input. Alternatively, even a COM interface could be supported, thus providing tighter integration with the clients.

For the clients it is also a requirement that they follow the generic guidelines, represented in Chapter 5, for saving their specific information to the KB. The KB interface, together with the client's interface, provides seamless translation from the scope of a specific domain to the scope of the knowledge model and vice versa. This is required in order for the knowledge base not to grow into an uncontrollable mass of redundant data, but instead to maintain a semantic database containing rules for translating terminology and semantics.

For the clients it is a set requirement that they follow the generic guidelines, represented in Chapter 5, for saving their specific information to the KB. The KB interface, together with the client's interface, provides seamless translation from the scope of a specific domain to the scope of the knowledge model and vice versa. This is required in order for the KB not to grow into an



Function	Description
getProducts()	Returns all the products in the Knowledge Base.
getSubAssemblies(parent)	Returns all the subassemblies directly under the pare
getParts(parent)	Returns all the products in the Knowledge Base.
getType(object)	Returns the type of the object entered. The object ca
getParam(object, key)	Returns a paremeter identified by key-variable and o
getObjectResource(object)	Returns the resource object of the assembly object.
getActivities(object)	Return all the activities related to object / featureset
getRequiredResources(activity)	Returns resources required.
getRequiredSkill(activity)	Returns skills required.
getProcesses()	Returns the top level activities. E.g. Manufacturing o
getNextActivity(activity)	Returns the next activity in a sequence. If no next de
getPreviousActivity(actitity)	Returns the previous activity in a sequence. If no pre
getChildActivities(activity)	Return collection of all child activities of entered acti
getParentActivity(activity)	Returns the parent activity of entered activity.
getActivityTargets(activity)	Returns the objects that activity target, e.g. part in m
getResources()	Lists all the resources in KB.
getActivities()	Lists all the activities in KB.
getRequiredResources(resource)	Returns collection of resources required to use the r
getRequiredSkill(resource)	Returns collection of skills required to use the resour
getPosition(resource)	Returns the position of the resource in production lin
getLayout(resource)	Returns sub-area-resources of selected location res
getFeatureSets(part)	Returns all the featuresets of the part.
getFeatures(featureset)	Returns all the features in featuresSet.
getFeatures(festure)	Returns all the features related to feature.
getFactories()	Returns all the factories in Knowledge Base.
getNextPosition(resource)	Returns the next area-position in the layout, e.g. cell
getPreviousPosition(resource)	Returns the previous area-positon in the layout.
getFeatureType(feature)	Returns the type of the feature.
getLifecycleEngines()	Return cellection of lifecycle engines in Knowledge E
getLifecycleEngine(object)	Return the lifecycle engine requested.
getLifecyclePhases(lifecycleengine)	Returns collection of lifecycle phases that are in requ
getLifecyclePhase(object)	Return the current lifecycle phase of the object.
getNextLifecyclePhase(lifecyclephase)	Returns the next lifecycle phase in lifecycle engine.
getPreviousLifecyclePhase(lifecyclephase)	Returns the previous lifecycle phase in lifecycle engi
getDocuments(object, lifecyclephase)	Returns collection of documents related to object in
getDocument(document)	Returns a single document.
getGeometry(part)	Returns a geometry of a single part.
getGeometry(featureset)	Returns a geometry of a single featureset.
getGeometry(feature)	Returns a geometry of a single feature.
getPartAsXML(partid)	Returns a part from knowledge base in PRO-Fma fo
setParam(object, key, value)	Set parameter of object to requested value.
setPartAsXML(partid, xmldata)	Saves a part and its parameters to knowledge base

Figure 6.7: Get() functions of the interface

uncontrollable mass of redundant data, but instead to maintain a semantic database containing rules for translating terminology and semantics. Most of the reasoning is done inside the clients, but in the future some reasoning processes can be implemented in the knowledge base interface. These reasoning services inside a client include, for example, a scheduling agent inside a simulation tool that utilises simple queries first to access the knowledge inside this KB.

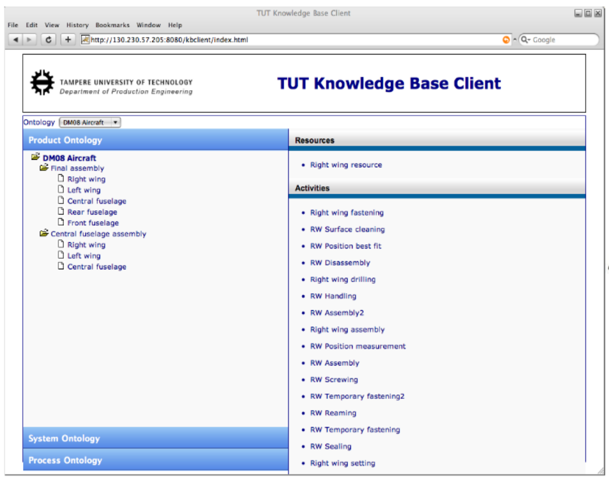
In this approach the knowledge stored in the KB is accessed through a Web Services interface using SOAP. The Web Services interface is composed of two kinds of methods, as mentioned earlier: fundamental methods and reasoning methods. The fundamental methods have the function of retrieving or posting information to the KB. The retrieval methods request data from the KB. These requests are responded to with information about an object or several objects in the desired format (XML, JSON, etc). As an example a client application can request a list of products available in the KB in XML format. If the request cannot be met, the response of the Web Service will be "no data available".

Posting methods are meant to receive data sent by clients to be stored in the KB. This method can propose an update to actual information on the instance level or totally new data to be added to the KB. The posting method does not modify the basic class structure. A confirmation message that a successful transaction has taken place is sent to the client at the end of the operation. The reasoning methods are meant to support the client's activities by giving access to already-reasoned information. These methods are mainly for retrieving information and will be defined further in the near future.

The advantage of a Web Services interface is that the client does not need to be tightly attached to the KB. If a client is interested in any of the Web Services offered by the KB, it just has to request the list of services via the web. A WSDL (Web Services Description Language) XML document, where all the interface methods and parameters are also available, is available online; this allows client applications to be developed fast and accurately.

6.2.2 Knowledge Base Web Client

In order to inspect the knowledge stored in the KB, a viewing client was developed; see Figure 6.8. As mentioned earlier, the client uses Web Services to get the product information. The client automatically queries and



TUT Knowledge Base Client

File Edit View History Bookmarks Window Help

http://130.230.57.205:8080/kbclient/index.html

TAMPERE UNIVERSITY OF TECHNOLOGY
Department of Production Engineering

TUT Knowledge Base Client

Ontology: DM08 Aircraft

Product Ontology

- DM08 Aircraft
  - Final assembly
    - Right wing
    - Left wing
    - Central fuselage
    - Rear fuselage
    - Front fuselage
  - Central fuselage assembly
    - Right wing
    - Left wing
    - Central fuselage

System Ontology

Process Ontology

Resources

- Right wing resource

Activities

- Right wing fastening
- RW Surface cleaning
- RW Position best fit
- RW Disassembly
- Right wing drilling
- RW Handling
- RW Assembly2
- Right wing assembly
- RW Position measurement
- RW Assembly
- RW Screwing
- RW Temporary fastening2
- RW Reaming
- RW Temporary fastening
- RW Sealing
- Right wing setting

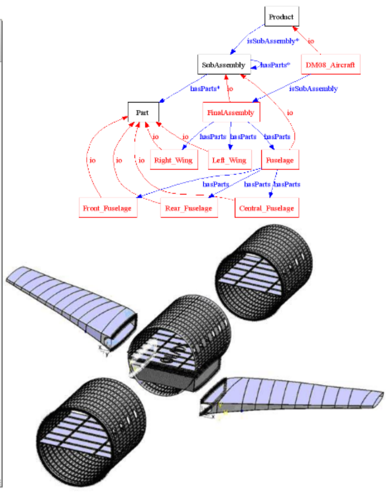


Figure 6.8: Knowledge Base's Web Client

divides the information into product-, process-, and system-specific knowledge sets. By selecting the product ontology, the user will see the product structure, sub-assembly, and parts, and on the right-hand side he/she can immediately see the activities related to each part or subassembly or product.

For example, if the user browses the part structure, the client will run the search query for the operations that have possibly been saved. By selecting from the client's right-hand side and clicking the operation that is shown, the client will query whether or not a dedicated resource for completing the operation exists. If the user is interested only in the activities he/she can click the process domain from the client's left-hand side area and the client will retrieve a process, task, operation, and action list for the user.

At the moment a functionality for reading and modifying existing information is being implemented. In future this client will also be able to modify the stored information and create new instances. In order to allow the user to explore the data stored in the KB, each instance set can be retrieved for inspection.

Chapter 7

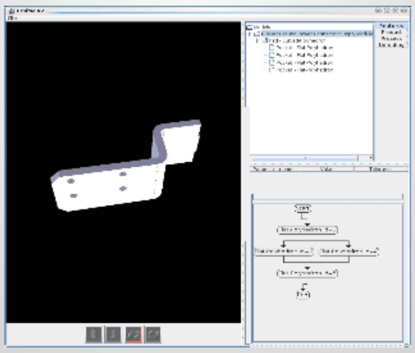
Case Studies

The case studies evaluate the feasibility of the chosen modelling approach. The Knowledge Representation (KR) is evaluated from several points of view:

- the expressiveness of the model;
- how the details can be modelled;
- how the higher level of abstraction lis handled with the KR;
- how the relations perform for reasoning purposes;
- whether the KR supports knowledge mapping between different information models, and
- whether the KR supports knowledge acquisition from the operating environment.

Figure 7.1 summarises the case studies and aims to evaluate the KR. The selected case studies evaluate different viewpoints. The first case study focuses on the detail level. The geometrical and non-geometrical features are handled with the KR, meaning that the product model is modelled on the feature level, including the position of the face vertices and the corresponding visualisation. The second case study evaluates knowledge mapping between selected tools. The tools include process planning, simulation, and factory floor control design tools. The main idea is to utilise the KR as the master model that is updated by the clients. The third case study evaluates the KR from the system point of view. The product model includes the geometric and non-geometric values that are utilised for defining the operations, mainly drilling. Once the manufacturing of the part is completed, the operational values are collected and saved for the part history.

Case 1: Feature Recognition



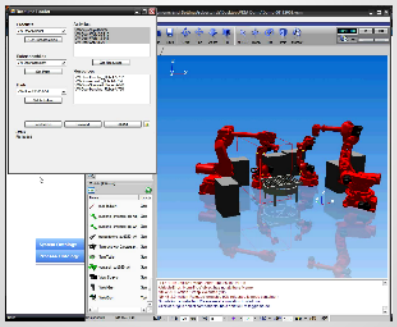
The aim of case one was to evaluate the formed knowledge representation from the manufacturing process perspective.

