INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED'07
28 – 31 AUGUST 2007, CITE DES SCIENCES ET DE L'INDUSTRIE, PARIS, FRANCE

INFORMATION ORGANISATION IN DESIGN: AN APPLICATION OF FORMAL ONTOLOGIES

A.G. Gunendran[1], A.F. Cutting-Decelle[2], R.I.M. Young[1], J.P. Bourey[3]

[1]Wolfson School of Mechanical and Manufacturing Engineering, Loughborough University, Loughborough, UK
[2]Ecole Centrale de Paris, Laboratoire de Génie Industriel, France
[3]Ecole Centrale de Lille, LGIL, France

ABSTRACT

Design activities need to be satisfied by several context considerations. The design data, information and knowledge need to be understood and interpreted by multiple context applications. Therefore, the organisation of those information and knowledge must be managed in a manner that all participants of design activities understand and share the actual meanings of the terms. An important approach onto design activities is to decompose the design activity into a set of processes, each of them being inter-related and sharing and exchanging information. Ontologies are potentially very powerful tools to share information and knowledge among systems. A number of different definitions are available for the word « ontology », but all of them highlight the basic functionality of an ontology as a methodology for enabling a better sharing of the meaning of terms. In this paper, we propose two approaches based on formal ontologies, thus enabling a better organisation of the information handled during the design process: the first one makes use of the PSL (Process Specification Language) as a key-enabler for process based communications and information exchanges. The other one proposes methods and tools facilitating communications through model based transformation (MDA) approaches.

Keywords: design, formal ontologies, information modelling, MDA, PSL

1. INTRODUCTION

Design activities need to be satisfied by several context considerations. The design data, information and knowledge need to be understood and interpreted by multiple context applications. Therefore, the organisation of those information and knowledge must be managed in a manner that all participants of design activities understand and share the actual meanings of the terms. An important approach is to decompose the design activity into a set of processes, each of them being inter-related and sharing and exchanging information.

Ontologies are potentially very powerful tools to share information and knowledge among systems. A number of different definitions are available for the word « ontology », but all of them highlight the basic functionality of an ontology as a methodology for enabling a better sharing of the meaning of terms. Ontologies are used in several application environments; they can be classified from several points of view. The classification based on their expressiveness is particularly important for design information/knowledge management; this classifies the ontologies as lightweight or heavyweight. The lightweight ontologies are lacking in axioms and reasoning mechanisms. Therefore, they are limited in their ability to share information beyond a closed and well understood environment. On the other hand, heavy weight ontologies, which contain axioms and reasoning mechanisms, are capable of capturing the semantics of the terms and their intended meanings in a mathematically rigorous way. Hence, an heavyweight ontology approach is capable of defining meanings which can then be shared across different contexts of design. The use of heavyweight ontology methodologies for the definition of concepts is important for information sharing across the boundaries of different environments.

The information and knowledge of the multiple contexts of design activities need to be supported by heavyweight ontologies with formal and rigorous methods. Further, the methodologies by which design information is defined must be explored to support those information sharing and exchanges.

This paper proposes a view of the current status of design information and knowledge sharing issues and discusses the potential for ontological engineering approaches based on heavyweight formal ontologies such as PSL, but also on MDA based approaches to provide a solution.

2. INFORMATION SUPPORT IN THE DESIGN PROCESS

A wide range of information is needed to support product development decision making. Given the global nature of many modern companies this information must be communicated at a global level as individuals in a design team are likely to be based in different locations throughout the world. On the other hand, there is a distinct difference between information that is provided as text for user interpretation and information that is sufficiently structured to be interpreted by software applications. It is assumed that all information models provide sufficient structure to provide software support [1].

The provision of a common shareable information environment plays a critical role in an integrated product development process [2]. Whilst the research needed to achieve such an environment is substantial, the successful achievement of such an environment offers significant potential benefits to system users. Product models have been recognised for many years now to be a means to the provision of a common source of product information to support integration and data sharing [3], they also provide a source and repository for information concerning a product under development.

The issues to be resolved in information model support for product development can be raised through two simple questions: What information contexts are needed to support product design and manufacture in a global design and manufacture environment? And how can information be shared across these multiple contexts?

Figure 1 below provides a general view of an information supported integrated product development system, highlighting the separation of the applications which support design from the information which they use. This is an essential aspect of integrated information systems support and offers the potential advantages of complete data integrity, rapid flexibility, maintainability, vendor independence and life cycle support.

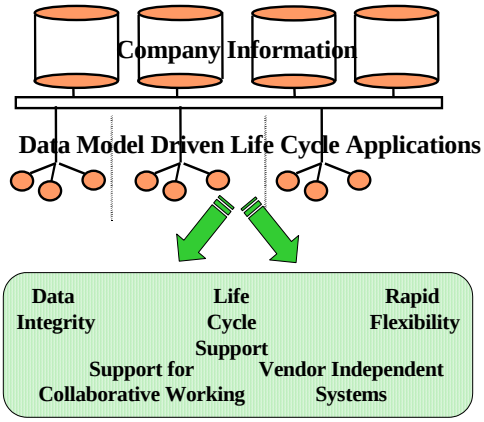

Fig. 1: Information supported integrated product development

The pursuit of an information supported integrated product development system raises a number of issues, among which: the functions supported in the product development process, the types of information models needed to support each function, the information structures needed to support the input and output of each function, the information needs to be shared between functions and the way of achieving this, the way of achieving flexible information systems, also readily maintained and updated, do information models just capture information or are some of them knowledge models and the way of characterising a useful set of information systems design tools. While the definition of appropriate system design tools and methods is an issue in integrated product development systems, especially in terms of requirements capture and requirement analysis, the research presented here has used the Unified Modelling Language (UML) to support system design and information model representation.