MSDM model for product and process hierarchy

Geometrical and Non-geometrical features and connections to the processes

KR in RDF and visualization in X3D (KR is capable to represent 2D and 3D geometry)

Case 2: Integration of Design Toolset



The main aim of the case 2 was to evaluate the KR in system level

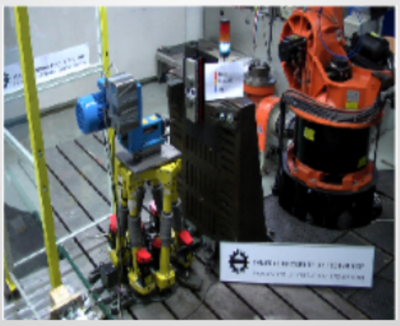
Knowledge share between different tools (commercial tools) via common Knowledge Representation (Core Ontology)

Resource descriptions (simplified level)

Process planning, simulation and control as contributors/users

Each client has middleware to ensure the correct input form

Case 3: Adaptive Operations Environment



The case 3 evaluated the KR from the operational environments viewpoint by adding the operational requirements to the model

Reasoning is based on the knowledge representation (Core Ontology)

Resource properties are described through CoreOntology

Different Manufacturing Scenarios are saved to KB (digital, virtual and real)

The model is updated along the way via real operational parameters

Figure 7.1: Summary of the case studies performed in order to evaluate the expressiveness and feasibility of the KR

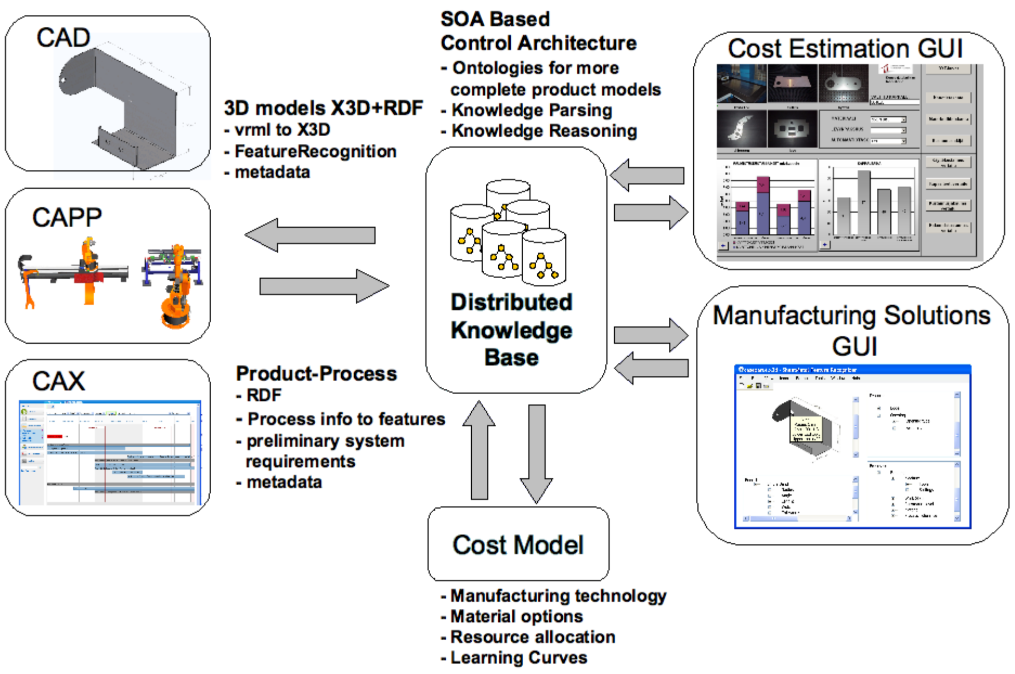


Figure 7.2: Features connected to manufacturing costs

7.1 Case Study 1 - Feature Recognition

The aim of Case Study One is to evaluate the knowledge representation that is formed from the manufacturing process perspective. The manufacturing features are recognised and the preliminary processes are connected to them. The proposed knowledge representation must be able to save the feature information and provide relations at the correct level of abstraction.

Piirre2.0 project's description and needs for a formal KR

The case described here belongs to the project called Piirre2.0 - Utilisation of features in the cost analysis of the manufacture of sheet metal products. Technically, the project aims to improve the knowledge of the manufacturing methodologies of the designer of the sheet metal products, improve the sheet metal manufacturing processes, and facilitate knowledge sharing via open formats and knowledge reasoning. The business goals of the project were to improve the sheet metal product manufacturing know-how, share the know-how inside the company, and the overall competence of Finnish industry.

In order to fulfil the project's technical requirements, the product feature

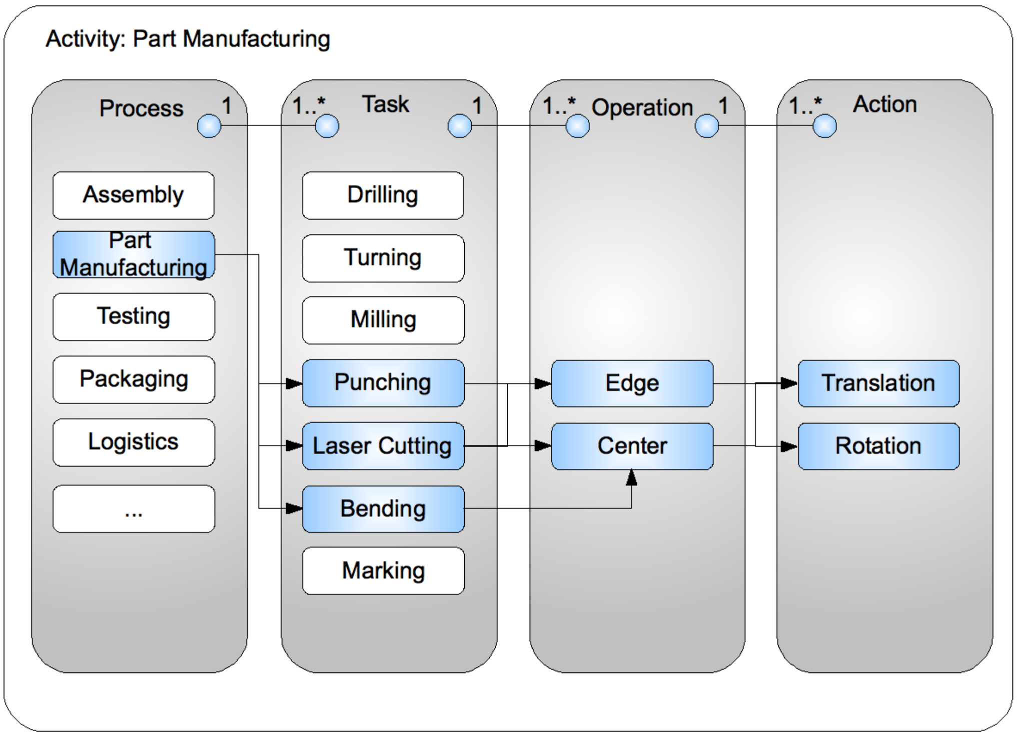


Figure 7.3: Manufacturing Process Levels

recognition, feature classification, and manufacturing state decomposition models are incorporated. Product knowledge is seen as a combination of product-specific information, such as functionality, material and topology, and the corresponding product model. This knowledge includes geometric and non-geometric information, which is represented through features. The features are the foundation elements of the knowledge representation, which can be used for analysis and knowledge acquisition. An approach to deriving the essential manufacturing-specific knowledge from the product model is proposed for use for cost allocation, process planning, and production system definition.

The scope of the project is illustrated in Figure 7.2. On the basis of the classification and identification of possible manufacturing features and corresponding processes, the system requirements and automation levels can be defined. On the cost modelling side, the chosen system, batch size, and learning curves will affect the total costs of a product. By combining the feature-based analysis with advanced cost modelling, there is a possibility of tracing the costs according to the features that are designed. The Core

Ontology is used as the common knowledge exchange format and the KB as the platform for facilitating the knowledge share; see Figure 7.2.

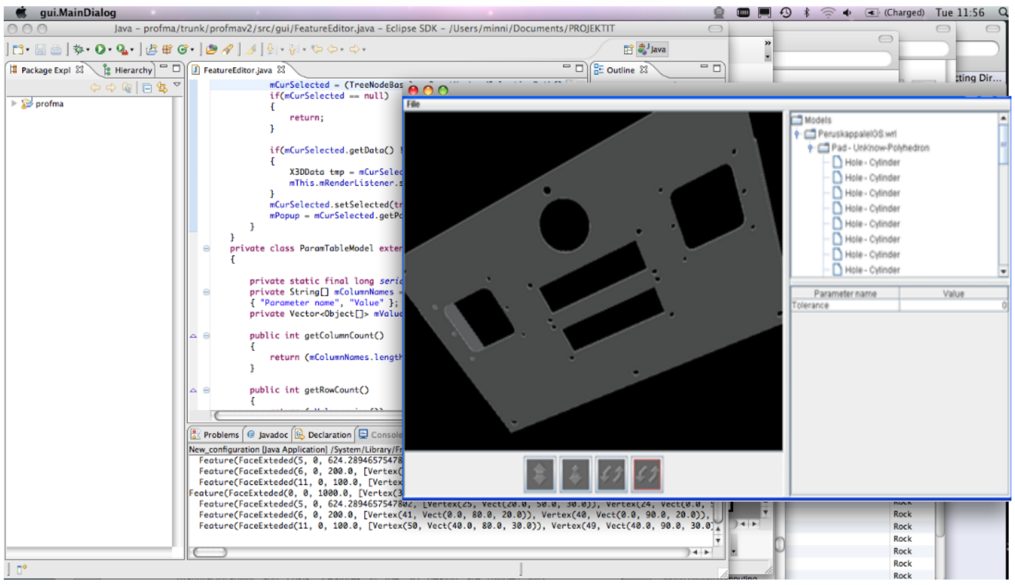
Manufacturing Features, Corresponding Processes, and the Core Ontology

As the feature-based model was developed in order to fill the gap between detailed geometry information, the elementary relations, and abstract functional information. Features in the 3D design are divided into geometric and non-geometric features that have an impact on the processes used to manufacture the desired product forms. For example, the tolerance requirements can affect the tool selection and, further on, the selection of actual manufacturing systems.

Manufacturing processes are described with a manufacturing taxonomy; see Figure 7.3. The taxonomy illustrates the levels of operations in the KR that is used. It has to be noted that at the moment there is no validity check in the ontology that would follow the taxonomy proposed here. However, the parsers used between a design tool and the Knowledge Base KB) follow this guideline. The process ontology is the key operator when the product and the system knowledge are being combined. Activities are defined on the basis of the theoretical background as Part Manufacturing, Assembly, Testing, and Packaging.

In the model used here, the manufacturing-specific feature information is divided into four layers: *Features, Feature Sets, Manufacturing Features*, and *Manufacturing Feature Sets*. The features and feature sets are results of the reasoning and recognition based on pre-defined features. The manufacturing features and feature sets are application-specific reasoning results, for example the group of four similar holes in the sheet metal product.