The basic concept of information model supported product development is that any software function that is involved in supporting product development draws on the information models for information inputs and will supply its outputs into the information models. The product information for each application is specific to the application context. The significance of these information structures is

that interactions between applications are handled via the information models. The only constraint on the action of an application is the availability of information in the information model. Interactions between applications must therefore be dealt with via interactions between the information in the models.

A product model must be able to support multiple information contexts if it is to be of real value in an integrated product development system. Such a model is considered to be at the heart of any such system as it must maintain all the key-information related to the product which is under development. The words 'context' and 'view' are widely used in current research papers to describe situations where multiple members of teams must share information. However these terms tend to be used interchangeably without any clear definition. According to the Oxford English dictionary context related to written text refers to "the parts which immediately precede or follow any particular passage of text and determine its meaning". Our interpretation of this in terms of information models is that context provides the environment that determines the terms, activities and information. The Oxford English dictionary definition of view is "a particular way of considering a matter or question" and we follow this definition directly. Applying these definitions we consider that each phase of the product lifecycle provides the context for the decisions within that phase. For example in a design context a manufacturing engineer would view a product in terms of its manufacturability. However, within a manufacturing context a manufacturing engineer would view a product in terms of planning or scheduling. These views of information within context and their relationships, need to be understood and defined if information is to be clearly organised. Figure 2 shows a UML class diagram of the general product data structure that has been defined into which multiple information contexts can be constructed. This is achieved through the contexts class (e.g. design context, manufacturing context, …), which can support a range of design views of a product, or manufacture views of a product, or other views as necessary.

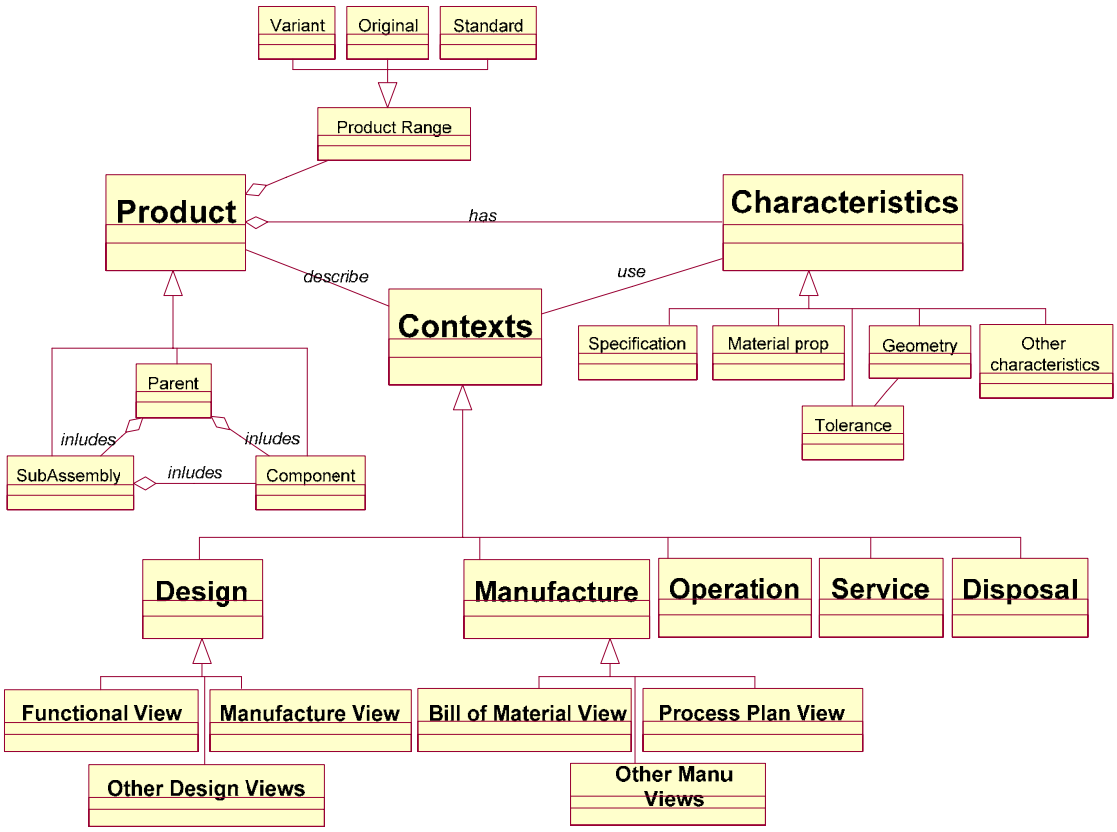

Fig. 2: A product data structure to support multiple contexts

In the following sections, we will present ontology-based approaches aimed at facilitating the correspondences between the different models needed by the product development process.

3. FORMAL ONTOLOGIES MET IN DESIGN

Design activities need to consider several kinds of information from various sources such as customers, internal groups, factories, suppliers and systems. These various sources have to be used by several persons to make design decision. Therefore, information coming from various sources has to be understandable and shareable by several users. Ontologies are identified as an outstanding method for the information sharing. However, the definition of the word ontology is varying depending on applications and covers a wide range from just the taxonomy of the information of concepts to more rigorously defined vocabulary of terms with well defined relationships and restrictions on the domain semantics utilising mathematical rules and theorems. In this section, we will present some of the main features of the ontologies usually met in design.