A feature can be a certain face or an edge. The feature sets are more complex combinations of recognised and meaningful faces. The manufacturing features include processing and system requirements, which in this case are preliminary requirements for the tooling that are based on the dimensional values of the features and grouping of the similar features into process groups.



```
            mCurSelected = (TreeNodeBas
            if(mCurSelected == null)
            {
                return;
            }

            if(mCurSelected.getData() !
            {
                X3DData tmp = mCurSelec
                mThis.mRenderListener.s
            }
            mCurSelected.setSelected(tr
            mPopup = mCurSelected.getPo
        }
    }
    private class ParamTableModel exter
    {

        private static final long seria
        private String[] mColumnNames =
        { "Parameter name", "Value" };
        private Vector<Object[]> mValue

        public int getColumnCount()
        {
            return (mColumnNames.length
        }

        public int getRowCount()
        {
```

```
New_configuration [Java Application] /System/Library/Fr
  Feature(FaceExteded(5, 0, 624.28946575478
  Feature(FaceExteded(6, 0, 200.0, [Vertex(
  Feature(FaceExteded(11, 0, 100.0, [Vertex
Feature(FaceExteded(0, 0, 1000.0, [Vertex(3
  Feature(FaceExteded(5, 0, 624.2894657547802, [Vertex(25, Vect(20.0, 50.0, 30.0)), Vertex(24, Vect(0.0,
  Feature(FaceExteded(6, 0, 200.0, [Vertex(41, Vect(0.0, 80.0, 20.0)), Vertex(40, Vect(0.0, 90.0, 20.0)),
  Feature(FaceExteded(11, 0, 100.0, [Vertex(50, Vect(40.0, 80.0, 30.0)), Vertex(49, Vect(40.0, 90.0, 30.0)
```

Figure 7.4: Pro-FMA

Feature Recognition tool, Pro-FMA

In order to add meaning to the models as described earlier, a feature recognition software suite was developed in the Piirre2.0 project. Pro-FMA is a tool for re-generating the features lost during the export process from a CAD system. Previously, some ad hoc developments were planned to be implemented inside a specific CAD system. However, it turned out to be too much work for too few achievements. Better results are achieved when a platform-independent Java application is used. This allows more freedom not only from the operating system, but also from a CAD system, since no matter what the system, the quality of information is still low. For exporting 3D geometry from a CAD system, VRML (virtual reality modelling language) is used. VRML provides the lowest possible 3D geometry. Depending on the authoring system used, the conversion result might just be a group of disconnected faces and vertices. The VRML file is parsed into X3D (eXtensive 3D, a successor to VRML) and geometric features such as holes are defined [39].

Figure 7.4 shows the GUI (Graphical User Interface) of the feature recognition tool. On the right-hand side of the figure, the pre-defined features are listed. In the bottom right-hand corner, the feature-specific non-geometrical parameters, such as dimensions and tolerances, are shown.

There are some special features that are interesting from the perspective of the manufacturing of the sheet metal product. These must be recognised and categorised as such. In the current implementation, there are seven different kinds of predefined features: cylinder, box, filled rectangle, semi-cone, countersunk, counter-drilled, and half-cylinder. The Core Ontology has a class structure for saving the recognised feature information as parametric information, such as type of feature, and dimensional properties that are found, including diameter, height, width, and length. Properties that have numerical values are categorised under the non-geometric feature class. Pro-FMA uses heuristics based on vertex positions, connecting edges, and the number of edges to classify the features that are found. If a feature does not fit into any of the predefined classifications, it is defined as a complex shape feature. For more information on the feature recognition process, see Lanz et. al. [39].

Most of the predefined features are parameterised in such a way that the topology matters and not the distances between edges or vertices. If the feature is not recognised by its topology, then the user should add the necessary parameters.

The manufacturing processes are described through the Manufacturing State Decomposition Model (MSDM), introduced in Chapter 5. The MSDM captures the preliminary process definitions for the features that are found. For the sheet metal processes for material removal (by cutting, punching, or by laser) or bending, the reasoning of the process requirements is not too complex. The reasoning becomes complicated once the batch size is known and products are nested and organised into queues. However, the aim of this case is not to go that far.

Pro-FMA and Connection to the KB

The connection with the knowledge base takes place through the middleware, which offers services and functions that third-party programs such as Pro-FMA can use. Using these functions, Pro-FMA inserts features and and corresponding parameters into the knowledge base and also retrieves the knowledge in the same manner.

Pro-FMA inserts data by generating an RDF file that follows the ontology specification that has been defined for this purpose. The SaveRDF() function is invoked to save the data; see Figure 7.6. The parameters, the RDF data, are sent with SOAP to the middleware, which translates the RDF information into the KB language. The KB uses a traditional relational database

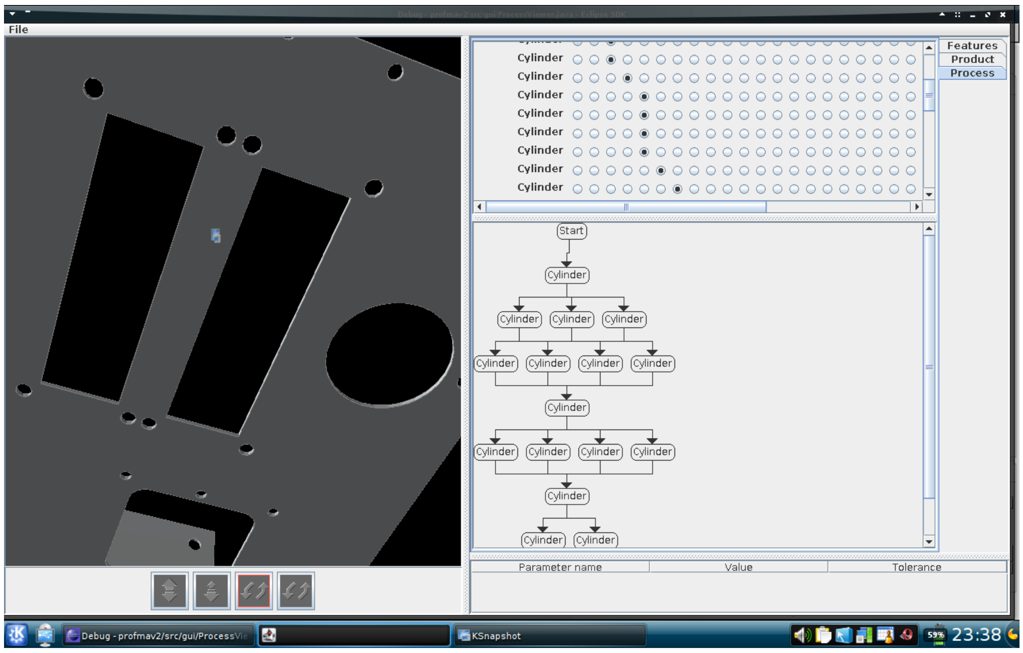


Figure 7.5: Pro-FMA Process Editor

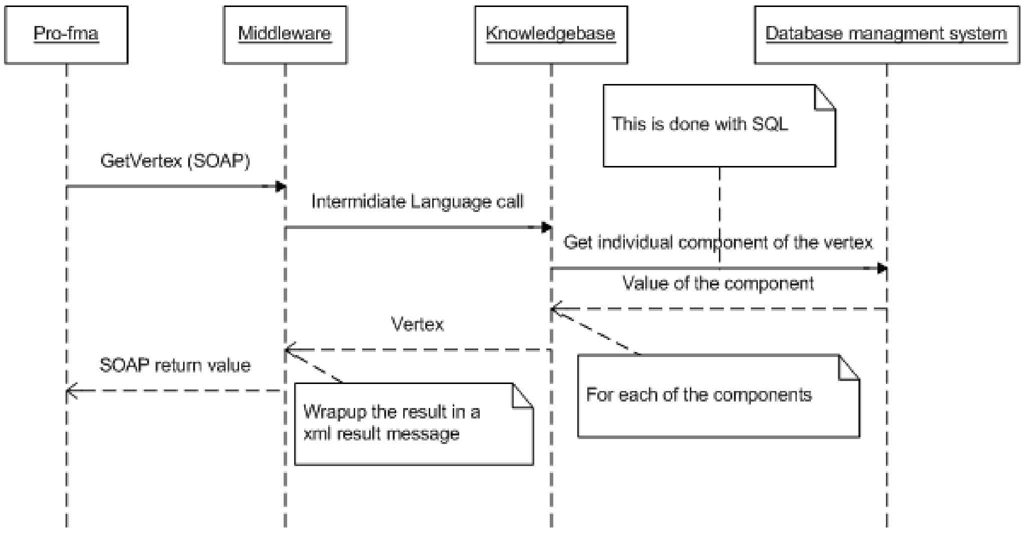

Pro-fma
Middleware
Knowledgebase
Database managment system
This is done with SQL
GetVertex (SOAP)
Intermidiate Language call
Get individual component of the vertex
Value of the component
Vertex
SOAP return value
Wrapup the result in a xml result message
For each of the components

Figure 7.6: Sequence graph of connection between Pro-FMA and Knowledge Base

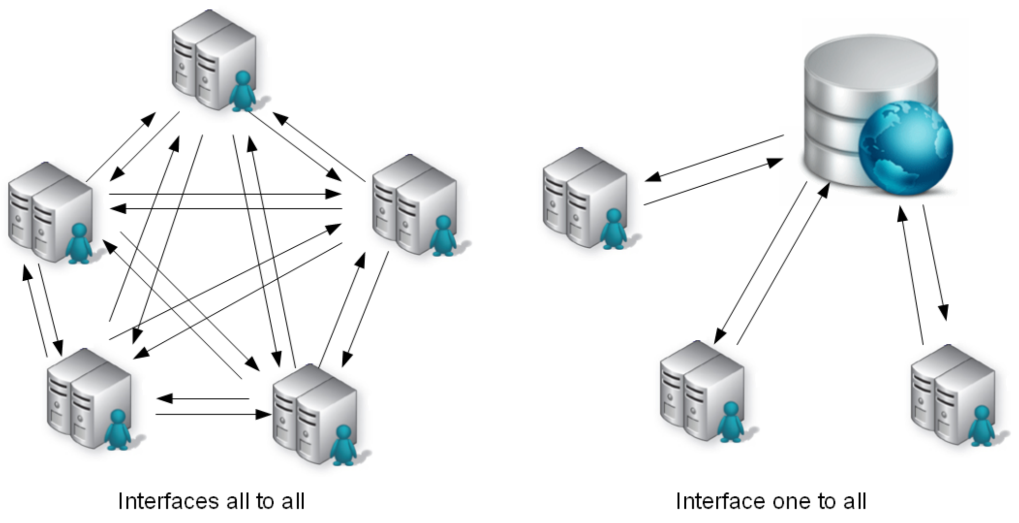


Figure 7.7: Knowledge exchange between clients

at its core and during the final step the KB uses SQL to actually save the information to the DB system.