3.1 Main features

As the integration of the different software applications used in design increases [4], developers face increasingly complex problems related to their interoperability [5], [6], [7]. Independent contractors and suppliers who collaborate on demand to bring to market new products must share product-related data. Legacy vendor applications that are not designed to inter-operate must now share processes. When enterprises collaborate, a common frame of reference or at least a common terminology is necessary for human-to-human, human-to-machine, and machine-to-machine communication. Similarly, within a core enterprise where distributed collaboration between remote sites and production units take place, a common understanding of business and manufacturing-related terms is indispensable. However, this common understanding of terms is often at best implicit in the business transactions and software applications and may not even be always present. Misunderstandings between humans conducting business-related tasks in teams, and ad-hoc translations of software applications contribute to the rising costs of interoperability in manufacturing.

Ontology engineering offers a direction towards solving the interoperability problems brought about by semantic obstacles, i.e. the obstacles related to the definitions of business terms and software classes. Ontology engineering is a set of tasks related to the development of ontologies for a particular domain. An ontology is built on a taxonomy of concepts and their definitions supported by a logical theory (such as first-order predicate calculus).

Ontologies have been defined as an explicit specification of a conceptualization [8]. Ontology engineering aims at making explicit the knowledge contained within software applications, and within enterprises and business procedures for a particular domain. An ontology expresses, for a particular domain, the set of terms, entities, objects, classes and the relationships between them, and provides formal definitions and axioms that constrain the interpretation of these terms [9]. An ontology permits a rich variety of structural and nonstructural relationships, such as generalization, inheritance, aggregation, and instantiation and can supply a precise domain model for software applications [10]. For instance, an ontology can provide the object schema of object-oriented systems and class definitions for conventional software [11].

Ontological definitions, written in a human readable form, can be translated into a variety of logical languages. They can also serve to automatically infer translation engines for software applications. By making explicit the implicit definitions and relations of classes, objects, and entities, ontology engineering contributes to knowledge sharing and re-use [9].

3.2 Heavy / lightweight ontologies

The ontology community has introduced a classification based on the expressiveness of the definitions as lightweight, and heavyweight [12], [13]. The major difference between light- and heavyweight ontologies lies in the development of axioms and in the use made of those axioms.

- **Lightweight ontologies**: A taxonomic structure of terms with associated definitions of concepts, relationships between concepts and properties that describe concepts can be called as “lightweight ontologies” [12], [14]. Lightweight ontologies are hardly axiomatised (Gangemi et al., 2005). Therefore, lightweight ontologies can only be used either by members of a community for whom the meaning of the terms is more or less known in advance or within a closed software domain environment. Even if two lightweight ontologies are defined for the same domain, but in different environments there is no guarantee of information sharing among those ontologies [15]. Hence, information sharing by utilising lightweight ontologies is limited for either a human centered or a closed software boundary environments. Since the design environment requires information sharing

across various domains and the use of multiple software tools, lightweight ontologies are generally considered as not very well suited to the design process.
- **Heavyweight ontologies**: Heavyweight ontologies are extensively axiomatised and constrained in addition to the components of lightweight ontologies. The purpose of the axiomatisation is to exclude terminological and conceptual ambiguities, due to unintended interpretations and to clarify the intended meaning of the terms of the ontology [12], [14]. The rigorous expressiveness of heavyweight ontologies can be obtained by utilising mathematical rules and theorems. Further, information sharing can be performed in a very flexible way with the heavyweight ontologies even in the environments of people with different knowledge background as well as between the boundaries of multiple software systems. Therefore, heavyweight ontologies provide a very powerful method for information sharing in complex environments such as the design environment.

The PSL language presented in the following section provides a good example of an heavyweight ontology.

4. AN EXAMPLE OF A FORMAL ONTOLOGY: THE PSL LANGUAGE

ISO 18629 PSL is aimed at facilitating interoperability of product and process information by providing a generic language applicable to a broad range of specific process representations in manufacturing and other applications [4]. PSL is an ontology for discrete processes written in the Knowledge Interchange Format (KIF) [16] itself an ISO candidate in (ISO/JTC1 1999), (Common Logic [17]). Each concept in the PSL ontology is specified with a set of definitions, relations, and axioms all formally expressed in KIF. Relations specify types of links between definitions or elements of definitions; axioms constrain the use of these elements. In addition, the PSL ontology is based on set theory, first order logic, and situation calculus [18]. For Grüninger, and because of this reliance on theories, every element in the PSL language can be proven for consistency and completeness [19].
All parts in the standard are independent of any specific process representation or model used in a given application. Collectively, they provide a structural framework for interoperability. PSL describes what elements should constitute interoperable systems, but not how a specific application implements these elements. The purpose is not to enforce uniformity in process representations. As objectives and design of software applications vary the implementation of interoperability in an application must necessarily be influenced by the particular objectives and processes of each specific application.
PSL currently aims at specifying technical processes for achieving interoperability among various software tool representations used throughout industrial companies. Although mappings to EXPRESS, XML and UML are planned in the future, PSL is not fundamentally based nor fundamentally makes use of Web services. As such, the approach followed, and the use of the language is different from the use of PSLX [20] and BPEL4WS [21].

4.1 Architecture and content

PSL (ISO 18629-1, 2004 [22]) is organised in a series of parts using a numbering system consistent with that adopted for the other standards developed within ISO TC184/SC4. PSL contains:
- **Core theories (Parts 1x):** PSL-Core, Outer Core, Duration and Ordering Theories, Resource Theories, Actor and Agent Theories ;
- **External Mappings (Parts 2x)**: mappings to EXPRESS, UML, XML and
- **Definitional extensions (Parts 4x):** Activity extensions, Temporal and state extensions, Activity ordering and duration extensions, Resource extensions, Process intent extensions.

Parts 1x and 4x contain the bulk of ISO 18629, including formal theories and the extensions that model concepts found in applications. Parts 1x are the foundation of the ontology, Parts 4x contain the concepts useful for modeling applications and their implementation. Table 1 presents primitive concepts found in PSL Core. Except noted otherwise, PSL version 2.2 is presented.