Summary of the Case Study 1

The aim of the case was to test how the recognised features can be saved as instances into the KR. The file is saved in two parts: RDF for describing the meaning of the model and X3D (eXtensive 3D) for visualisation purposes. The case evaluated the expressiveness of the KR and it was found that the feature level is sufficient for expressing detail-level information. It has to be noted that the KR itself does not generate the meaning for established manufacturing sets, such as four similar holes. The higher semantical meaning must be defined in a reasoning tool, such as Pro-FMA, and the result can then be saved into the ontology.

7.2 Case Study 2 - Semantic Mapping between Different Systems

The second case study aims to evaluate the definitions of the core components of the knowledge representation. In this case, there are several different design tools that share very few semantics with each other. However, they do need to exchange and verify shared data. This case was performed as a part of the Framework Programme 6 IP-PiSA: Flexible Assembly Systems through Workplace sharing and Time-sharing Human-Machine Cooperation project.

Interoperability was the ultimate goal of design and planning systems. It enables the sharing and exchange of product data, information, and, finally, knowledge to take place amongst various software tools within a product development and production planning environment. With the definition of the common representation, this thesis also aims to introduce one solution for enhancing the collaboration between different design systems. The solution is a common ontology and semantic mappings between existing information structures. Figure 7.7 illustrates the current integration method and the proposed method for the exchange of design information.

As mentioned in the Introduction, the knowledge is exchanged between two or more systems by dedicated interfaces, or the information is stripped into basic geometry, or, in the worst case, transferred manually. When the company's design tool platform includes up to a hundred different design tools and software packages, point-to-point integration between all of the systems becomes quite impossible. The amount of re-designed models increases and the amount of manual work becomes too much for a designer to handle and thus makes him/her error-prone. This leads to a situation where the design is done with strongly filtered snapshot information.

The right-hand side of Figure 7.7 illustrates the other option for information exchange by utilising a common information model accessed through one interface. In this way, all of the overlapping design systems can contribute and retrieve their knowledge into a common knowledge model. In this case the focus is on the contribution of knowledge, rather than filtering and re-creating it. The systems here naturally cannot utilise all of the information fully, but they can use what they need and consider the remaining information as metadata. Metadata can be accessed and interpreted by humans as well.

The second case study inspects and tests the ontologies that were developed as a knowledge exchange medium between the design systems, as illustrated in Figure 7.8. In this case, the process plan of the case product is first created in the process planning tool, CAMeLEAN from Ranal Group Ltd. The second client using and verifying the process plans is a simulation tool, 3D Create from Visual Components. The simulation environment will retrieve the process plan and the resources that are needed from the KB, and the pro-

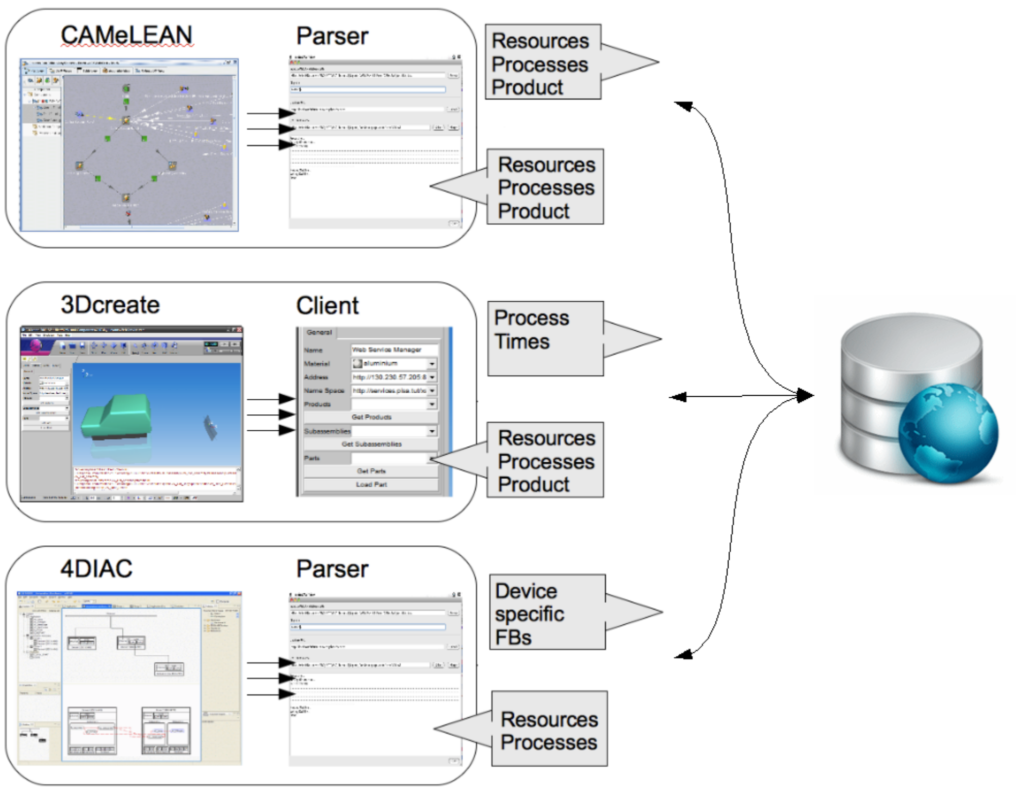


Figure 7.8: Ontologies and KB as knowledge exchange media in PiSA advanced design, planning and optimization toolset

cessing times and resource selection are validated. The verified or updated process time will be uploaded to the KR. The third partner is a control design tool, which utilises the resource descriptions defined by CAMeLEAN and verified by 3D Create. The following sections will go through each client's connection to the KB.

Knowledge Mapping between CAMeLEAN and Core Ontology

Knowledge Mapping between CAMeLEAN and the Core Ontology The planning of manufacturing processes is dedicated to the processes on the shop floor. Process planning software describes the processes at a work station and balances all the processes between the work stations of a complete production line. The processes generate or manipulate parts of the final product or the product itself. They have cycle times, predecessors, successors, and

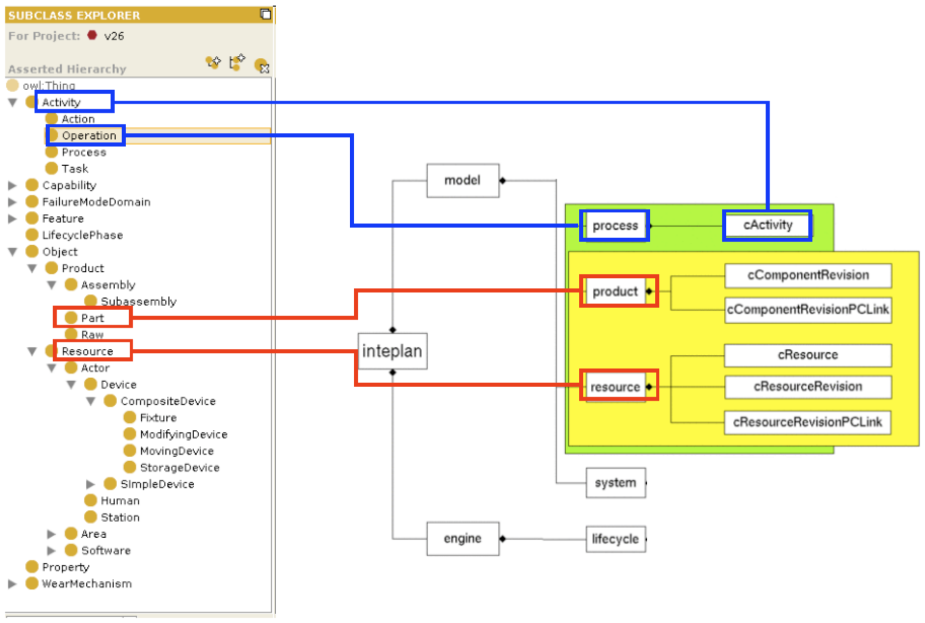


Figure 7.9: Extract of potential connections between classes of CAMeLEAN and the Knowledge Base

assigned resources [50].

Similarly to the parts of the product, resources also have geometric and non-geometric attributes. Examples of non-geometric attributes are life-cycle states and costs. Depending on the needs of the individual company, additional attributes should be includable. Humans, who work in and along with these processes, are only seen as a special type of resource. The activities of the shop-floor humans in the process plan are related to the cycle times of the manufacturing processes. Process planning software deals mainly with the non-geometric data and uses internal or external viewers or CAD software for the visualisation of the geometric data. Therefore, the path for the location of the geometric data of each process planning object is another important non-geometric attribute for all of those objects which own geometric data [50].

CAMeLEAN runs on an XML database, which allows a loose connection to the KB be implemented. Each of the objects is created as an independent file and one to many links are created among these files. Each object file is also a human-readable XML file, but is zip-compressed [38].

The connection between CAMeLEAN and the KB is the most important part in this case study, since CAMeLEAN is the first tool to contribute instances to the KB. An XML parser was created to translate the CAMeLEAN data structures into the form of the Core Ontology. The operational steps for the parser are the following:

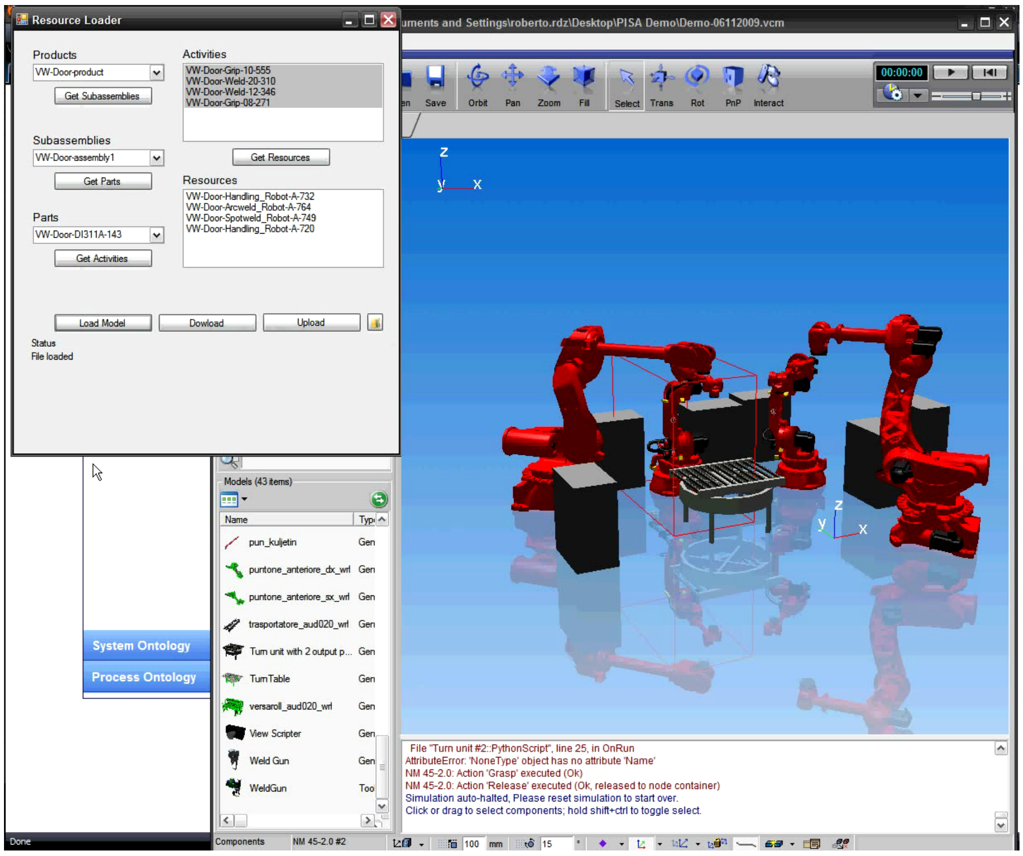
- parse through directory structure,
- decompress the zml files with zlib to xml files,
- parse file with XML dom parser,
- build dependency tree from xml files,
- add values to the nodes, and
- save data to the KB in the form of RDF.