Table 1: Concepts in PSL Core (ISO IS 18629-11) [23]

PSL Core Primitives	Type	Informal definitions and axioms
activity	relation	Everything is either an activity, an activity occurrence, a timepoint, or an object. Objects, activities, activity occurrences, and timepoints are all distinct kinds of things (disjoint classes).

activity_occurrence	relation	An activity occurrence is associated with a unique activity. But there are activities without occurrences.
timepoint	relation	Given any timepoint t other than inf-, there is a timepoint between inf- and t. Given any timepoint t other than inf+, there is a timepoint between t and inf+.
object	relation	An object participates in an activity at a given timepoint and only at those timepoints when both the object exists and the activity is occurring.
before	relation	The before relation only holds between timepoints. It is a total ordering, irreflexive, and transitive relation.
occurrence_of	relation	Every activity occurrence is the occurrence of some activity and associated with a unique activity.
participates_in	relation	The participates_in relation only holds between objects, activities, and timepoints, respectively.
beginof	function	The beginning of an activity occurrence or of an object are timepoints.
endof	function	The ending of an activity occurrence or of an object are timepoints.
inf+	constant	Every other timepoint is before inf+.
inf-	constant	The timepoint inf- is before all other timepoints.

- Core Theories (Parts 12 - 15): The core theories are based on first-order logic ; they model basic entities necessary for building the PSL extensions. The PSL Core and Core Theories pose primitive concepts (those with no definition), function symbols, individual constants, and a set of axioms written in the language of PSL. Core theories are required to formally prove that extensions are consistent with each other, and with the core theories. The core theories are at the root of the PSL ontology against which every item that claims to be PSL compliant must be tested for consistency.
- Domain-specific Definitional Extensions (Parts 41 - 45): The extensions to the Core and Outer-core are the constructs used in PSL to represent processes in an application. The Core and Outer Core alone are not sufficient to meaningfully represent the semantics of applications for the purpose of interoperability. They are necessary but have little expressivity by themselves. All terms in the extensions are given definitions using concepts specified and axiomatised in the Core theories. This ensures that definitional extensions are conformant to PSL. A software application will typically use the concepts defined in the extensions.

4.2 Organising design information through the use of PSL

It is important to note that the terms defined in PSL are generic process related definitions and do not directly describe manufacturing processes [24]. The language provides an underlying rigorous basis which can used to provide non-ambiguous definitions for manufacturing processes and so there is a need to develop a dictionary of manufacturing process terms, based on PSL, which can be used to relate directly to manufacturing process information sharing. For example we can define a hole drilling process in PSL as follows:

```
(forall (?occ)
     (implies  (occurrence_of ?occ DrillHole)
        (exists (?occ1 ?occ2)
             (and   (occurrence_of ?occ1 Drilling)
                    (occurrence_of ?occ2 CentreDrilling)
                    (min_precedes ?occ2 ?occ1 DrillHole)))))
```

This provides us with a rigorous way of semantically defining a machining process for a hole. However, given the various contexts against which we can consider a hole there is still a need to be able to represent a hole from a given context. For example, a different semantics may need to be defined for each of the following contexts:

- Functional context => Purpose of the hole, surface finish, tolerances
- Geometry context => Dimensions of diameter and depth
- Manufacturing context => Generic manufacturing method
- Machining process context => Specific processes such as drilling, boring
- Assembly context => Tolerances, fit requirements

In this case not only does the semantics need to be defined but the relationships between the contexts need to be understood. The limitations of text based ontologies for the information organisation have

to be identified and we have to propose the use of mathematical rigorous foundation ontologies for the semantic definition of multiple contexts and their relationships. Further, libraries of influencing contexts and their relationships for the foundation ontologies can be defined based on enhanced PSL and UML methods. The utilisation of foundation ontologies for the information organisation that supports the interoperability is shown in figure 3 below.

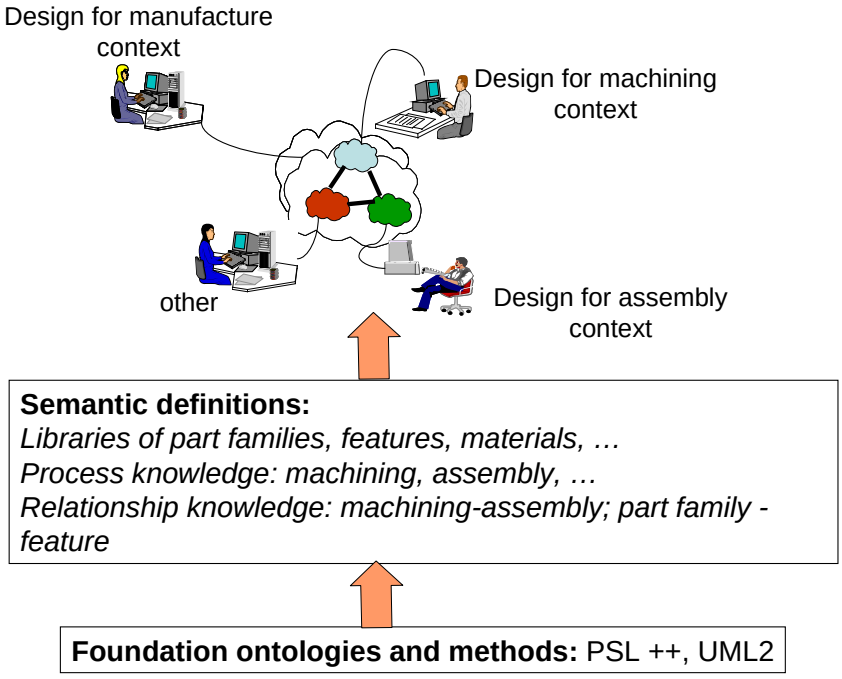

Fig 3: An information organisation based on foundation ontologies for interoperability

Another important use of formal ontologies is provided by the MDA approach: this approach is the subject of the following section.