Figure 7.9 shows a few example links between the Core Ontology and potential similar classes in CAMeLEAN. The classes in the green and yellow areas are the main content of the *Process Plan* in CAMeLEAN. The classes of the lifecycle engine are related to those, but not shown in Figure 7.9 [38].

Different from the classes in the yellow area, the classes in the green area posses 2D-geometric attributes, which are necessary to build PERT diagrams of the Process Plan. These require additional classes for representing 2D-geometry; however, those are not shown in Figure 7.9.

Knowledge Exchange Between 3Dcreate and KB

The second step in this case study is to link the Core Ontology straight to the simulation tool that is used. The idea is to utilise the ontological product and process definitions to populate the simulation environment automatically. Currently, the client, illustrated in Figure 7.10, is implemented inside the Visual Component's 3D Create simulation environment. The client can access the KB directly and query whether or not the product description has reference parts or visual components connected to the ontology. If the products and parts exist and they have a corresponding visualisation, they can be retrieved to the simulation environment. The process plan is simulated inside 3D Create and validated processing times are updated to the KB. If there have been changes to the layout or resources used, the corrected models



Resource Loader

Products: VW-Door-product

Activities:
VW-Door-Grip-10-555
VW-Door-Weld-20-310
VW-Door-Weld-12-346
VW-Door-Grip-08-271

Subassemblies: VW-Door-assembly1

Resources:
VW-Door-Handling_Robot-A-732
VW-Door-Arcweld_Robot-A-764
VW-Door-Spotweld_Robot-A-749
VW-Door-Handling_Robot-A-720

Parts: VW-Door-DI311A-143

Status
File loaded

```
  File "Turn unit #2::PythonScript", line 25, in OnRun
AttributeError: 'NoneType' object has no attribute 'Name'
NM 45-2.0: Action 'Grasp' executed (Ok)
NM 45-2.0: Action 'Release' executed (Ok, released to node container)
Simulation auto-halted, Please reset simulation to start over.
Click or drag to select components; hold shift+ctrl to toggle select.
```

Figure 7.10: CoreOntology as a source for populating the simulation environment

or values can be uploaded to the KB.

Knowledge Exchange Between 4DIAC and KB

The Framework for Distributed Industrial Automation and Control (4DIAC) is being integrated with the PiSA SP3 toolchain for programming the various control devices within the production process. The control programs developed with the 4DIAC-IDE (Development Environment) use Function Blocks as the main building blocks [60]. Within the PISA project, the 4DIAC-IDE is connected to the KB and stores Function Block Types within the KB, as can be seen in Figure 7.11.

Function Block Types (FBT) are library instances of functional elements. Within the 4DIAC-IDE the developer can instantiate these FBTs and use

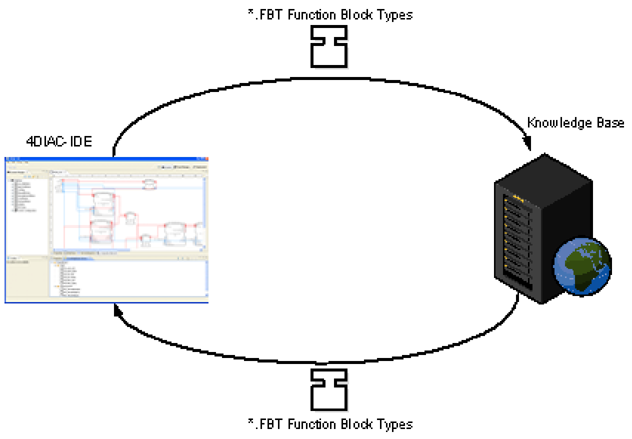


Figure 7.11: Ontologies and KB as knowledge exchange media in the PiSA advanced design, planning, and optimisation toolset

these instances to develop the control program for the hardware that is controlled. By storing all the FBT files that are available within the 4DIAC-IDE in the KB, a complete library is created that can be accessed by different users/developers. FBTs can be functional elements for all kinds of operations, such as mathematical operations, motion control, GUI development, and Human-Machine Interfaces. By sorting them according to their functionality and saving them into the KB, the developers get a better overview of what kind of functionality is being offered by the FBTs [60].

Required Knowledge Exchange Between 4DIAC and CAMeLEAN

The interaction between CAMeLEAN and 4DIAC takes place via the Knowledge Base (KB). CAMeLEAN depicts the planning for the manufacturing process, including the hardware components used within the process. As illustrated in Figure 7.12, CAMeLEAN can define multiple different kinds of resources, such as robots, conveyors, and operators. From the 4DIAC point of view the resource information that is required is the definition of the conveyors and/or end tools used in the process plan.

These resources are stored within CAMeLEAN into the class named cResources. cResources are parsed via a dedicated XML parser, which translates the CAMeLEAN internal data structures into the form of an ontology and saves it to the KB in RDF format. By connecting to the Knowledge Base, 4DIAC can read out the hardware elements that are necessary within the

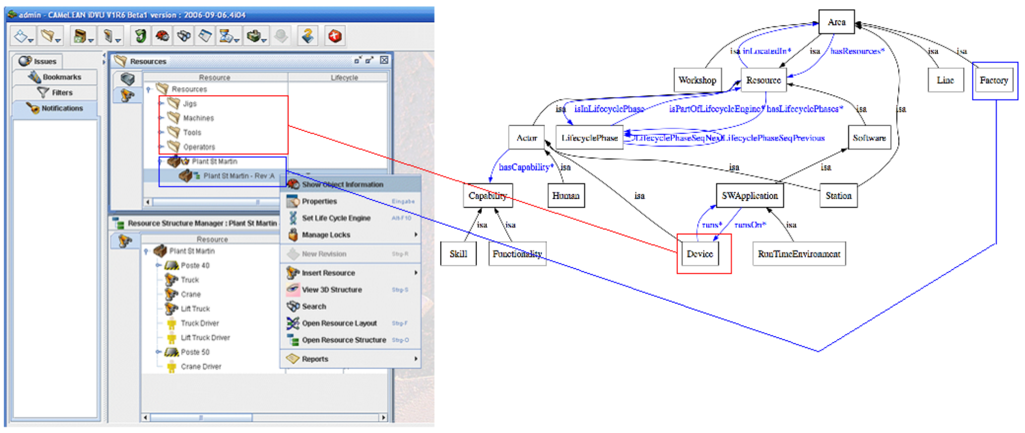


Figure 7.12: Resources descriped in the CAMeLEAN design environment and connection to the Core Ontology

manufacturing process and present them in the 4DIAC-IDE environment [60].

Summary of the Case Study 2

This case study dealt with three design and planning systems, which have very different understandings of semantics and/or values. Even when operating in the same field, personal/regional aspects of the same concepts can cause conflicts in transferring knowledge. Naturally, if an ontology is assessed by different users, they usually assign different beliefs to the concepts of the ontology, which are based on their interpretation of the view on the world represented by this ontology. Hence, the conflict of beliefs can result in a conflict of interpretation and thus of the semantics of the ontology [15].

One of the possible solutions, as demonstrated here, is a defined domain ontology where ontological modelling provides simplification and categorisation of the knowledge into a more typified but still flexible form. Each of the actors may still preserve their perception of the knowledge but by using common terms and rules to model the knowledge in the transfer process, more advantages can be achieved.

The client-independent knowledge, which is still readable and understandable by the clients, can be archived by using semantic mapping, in which the semantic models of the clients' exported data structures are transformed into a generalised knowledge representation such as the Core Ontology.

There is a risk involved within semantical mapping between different design systems, because it is virtually impossible to map and convert all the terms and details of each semantic model to another one. The complete transformation of one knowledge model to another requires much flexibility from the mapping process, but can be reached within some limitations and by realising that modelling everything is not required in order to increase productivity.

In conclusion, mapping all the information available is not needed, and wisely used interfacing provides a very general but flexible knowledge representation without compromising its validity. Knowledge stored in this kind of model is not required to be 'complete' or even fully compliant with the existing knowledge. The case implementation resulted in a system where the KB is used as a reference architecture. The knowledge that can be mapped is mapped and the rest is considered as metadata and links to proprietary formats.

7.3 Case Study 3 - Ontologies and KB for Proposed Holonic Manufacturing System

The third case study evaluates the expressiveness and soundness of a knowledge representation and its overall performance in a holonic manufacturing environment. By utilising the DiMS concept from Nylund and Salminen [48,49], combined with the information architecture, a laboratory demonstration was conducted. The DIMS concept aims at setting up a holonic manufacturing environment.

The holonic concept is a possible way to facilitate a balance between the necessary logical rules and a global abstract description of the entities that provide the capability of communication with and between the system elements. The DiMS concept provides guidelines for the overall control architecture, while the knowledge representation that was developed and the Knowledge Base that was implemented were used as the core of the demonstration. The knowledge representation defined the basic communication interfaces for the holarchy and defined what information would be exchanged.

The second goal for the KR and KB was to test the feasibility of connecting different actors under the same reference architecture and to update the KR with simulated data such as processing times and real manufacturing values. The holonic entities in this case were a robot unit (a drill attached to a Fanuc

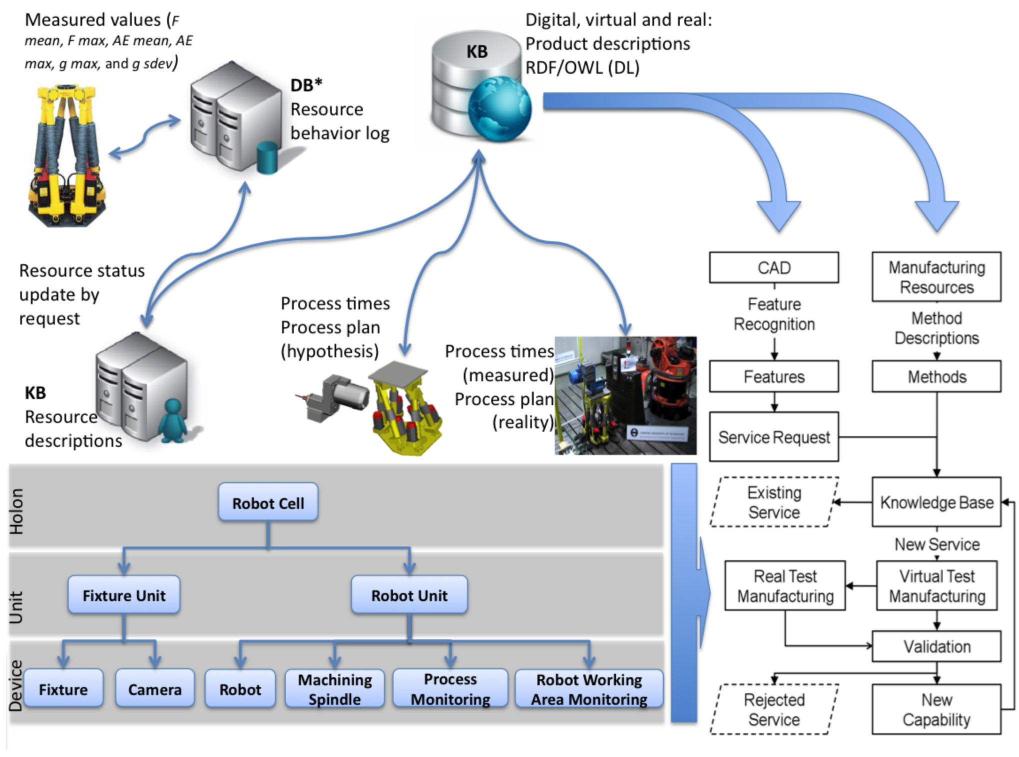


Figure 7.13: Reference Architecture, System Description and Workflow

200iB parallel kinematics robot and the Safety Eye System from Pilz, a fixture unit (fixture and camera), an operator, and order software as holonic entities, illustrated in Figure 7.13.

The product knowledge of each part, in demonstration part A (a cylindrical object with two holes) and part B (a similar cylindrical object with four holes), were analysed with Pro-FMA. Pro-FMA defined the types and characteristics of the features that were found and their locations, as well as describing the sequence for creating the holes. The knowledge of the part was first sent to the simulation environment, 3DCreate, where the virtual manufacturing was conducted on the basis of the available resources, such as a human operator capable of transferring material and ready parts and a device combination capable of performing the required drilling operation.

The second entity, the fixture unit, consisted of a camera and a fixture unit for ensuring that the part was fixed and in the right position before the manufacturing process could be started. The connections of the software and hardware entities are illustrated on the left-hand side of Figure 7.13 and the

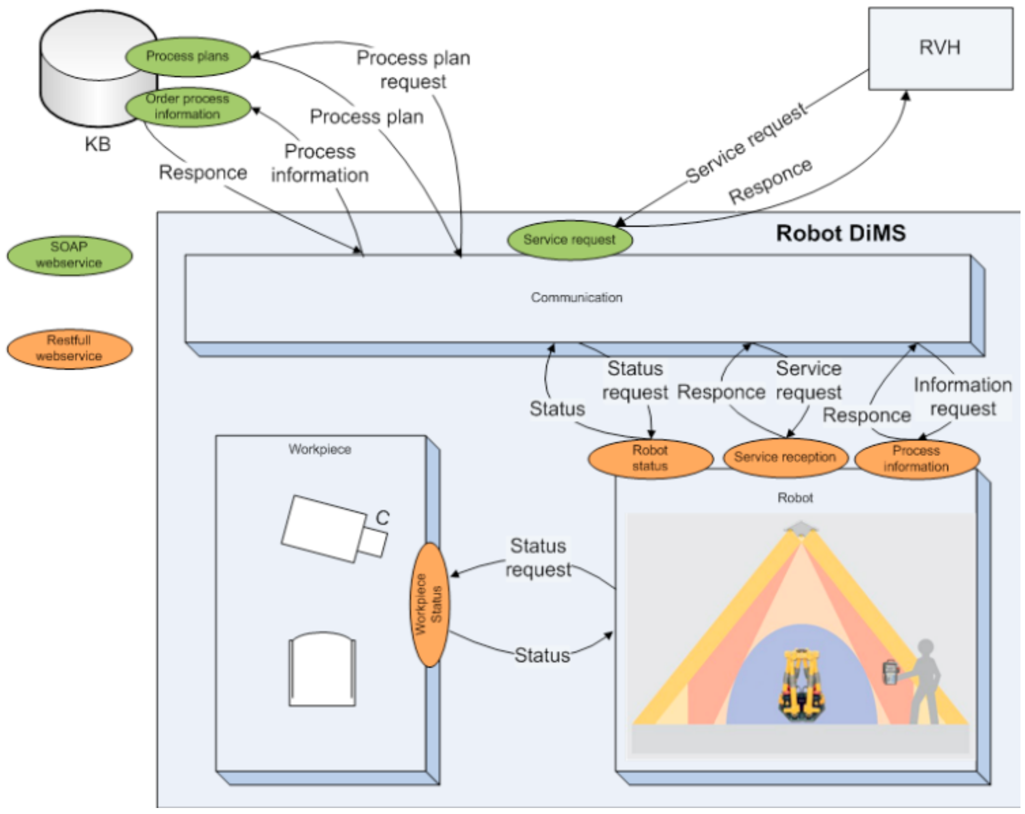


Figure 7.14: Communication of holonic entities in the laboratory demonstration [55]

steps taken in the process are illustrated on the right-hand side of Figure 7.13.

The taxonomy of the real robot cell is presented in Figure 7.13. The robot cell is a resource entity at the level of a manufacturing stage. The cell is divided at the manufacturing unit level into a fixture unit, a robot unit, and a human operator. The real part of the robot cell consists of the robot itself, a Safety Eye, and a machining spindle. The robot is a Fanuc F-200iB six-degrees-of-freedom servo-driven parallel link robot. The robot itself lacks the ability to announce some aspects of the required data, so the intelligence of the robot was augmented using a force and moment sensor.

The data from the sensor are stored in the KB, where the state of the robot is monitored against an acceptable value range. If the data from the sensor reach a certain limit, the robot will perform a predefined task, which can be, for example, stopping the ongoing process and announcing its changed state (in this case an error) to the surrounding system via the KB.

The robot's operational work sequences were preprogrammed with Labview as subroutines. The drilling procedure was connected to a feature representation of the part stored in KB. The service request, named the Order Holon (OH) in the flowchart shown in Figure 7.13, started the whole process by requesting a new manufacturing processes for the demo part via web services. If the process description did not exist for the part, the user was notified that the process definition was missing. If the OH did find the process attached to the part description, the KB returned the process description to the Order Item Holon (OIH), which forwarded the message to the Labview interface. Labview operated in this demonstration as an interface for the robot holon (RRH) [40, 55].

The communication of the holonic manufacturing system was based on web services, as illustrated in Figure 7.14. The robot holon carried out the drilling procedure and the sensors attached to the drill collected and sent the machining values: identification (ID) of part and process, force (F) from the force sensor, Acoustic Emission (AE), and acceleration (g) dedicated to each feature (or a process sequence back) to the controlling interface and the controlling interface returned the operational values back to the KB.

Summary of the Case Study 3

The aim was to utilise the knowledge created in a heterogeneous environment and contributed to a reference architecture maintained by the KB as a basis for automatic process generation and the simulation of assembly systems on different levels of abstraction. The approach also aimed to explore the possibilities of utilising the product-, process-, and system-related knowledge in an environment where different design and execution systems are contributing and retrieving information on different levels of abstraction.

The current status of the knowledge representation allows: (1) the meaning of the content to be combined with proprietary/closed formats by offering references and content description for the models, and (2) this architecture to be used in a holon-based manufacturing environment. The model itself can capture the information of the context during the time of operation.The demonstration highlighted the importance of a reference model as a connecting medium between different tools, varying from the product design to the simulations, and to the execution of the manufacturing processes.

7.4 Impact of the Case Studies on the KR

The cases each tested the feasibility of the KR. Case 1, which utilised the KR as the main format for publishing the results of feature recognition to the KB, tested the Object domain and its connections to the Feature class. On the basis of this case it was found out that some of the classes may need more definition.

The implementation of Case 2 evaluated the meaning of the classes and relations compared to the vocabulary the surrounding clients used. For the most part the KR worked as required; however, the connection between proprietary files and the ontology is still under discussion. The case also showed that a resource library needs to be modelled in order to reduce copying information from one set of instances to another. It also gave requirements for how the capabilities should be modelled. On the basis of the feedback, a new application was created for creating new resources and maintaining their characteristics. The tool also allows the generation of capabilities for devices according to the KR specified here.

The third case study evaluated the KR from the manufacturing system point of view. As the KR is able to capture the machining values per product, this suggest that in the future the operational values must also be collected for a device. This requires possible extensions towards device classes. These extensions were under development during the publication of this thesis.

Chapter 8

Conclusions

Design and manufacturing operations are global and companies are dependent upon the efficient and effective exchange of knowledge within the company and among its suppliers. Even though the OEMs are trying to dictate the selection of design tools, the reality is often such that the suppliers and partners are forced to keep several similar design tools. This often leads to problems of knowledge exchange and interoperability between numerous design tools, since once out of the authoring system the model loses most of the saved design details, the meaning of the entities' information content, and connections to the original model.

The objectives of the thesis were:

1. to develop and create a method for representing product structure and the corresponding process representation;

2. to extend the representation to the product feature level and represent the processes that occur on the feature level;

3. to develop a model for representing product-, process-, and system-specific knowledge on the basis of the relations between the product level and the feature level, and

4. to formalise the model thus developed into the form of a knowledge representation that allows knowledge inference to be applied.

This thesis proposed a rigorous model which can provide well-defined meaning for entities in the integration of a design and manufacturing environment. The Assembly State Decomposition Model (ASDM) showed the connectivity between the products and processes. The Manufacturing State Decomposition Model (MSDM) defined the connection between manufacturing features and operations. The Product-Process-System (PPS) model illustrated that the combination of the product-, process-, and system-related information is possible on the conceptual level.

On the basis of the relations defined in the PPS model, the knowledge representation was created. The knowledge representation formalised in FOL. In order to implement the KR, the Core Ontology was created. For modelling the classes and the relations between them, a Protégé OWL DL version 3.4 ontology modelling tool, was used. For visualisation, a shareware tool, Graphviz, was used.