5. AN APPLICATION OF FORMAL ONTOLOGY: THE MDA APPROACH

The MDA (Model Driven Architecture) defines an approach to IT system specification that separates the specification of system functionalities from the specification of the implementation of this functionality on a specific technology platform [25]. MDA defines a model architecture through the development of a set of guidelines for structuring specifications expressed as models [26]. The MDA approach and the standards that support it allow the same model functionality to be achieved on multiple platforms through auxiliary mapping standards, or through point mappings to specific platforms. It also allows different applications to be integrated by explicit relations between their models, thus enabling the integration, the interoperability and the evolution of supporting systems.

5.1 Basic concepts

Model-Driven Architecture begins with the idea of separating the specification of the operation of a system from the details of the way the system uses the capabilities of its platform [27]. MDA provides an approach for, and enables tools to be provided for: specifying a system independently of the platform that supports it, specifying platforms, choosing a particular platform for the system and transforming the system specification into one for a particular platform. The primary goals of MDA are portability, interoperability and reusability. The MDA concepts are presented in terms of existing or planned system. This system may include anything: program, single computer system, some combination of parts of different systems, federation of systems each under separate control, people, enterprise, federation of enterprises… The discussion focuses on the software tools within the system. A model of a system is a description or specification of that system and its environment for a given purpose. A model is often presented as a combination of drawings and text. The text may be expressed in a modelling language or using a natural language. MDA is an approach to system development. It increases the power of models. It is model-driven because it provides a means for using models to

improve the understanding, design, construction, deployment, operation, maintenance and modification.
The architecture of a system is a specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors [28]. Model-Driven Architecture prescribes certain kinds of models to be used, how those models may be prepared and the relationships between the different kinds of models. A viewpoint on a system is an abstraction technique using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within the system. Here 'abstraction' means the process of suppressing selected detail to establish a simplified model. Concepts and rules may be considered to form a viewpoint language. A platform is a set of subsystems and technologies that provide a coherent set of functionalities through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented.
This architecture defines a hierarchy of models from three different points of view: the Computation Independent Model (CIM), the Platform Independent Model (PIM), and the Platform Specific Model (PSM) [29].
The computation independent viewpoint focuses on the environment and the requirements of the system; the details of the structure are hidden or not yet defined. The platform independent viewpoint focuses on the operation of a system while hiding the details necessary to a particular platform. A platform independent view shows the part of the complete specification that does not change from one platform to another. A platform independent view may use a general purpose modelling language, or a language specific to the area in which the system will be used. The platform specific viewpoint combines the platform independent viewpoint with an additional focus on the detail of the use of a specific platform by a system.

5.2 Model transformation

Model transformation defines the process by which a model is converted to another model of the same system. The MDA process shows the role the various models (CIM, PIM and PSM) play within the MDA framework. A transformation tool takes a CIM and transforms it into a PIM. A second (or the same) transformation tool transforms the PIM into PSM. The transformation tool takes one model as input and produces a second model as output. Generally speaking, a transformation definition consists in a collection of transformation rules, which are unambiguous specifications of the way (a part of) one model can be used to create (a part of) another model. Based on those observations, it is possible to define transformations, transformation rules and transformation definitions [30], such as:
- transformation: automatic generation of a target model from a source model, according to given definitions ;
- transformation definition: set of transformation rules that together describe how a model can be transformed from the source language into a model in the target language ;
- transformation rule: description of how one or more constructs in the source language can be transformed into one or more constructs in the target language.
The main transformations are [26]:

- **PIM to PIM**: transformation used when models are enhanced, filtered or specialised during the development lifecycle without needing any platform dependent information. One obvious mapping is the analysis of design models transformations. PIM to PIM mappings are generally related to model refinement or model refactoring.
- **PIM to PSM:** used when the PIM is sufficiently refined to be projected onto the execution infrastructure. The projection is based on the platform characteristics. Describing these characteristics is done using a UML description (and eventually a profile for describing common platform concepts) within a Platform Description Model. The translation from a logical component model to a commercial existing component model is a kind of PIM to PSM mapping.
- **PSM to PSM**: transformation needed for component achievement and deployment. For example, component packaging is done by selecting services and preparing their configuration. Once packaged, the components delivery could then be done by specifying initialisation data, target machines, container generation and configuration, etc. PSM to PSM mapping are generally related to platform dependent model refinement.
- **PSM to PIM:** transformation required for abstracting models from existing implementations in

a particular technology into a platform-independent model. This procedure often resembles a "mining" process generally hard to make fully automated. It may be supported by tools. Ideally, the result of this mapping will match the corresponding PIM to PSM mapping.

There are several kinds of model transformation approaches: an interesting classification, based on design features, has been developed by Czarnecki and Helsen [31].

5.3 Application of the MDA approach to the design process

Figure 4 below shows a range of types of information supporting design for manufacture decisions. This information needs to be brought together to support design decisions. Past design decisions can be obtained from best practice methods, but this information is complex and influenced by many other kinds of information. We have focused here on the organisation of best practice methods into three kinds as design, manufacture, and service best practice methods to support the decisions on design for manufacture, design for service and other design activities. We have explored the information organisation particularly through two significant ways in which best practice methods can be captured; features and part families.