The cases where the knowledge representation that was implemented was used showed that the Core Ontology is generic yet expressive enough and possesses enough rigidity to be used as a standardised knowledge exchange interface between the design tools and that it can be used in a dynamic manufacturing environment as a common knowledge exchange interface and a knowledge-capturing repository. It was seen that the common knowledge architecture served its purpose by reducing the number of interfaces needed in the holonic manufacturing environment. While the amount of interaction between entities still remains large, each of those systems will utilise the knowledge representation introduced here for retrieving information that is needed and saving the results of reasoning performed and actions taken.

The thesis proposes a novel approach to how the process planning and manufacturing-related knowledge can be structured, enhanced, and exchanged among different systems. The ontological approach represented here proved that:

1. product features can be used for pre-defining the required processes and setting requirements for manufacturing systems, and
2. the knowledge can be viewed from production time perspective.

However, the result of this thesis is not aimed at producing a standard which commercial suppliers, vendors, or companies could implement immediately.

Numerous potential standards, de facto standards, and models defined inside certain interest groups exist which can convey product-, process-, or system-related information, such as ontoSTEP, OAM (Open Assembly Model), PSL

(Process Specification Language), and CMSD (Core Manufacturing Simulation Data). Each of these can capture much larger quantity of domain-specific knowledge. By adopting the standardised representations into the formalised model represented here, the amount of knowledge that is captured can be increased and yet the KR can still keep its generic nature.

Chapter 9

Summary of Contributions

Even though multiple elegant and standardised solutions for capturing, structuring, and inferring the product, process, and system information of a chosen domain exist, they are separate models with significant overlaps with each other. There are no platform-independent methods or tools available at the moment for modelling integrated product, process, and system knowledge.

Likewise, there are no systems on the market that have the capability to link processes, behaviours, or functions to manufacturing and assembly features as an independent system and a re-usable tool. Partly for this reason, neither a tested nor an implemented reasoning tool between product and process knowledge exists (other than software-specific expert systems dedicated to a specialised domain).

The defined Knowledge Representation Formulation (KRF) methodology shows that product-, process-, and system-specific knowledge is linked and can be modelled in a feasible way. The intermediate steps of the KRF methodology showed in detail how the domain-specific knowledge areas can be linked with each other. For this two graph-based methods, the Assembly State Decomposition Model (ASDM) and Manufacturing State Decomposition Model (MSDM), were developed. These methods do not only serve for the formulation of a conceptual knowledge representation model, such as PPS in Figure 5.14, but can be used as stand-alone methods for process design purposes.

As another contribution, this thesis provides a methodology, the Knowledge Representation Formulation (KRF) methodology, for creating a knowledge representation for integrating separate domains.

The thesis showed that process specific knowledge can be linked to the geometric features of the product, as explained in Chapter 5, Step 2, and in Case Study 1. Unlike the ontologies presented in other research projects, this ontology included the product features as the third important knowledge domain. In the end it is the product features that set the requirements for the processes and systems and not the other way around, though links backwards exist as well.

The effect of assembly-specific features, as well as manufacturing features, can be reasoned about and propagated between different kinds of modelling and analysis tools in such a way that the knowledge content of the model is still preserved.

The implementation supports the claim that different clients and different data structures can be interconnected via a generic domain ontology. The ontology can explain the structures of a model by going into single features or it can describe the content of a model in the manner of a reference architecture. It also showed that comparing the conventional methods, which do not include the utilisation of product information in feature level, to this approach, the manual knowledge input decreased over the time.

The knowledge representation, the Core Ontology, and the implementation of the Knowledge Base (KB) environment showed that if the knowledge is mapped to a generic format, the validation of the content becomes faster and easier. Since the design support tools do not need to create all-to-all interfaces among each other, the amount of re-modelling decreases. The knowledge representation was also able to keep the level of information content over a time period instead of succumbing to the original design-by-snapshot methodology. As a contribution to the research projects, it can also be concluded that the model supports the re-use of models in such a way that the implementation of this kind of knowledge representation becomes feasible.

Chapter 10

Future Work

Technical developments in recent years have produced stand-alone systems where performance is routinely reached. This solid background has allowed the extension of these systems into networks of components, which gather very heterogenous elements, each in charge of a part of the holistic action of the system. Because of the desired increase in efficiency, these components need to be more and more integrated to the design phase in the sense that their mutual interactions are extremely strong and have a considerable effect on the final system's output. The types of interactions are changing into a complex network of possibilities within certain limits instead of a steady and predefined process flow. This situation is relatively new and causes pressures for defining the role of intended interaction [10, 12, 13].

According to Chavalarias et al. [10], there is no doubt that one of the main characteristics of complex and adaptive production platforms in the future will be the ever-increasing utilisation of ICT. Computing power has opened up the possibility of storing huge datasets and performing massive calculations on those sets and eventually filtering the usable information for the system's use. However, while the industrial world has seen the possible advantages, the implementations fall short as a result of the changes to the whole production paradigm that are required, going from preplanned hierarchical systems to adaptive and self-organising complex systems.

The continuation of the work represented here in the near future consists of building the knowledge representation so that it is more representative and extended in order to capture more of the filtered information fed from different downstream applications, as illustrated in Figure 6.2. It is envisioned that a formal knowledge representation will be developed that allows for the integration of static and mobile manufacturing entities with advanced capa-

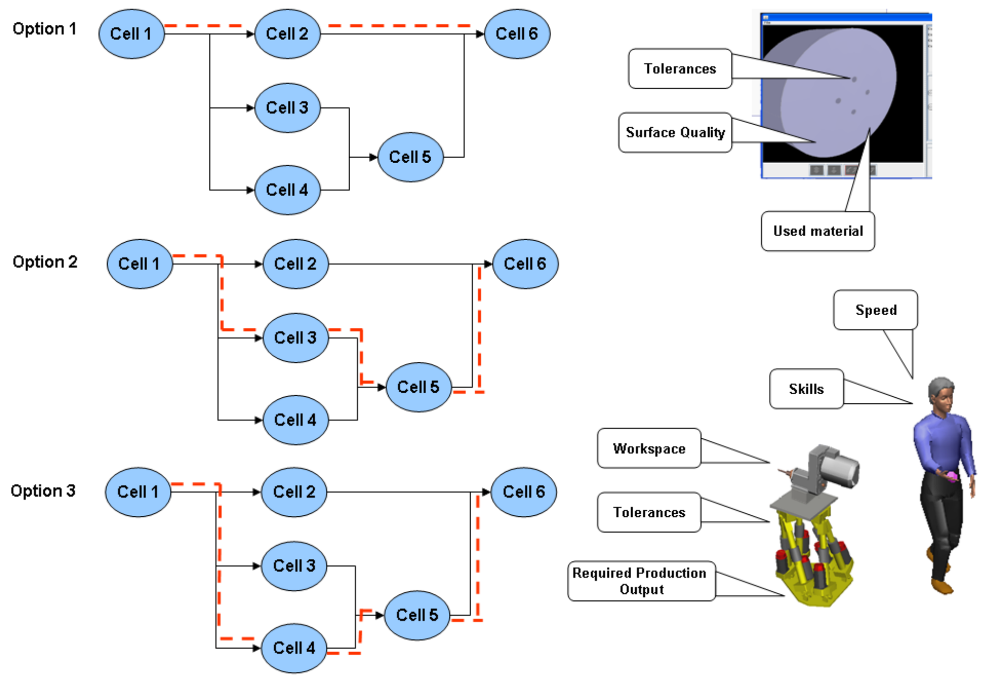


Figure 10.1: Options for process flow based on the product features, device capabilities and other production parameters

bilities and described behaviours for an adaptive and self-organising production environment. The primary knowledge framework must provide a means to support the aims of monitoring resource conditions and the use of energy per work piece, and serve as a generic yet time-based storage unit for production knowledge. This platform will eventually be the knowledge platform where the "intelligent" reasoning will be done.

In order to enhance the processes inside the virtual factory or real factory, the fastest, cheapest, and most energy-efficient production method needs to be calculated on the basis of the real production parameters. The defined multi-criteria have to include the aspects of resource utilisation levels, device reliabilities, energy consumption, and other sustainability-related factors, the adjustment of which relates to the costs of the operations. The process flow changes dynamically, depending on the parameters defined in the multi-criteria and on the dynamic production environment as illustrated in Figure 10.1.

The generation of process plans according to the scheduling and known workflow do need the description of resource characteristics that identify what resource combinations can be used to fulfil the process requirements set by the product. This leads to challenges in the creation of a feasible knowledge representation of various system components and the rules for combining those. During the finalisation phase of this thesis, some work towards expressing system interfaces in order to combine devices together was being done.

In future more of the real working conditions will be saved to the production history of the part, as well as the operational history of the single resource. By saving this information, the backtracking of the systems' conditions, energy consumption, or design features of the parts becomes feasible. In addition, the current use state of the tool and other characteristics of machinery can be saved and used to improve the process efficiency, pinpointing the possible quality problems related to the features of the part or single machines and the overall improvement of various processes.

The version of the Core Ontology used here was very generic by nature and the future work consists, among others, of developing more detailed resource models to capture capabilities, geometrical properties, and other metadata relevant to that particular resource. The models need to be improved in such a manner that they can capture and convey the operational parameters of each resource into the simulation environment, where the different production scenarios can be tested before being applied on the factory floor. Possible models exist, such as Core Manufacturing Simulation Data (CMSD) from Riddick and Lee [59], for representing the basic process and resource information, but in order to fulfil the requirements of the holon-based manufacturing framework there is a need for the development of new models or extensions of the existing ones. The other option is the PSL [4, 27] which also allows the process description, but in the form of a formal ontology.

Bibliography

[1] E. Awad and H. Ghaziri, *Knowledge management*, Prentice Hall, 1st edition ed., 2004.

[2] F. Baader and W. Nutt, *Basic description logics*, in In the Description Logic Handbook - Theory, Implementation and Applications, F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, eds., Cambridge University Press, 2002, p. 574.

[3] R. Bidarra and W. Bronsvoort, *Semantic feature modeling*, in Computer Aided Design, vol. 32, Elsevier, 2000, pp. 201–225.

[4] C. Bock and M. Gruninger, *Psl: A semantic domain for flowmodels*, Software System Modelling, 4 (2005), pp. 209–231.

[5] S. Borgo and P. Leit, *Foundations for a core ontology of manufacturing*, Integrated Series in Information Systems, 14 (2007), pp. 751–775.

[6] W. Borst, *Construction of Engineering Ontologies*, PhD thesis, University of Tweenty, Enschede, NL, 2005.

[7] A. Caglyan and C. Harrison, *Agent Sourcebook - A complete Guide to Desktop, Internet and Intranet Agents*, Wiley Computer Publications, 1997.

[8] P. Cech, *Ontology as an integrating element in an information-knowledge infrastructure*, in International Conference on Multimedia and ICT in Education, 2009, p. 5.