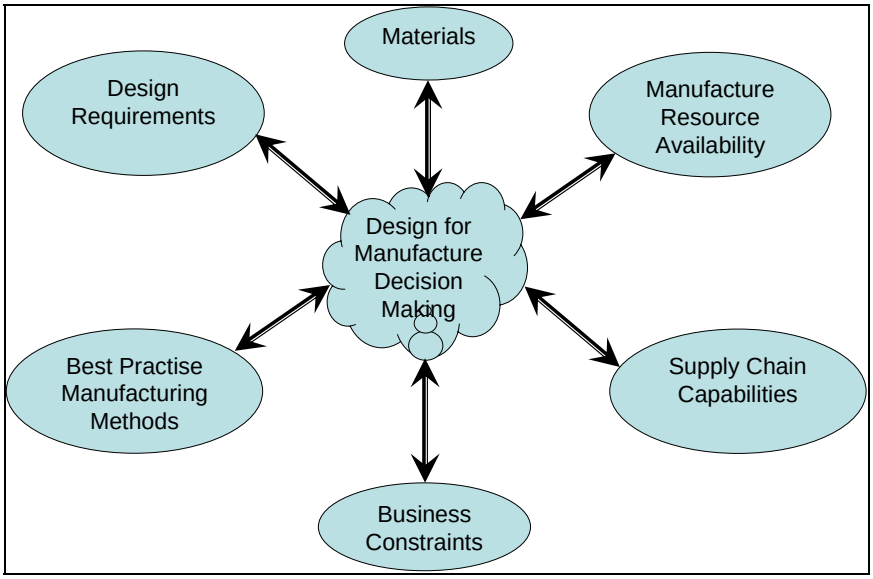

Fig. 4: Multiple kinds of information met in design

From our evaluation of the range of information needed for the capture of best practice methods we can design a class structure: Figure 5 below shows the top-level of the organised information for best practice methods based on feature and part family libraries represented in Unified Modelling Language (UML) notation.

Figure 5 proposes that best practice design, manufacture, and service methods can each be organised based on feature and part family libraries to support decisions. It also shows that best practice methods may be influenced by requirements, materials, business constraints, supply chain capabilities, and resources. However, it does not show the range of interdependencies which may constrain the relationships. For example, if we consider the required material information for a turbine blade of aircrafts in design, manufacture, service, and design for manufacture contexts may vary as follows:

- Design: Material properties to withstand high forces, stresses, and tip speed. Surface hardening may be required to withstand stresses and forces ;
- Manufacture: Manufacturability of fan from required material to required size. Shot peening method may be used to harden the surface ;
- Service: Serviceability such as the applicability of HVOF (High Velocity Oxy Fuel) coatings for the durability ;
- Design for manufacture contexts: this phase depends on the selected material and process; the geometry may require modifications. For example, if the shot peening process method selected to harden the surface, on Nickel material part requires the chamfer more than 0.3 mm while the Titanium part requires more than 0.5 mm chamfer.

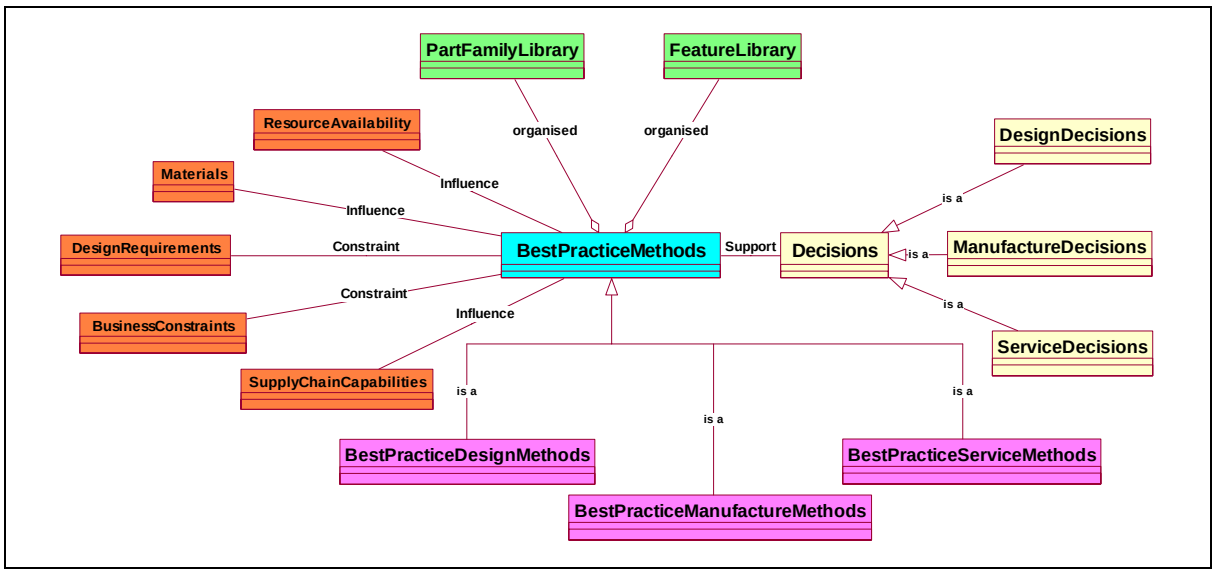

Fig. 5: UML schema of the top-level information for best practice methods based on feature and part family libraries

The complex relationships between contexts are shown in figure 6 below. The information content is not the same for these contexts, but the information system needs to be designed to support each set of requirements. Further, this problem exists at many different levels. For example, with the manufacturing phase, machining, assembly, and inspection each have their own information requirements. We therefore need significantly improved modelling methods if we are to successfully organise information to capture the range of information needs, their relationships and constraints.
However, the traditional UML class diagrams do not readily support the representation of multiple context information and their relationships. One potentially attractive methodology proposed by the Object Management Group (OMG) is UML-2 which provides an Object Constraints Language (OCL) [32] to apply context constraints to objects instances. UML-2 is a part of MDA approach, which aims to tackle the context dependencies and semantic problems.

6. DISCUSSION - ISSUES

A real test case has been developed by the Task Group (TG2) of the INTEROP Project, with the aim of providing a model-driven method that could guide enterprises in using enterprise models to generate Enterprise Software Applications (ESA) as automatically as possible, and more particularly to parameterise Enterprise Resource Planning (ERP) systems. This research focused on the difficulties enterprises face to generate ESA from enterprise models and how a model-driven approach could provide a useful way to solve them, while enabling interoperability at the same time. The Task Group also developed different enterprise models [33] for the case study, at different levels of abstraction and a first set of experiences of model transformations at the vertical level.
Based on this work, a first method to parametrise ERP from enterprise models using model transformations in a vertical approach has been proposed [34]. The approach proposed focused on establishing an interoperability model enabling model transformation between CIM into PIM level not only in a vertical way but also horizontaly, with as many ontology models (ontologies) as necessary. Fig. 7 shows the evolution of the initial idea of TG2 towards the use of only one ontology model. At this stage the objective of the work is to provide an interoperability model (meta-level) that can be connected to the parameterisation of ERPs using the Model Driven Interoperability approach.
Current developments made by the authors focus on the integration of concepts provided by the PSL language into the MDA approach, thus facilitating, through the use of a sound and well-defined ontology-based approach, the research of correspondences between the different ontologies applicable to the numerous activities of the design process.

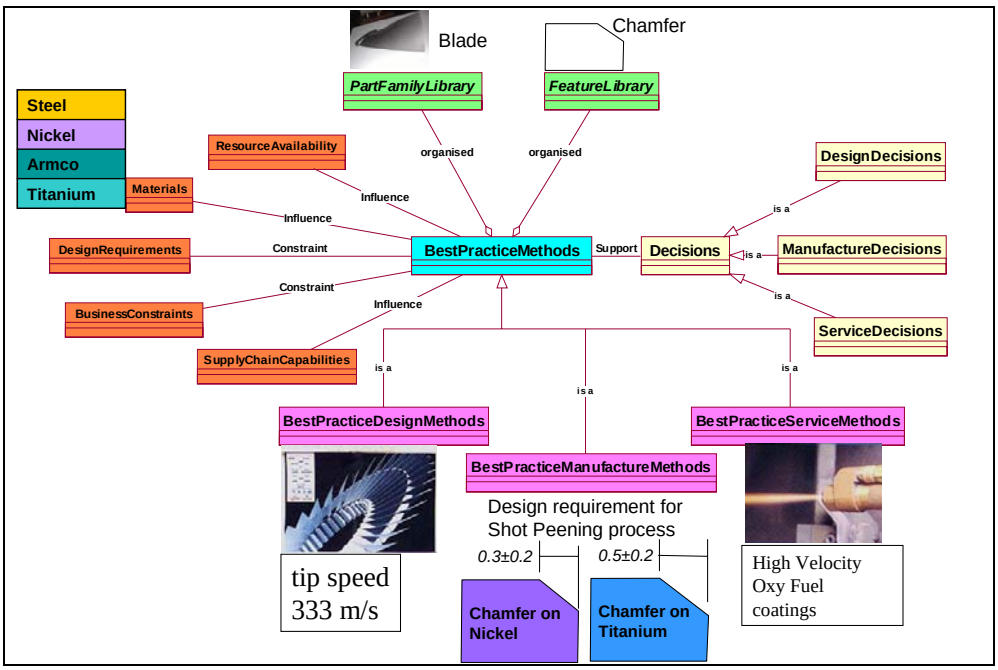

Fig. 6: complex relationships between different contexts

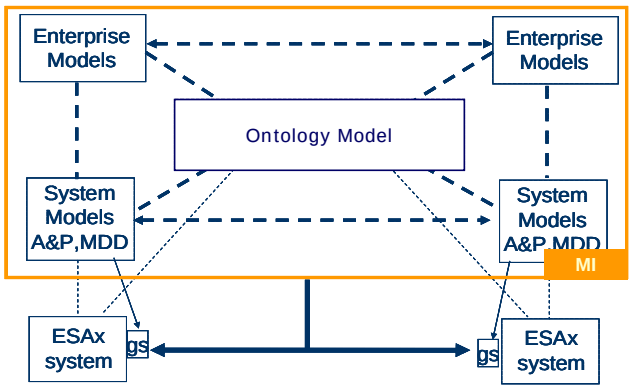

Fig. 7: Model Interoperability: vision targeted by the TG2 [29]

ACKNOWLEDGEMENTS

The authors are thankful for the financial support provided by the UK Engineering and Physical Sciences Research Council (EPSRC) and the Economic and Social Research Council (ESRC) under grant numbers EP/C534220/1, GR/R64483/01, EP/E002323/1 and RES-331-27-0006, without which this research could not have been done. The authors would also like to thank their colleagues on the "Knowledge and Information Management" project who have contributed to the discussion of the topic. The INTEROP project mentioned in this paper is funded by the European Commission within the 6th Framework Programme, Interoperability Research for networked Enterprises Applications and Software (INTEROP NoE), IST-2003-508011.

REFERENCES

[1] R.I.M. Young, O. Canciglieri-Jnr C.A.Costa M. Dorador, J. Zhao, W. M. Cheung, "Information Support in an Integrated Product Development System" in integrated Design and Manufacturing in Mechanical Engineering, Chedmail, Cognet, Fortin, Mascle, Penga (eds) Kluwer Academic Publishers, 2002, pp 139 - 146, ISBN 1-4020-0979-8

[2] Fowler J, "STEP for data management, exchange and sharing". (UK, Technology Appraisals), ISBN 1871802369, 1995.