[9] V. Chaudri, A. Farquhar, R. Fikes, P. Karp, and J. Rice, *The generic frame protocol 2.0*, July 1997.

[10] D. Chavalarias, K. Aberer, E. Aurell, O. Babaoglu, C. barrett, P. Bessiere, P. Bourgine, G. Caldarelli, L. Cardelli, J. Kasti, M. Cotsaftis, et al., *Complex systems: Challenges and*

opportunities, an orientation paper for complex systems research in fp7, European Comisssion, 2006.

[11] Clark and Parsia, *Pellet: Owl 2 reasoner for java*, 2010.

[12] M. Cotsaftis, *From trajectory control to task control - emergence of self-organization in complex systems*, Emergent Properties in Natural and Artificial Dynamical Systems, (2006), pp. 3–22.

[13] M. Cotsaftis, *A passage to complex systems*, in Complex Systems and Self-organization Modelling, C. Bertelle, G. H. E. Duchamp, and H. Kadri-Dahmani, eds., Springer, 2009, p. 236.

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

[15] A. Daemi and J. Calmet, *Assessing conflicts in ontologies*, WSEAS Transactions on Information Science and Applications, 5 (2004), pp. 1289–1294.

[16] R. Davis, H. Shore, and P. Szolovits, *What is a knowledge representation?*, AI Magazine, 14 (1993), pp. 17–33.

[17] L. H. de Mello and A. C. Sanderson, *A correct and complete algorithm for the generation of mechanical assembly sequences*, IEEE Transactions on Robotics and Automation, 7 (1999), pp. 228–240.

[18] D. Deneux, *Introduction to assembly features: An illustrated synthesis methodology*, Journal of Intelligent Manufacturing, 10 (1999), pp. 23–39.

[19] H. Dullah, E. Bohez, and M. Irfan, *Assembly features: definition, classification, and usefulness in sequence planning*, International Journal of Industrial and Systems Engineering, 4 (2009), pp. 111–132.

[20] J. Ellson, E. Gansner, Y. Hu, and A. Bilgin, *Graphviz - graph visualization software*, 2010.

[21] T. D. Fazio, *A prototype of feature-based design for assembly*, in Advances in Design Automation, ASME, 1997, pp. 9–16.

[22] S. Fenves, S. Foufou, C. Bock, and R. Sriram, *Cpm: A core model for product data*, nistr publication, National Institute of Standards and Technology, 2005.

[23] R. Fikes, A. Farquhar, and J. Rice, *Tools for assembling modular ontologies in ontolingua*, tech. rep., Knowledge Systems, AI Laboratory, 1997.

[24] S. C. for Biomedical Informatics Research, *Protégé owl dl ontology editor version 3.4*.

[25] W. Grosso, J. Gennari, R. Fergerson, and M. Musen, *When knowledge models collide (how it happens and what to do)*, tech. rep., Stanford Medical Informatics, Stanford University Stanford, 1998.

[26] T. Gruber, *A translation approach to portable ontology specification*, Knowledge Acquisition, 5 (1993), pp. 199–220.

[27] M. Gruninger and J. Kopena, *Planning and the process specification language*, in Workshop on the Role of Ontologies in Planning and Scheduling, ICAPS, 2005, pp. 22–29.

[28] W. Holland, *Assembly Features in Modeling and Planning*, PhD thesis, Delft University of Technology, NL, 1997.

[29] W. Holland and W. Bronsvoort, *Assembly features in modeling and planning*, Robotics and Computer Integrated manufacturing, 16 (2000), pp. 277–294.

[30] A. Hopgood, *Intelligent Systems for Engineers and Scientists*, CRC Press, 2nd edition ed., 2000.

[31] E. Järvenpää, M. Lanz, J. Mela, and R. Tuokko, *Studying the information sources and flows in a company supporting the development of new intelligent systems*, in proceedings of Flexible Automation and Intelligent Manufacturing, 2010.

[32] M. Johansson, B. Johansson, A. Skoogh, S. Leong, F. Riddick, Y. Lee, G. Shao, and P. Klingstam, *A test implementation of the core manufacturing simulation data specification*, in Winter Simulation Conference, 2007, pp. 1673–1681.

[33] H. Ko and K. Lee, *Automatic assembling procedure generation from mating conditions*, Computer Aided Design, 19 (1987), pp. 3–10.

[34] A. Koestler, *The Ghost in the Machine*, Penguin, 1967.

[35] S. Krima, R. Barbau, X. Fiorentini, S. Rachuri, and R. Sriram, *Ontostep: Owl-dl ontology for step*, tech. rep., National Institute of Standards and Technology, 2009.

[36] M. Lanz, *An approach to feature-based modelling and analysis for the final assembly*, Master's thesis, Tampere University of Technology, January 2005.

[37] M. Lanz, T. Kallela, E. Järvenpää, and R. Tuokko, *Ontologies as an interface between different design support systems*, in International Conference on Neural Networks or Artificial Intelligence Series, WSEAS, 2008, pp. 202–207.

[38] M. Lanz, T. Kallela, G. Velez, and R. Tuokko, *Product, process and system ontologies and knowledge base for managing knowledge between different clients*, in International Conference Distributed Human-Machine Systems, IEEE SMC, 2008, pp. 508–513.

[39] M. Lanz, P. Luostarinen, and P. Andersson, *Feature recognition as a basis for the cost allocation and decision making toolset*, in 20th International Conference on Production Research, 2009.

[40] M. Lanz, H. Nylund, A. Ranta, P. Luostarinen, and R. Tuokko, *Set-up and and first steps on capturing of realistic resource characteristics of an intelligent manufacturing environment*, in Flexible Automation and Intelligent Manufacturing (FAIM), 2010.

[41] M. Lanz and R. Tuokko, *Generic reference architecture for digital, virtual and real representations of manufacturing systems*, in Indo-US Workshop on Designing Sustainable Products, Services and manufacturing Systems, S. Rachuri, ed., National Institute of Standards and Technology, 2009.

[42] M. Lanz, R. Velez, and R. Tuokko, *Feature-based modeling and analysis for knowledge-intensive concurrent engineering in final assembly*, in Computer Aided Industrial Design and Concurrent Design, 2005.

[43] J. Lastra, *Reference Mechatronic Architecture fro Actor-based Assembly Systems*, PhD thesis, Tampere University of Technology, 2004.

[44] C. Leondes, *Expert Systems - The Technology of Knowledge Management and Decision Making for the 21st Century*, vol. 1, Academic Press, 2002.

[45] N. Lohse, *Towards an Ontology Framework for The Integrated Design of Modular Assembly Systems*, PhD thesis, University of Notthingham, May 2006.

[46] M. F. Lopez, *Overview of methodologies for building ontologies*, in Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5), August 1999.

[47] R. Neches, R. Fikes, T. Finin, T. Gruber, T. Senator, and W. Swartout, *Enabling technology for knowledge sharing*, AI Magazine, 12 (1991), pp. 36–56.

[48] H. Nylund, M. Hokkanen, K. Salminen, and P. Andersson, *Lifecycle of digital manufacturing supporting design and development of adaptive manufacturing systems*, in 12th Annual International Conference on Industrial Engineering – Theory, Applications and Practice, 2007, pp. 544–549.

[49] H. Nylund, K. Salminen, and P. Andersson, *Digital virtual holons - an approach to digital manufacturing systems*, in 41st CIRP Conference on Manufacturing Systems, 2008, pp. 103–106.

[50] R. V. Osuna, *Pisa sp3 task 3.1.4 conceptual toolset definition*, project report, Visual Components, 2007.

[51] PABADIS'PROMISE, *D3.1. development of manufacturing ontology*, project deliverable, The PABADIS'PROMISE consortium, 2006.

[52] E.-I. project, *Commonkads*, 2006.

[53] S. Rachuri, Y.-U. Han, S. Foufou, S. Feng, U. Roy, F. Wang, R. Sriram, and K. Lyons, *A model for capturing product assembly information*, Computing and Information Science in Engineering, 6 (2006).

[54] H. K. Rampersad, *Integrated and Simultaneous Design for Robotic Assembly*, Wiley Series in Product Development, 1994.

[55] A. Ranta, K. Ikkala, K. Salminen, M. Hlâĉek, and L. Vaŝek, *Tampere manufacturing summit 2009 demonstraatio - loppuraportti*, project report, Tampere University of Technology, 2009.

[56] S. Ray, *Tackling the semantic interoperability of modern manufacturing systems*, in Proceedings of the Second Semantic Technologies for eGov Conference, 2004.

[57] S. Ray and A. Jones, *Manufacturing interoperability*, Journal of Intelligent Manufacturing, 17 (2006), pp. 681–688.

[58] S. Ray and S. Wallace, *A production management information model for discrete manufacturing*, Production Planning and Control, (1995), pp. 65–79.

[59] F. Riddick and Y. Lee, *Representing layout information in the cmsd specification*, in Proceedings of IEEE Winter Simulation Conference, 2008, pp. 1777–1784.

[60] R. Rodriguez, M. Lanz, and M. Rooker, *Fp6 pisa sp3 task 3.4.1 implementation of the knowledge base*, project report, Tampere University of Technology, 2009.

[61] J. Shah and R. Tadepalli, *Feaure-based assembly modeling*, in Proceedings of ASME International Computers in Engineering Conference, 1992, pp. 253–260.

[62] R. Sondhi and J. Turner, *Representing tolerance and assembly information in a feature-based design environment*, in Proceedings of ASME Design Automation Conference, vol. 32-1, 1991, pp. 101–108.

[63] J. Sowa, *Knowledge Representation: Logical, philosophical, and computational foundations*, Brooks Cole Publishing Co., 2000.

[64] R. Studer, V. Benjamin, and D. Fensel, *Knowledge engineering: principles and methods*, Data and Knowledge Engineering, 25 (1998), pp. 161–197.

[65] T. Tallinen, R. V. Osuna, J. Lastra, and R. Tuokko, *Product model representation concept for the purpose of assembly process modeling*, in Proceeding of the International Symposium on Assembly and Task Planning, 2003.

[66] E. Tran, *Requirements and specifications*.

[67] D. Tsarkov and I. Horrocks, *Ordering heuristics for description logic reasoning*, in Proceedings of the 19th International Joint Conference on Artificial Intelligence, 2005.

[68] P. Valckenaers, H. van Brussel, L. Bongaerts, and J. Wyns, *Holonic manufacturing execution systems*, CIRP Annals - Manufacturing Technology, 54 (1994), pp. 427–432.

[69] A. van der Net, *Designing and Manufacturing Assemblies*, PhD thesis, Eindhoven University of Technology, 1998.

[70] W3C, *Owl web ontology language overview.*

[71] C. H. Yu, *Abduction? deduction? induction? is there a logic of exploratory data analysis?*, 1994.

[72] J. Zhao and S. Masood, *An intelligent computer-aided assembly process planning system*, International Journal of Advanced Manufacturing Technology, 15 (1999), pp. 332–337.