[3] Krause, F. – L., Kimura, F., Kjellberg, T. and Lu, S. C. – Y. "Product Modelling", Annals of CIRP, Vol. 42/2, pp. 5-12, 1993.

[4] L.C. Pouchard, A.F. Cutting-Decelle, J.J. Michel, M. Gruninger, ISO 18629 PSL: A standardised language for specifying and exchanging process information, IFAC Conference, Praha, July 2005

[5] Pouchard, L., N. Ivezic, C. Schlenoff, Ontology engineering for distributed collaboration in manufacturing. AIS, Tucson, Arizona, 2000

[6] Pouchard, L., O. Rana, The Role of Ontologies in Agent-oriented Systems. *Sixth Joint Conference on Information Sciences, Computational Semiotics Workshop*. Research Triangle Park, North Carolina, 2002
[7] Ray S. R. and A. T. Jones, Manufacturing interoperability, Concurrent Engineering: Enhanced Interoperable System. *Proceedings of the tenth ISPE International Conference*, pp.535-540, 2003
[8] Gruber, T., A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisitions* **5**, 199-220, 1993
[9] Gomez-Perez, A., in: *The Handbook of Applied Expert Systems* (J. Liebowitz, Ed.), Knowledge Sharing and Re-Use. Chapter 10, pp.. 1-36. Boca Raton, Florida, 1998
[10] Hunhs, M. N. and M. P. Singh, Ontologies for Agents. *IEEE-Internet Computing* **1: 6,** 1997
[11] Fikes, R. and A. Farquahr, Distributed Repositories of Highly Expressive Reusable Ontologies. *IEEE Intelligent Systems and their Applications*, **14:2,** 1999
[12] Gomez-Perez, A., Fernandez-Lopez, M., and Corcho, O. (Eds), Ontological Engineering, 2004, (Springer-Verlag) ISBN 1-85233-551-3.
[13] Casely-Hayford L., Environments, Methodologies and Languages for supporting Users in building a Chemical Ontology, A dissertation submitted to The University of Manchester for the degree of MSc Bioinformatics in the Faculty of Life Sciences, 2005, http://epubs.cclrc.ac.uk/bitstream/896/Dissertation.pdf
[14] Gangemi A., Grimm S., Mika P., Oberle D., Lamparter S., Sabou M., Staab S., and Vrandecic D., Core Ontology of Software Components Core Ontology of Services, 2005, http://cos.ontoware.org/
[15] Hristozova M., and Sterling L.,. An eXtreme method for developing lightweight ontologies, In Stephen Cranefield, Tim Finin, and Steve Willmott, editors, *Proceedings of the Workshop on Ontologies in Agent Systems (OAS 2002) held in conjunction with the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS03)Republished in CEUR Workshop Series*, Bologna, 2002.
[16] Genesereth, M. and R.E. Fikes, Knowledge Interchange Format, Version 3.0. Reference Manual. Technical Report Logic-92-1. Computer Science Department, Stanford University, Stanford, CA, 1992
[17] Common Logic Web site. Available from http://philebus.tamu.edu/cl/.
[18] Etchemendy J., The language of first-order logic. *CSLI Lecture Notes*, **34,** 1992
[19] Gruninger, M., in: *Hand-book on Ontologies in Information Systems (* R. Studer and S. Staab Eds.), A guide to the ontology of the Process Specification Language. Springer-Verlag, Frankfurt, 2003
[20] PSLX Consortium, Available from: http://www.pslx.org/en/. 2005
[21] BPEL4WS: Available from http://www-128.ibm.com/developerworks/library/specification/ws-bpel/
[22] ISO IS 18629-1, Industrial automation systems and integration – Process specification language – Part 1: Overview and basic principles, 2004
[23] ISO IS 18629-11, Industrial automation systems and integration – Process specification language – Part 11: PSL-Core, 2004
[24] A.G. Gunendran, R.I.M. Young, A.F.Cutting-Decelle, J.P. Bourey Organising Manufacturing Information for engineering interoperability, I-ESA 2007
[25] R. Grangel, C. Métral, A.F. Cutting-Decelle, J.P. Bourey, R.I.M. Young, Ontology based communications through model driven tools: feasibility of the MDA approach in urban engineering projects, Lecture Notes in Computer Sciences LNCS, Springer, 2007
[26] Model Driven Architecture (MDA), Architecture Board ORMSC, doc n° ormsc/2001-07-01
[27] MDA Guide Version 1.0.1, OMG, omg/2003-06-01, 2003
[28] Shaw and Garlan, Software Architecture, Prentice Hall ISBN 0-13-182957-2
[29] Report on Model Interoperability, INTEROP, TG2, Deliverable D2.2, 2006
[30] Kleppe A., Warmer J., Bast W., MDA explained: the Model Driven Architecture: Practice and Promise, Addison-Wesley, 2005
[31] Czarnecki K. and Helsen S., Classification of Model Transformation Approaches, OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, 2003
[32] OCL: Available from http://www.omg.org/technology/documents/formal/ocl.htm
[33] Report on Model Establishment, INTEROP TG2, Deliverable D2.1, 2006
[34] Grangel, R., Bourey, J.P. & Berre, A., Solving Problems in the Parameterisation of ERPs using a Model-Driven Approach. In: Interoperability for Enterprise Software and Applications Conference (I-ESA'06), 2006

Contact: A.F. Cutting-Decelle
Ecole Centrale de Paris / LGI, Grande Voie des Vignes,
92295 CHATENAY MALABRY, F
Email: afcd@skynet.be