Ontologies as an Interface between Different Design Support Systems

MINNA LANZ, TIMO KALLELA, EEVA JÄRVENPÄÄ, REIJO TUOKKO
Department of Production Engineering
Tampere University of Technology
P.O.Box 589
FINLAND
Minna.lanz@tut.fi  http://www.tut.fi/tte

Abstract: - The aim of the present paper is to shortly review the problem field the companies are facing as the information should flow between different design support systems and introduce an approach to solve some of the arising problems. The CoreOntology forms the information structure for the Knowledge Base (KB) and standardizes the communication interface for the design support systems. In the present paper, an approach is proposed to share platform independent product, process and system related knowledge between the different design support systems through the KB.

Key-Words: - Simulation, Design Support Systems, Ontology, Knowledge Management, Knowledge-Intensive Models

1 Introduction

Global competition and economical changes have caused manufacturing processes to become more and more distributed around the globe, while design teams may operate around the clock. In such a geographically and temporally divided environment, effective and proficient collaboration between design teams is crucial to maintain product quality, production efficiency and organizational competency.

The process of designing a manufacturing system, such as an assembly line, is typically a joint venture, including several companies, and various persons from different domains – product design, machine design, robotics, process planning, maintenance, etc. The importance of knowledge exchange among the design teams and engineers has been widely recognized in the industry.

The requirement for the collaborative information systems is the need to cope with the ever-faster innovation cycle in which new processes, models and systems are being developed. There are numerous different collaborative design support systems in the markets; unfortunately those offer little or no help in the re-use and distribution of knowledge.

The second awakening of XML-based formats with the emergence of ontological definitions along the semantic web have provided means for transferring expert knowledge into such form that can be reused by many, at any given point in time.

Formalizing the knowledge representation with the ontologies and agreed semantics, the knowledge can be understood by both human and “intelligent” information systems. Ontological classification of manufacturing systems offers a great approach to standardize the way people, software and machines communicate, embedding the meaning and the context within the message being sent.

An early definition describes ontology as “a hierarchically structured set of terms to describe a domain that can be used as a skeletal foundation for a knowledge base”. The main features of ontology are machine readability and understandability which in turn enable cross-communication between different systems or domains. This leads to the increased use of platform independent design as well [1, 4]

A knowledge management system (KMS) for the design of intelligent re-configurable manufacturing systems has to provide tools for storing and retrieving tacit/non-hierarchical information and strict ontological classifications from different domains. The retrieval and sharing is not the only challenge but a KMS must be able to parse and format the retrieved information into meaningful knowledge that aids the generation of manufacturing systems with different design support systems [1, 4].

The following paragraphs will outline the theoretical approach – currently under development in the Framework Programme 6, IP-PISA project – to the feature-based product-process-system ontology, KB structures and interfaces for an enhanced simulation environment.
2 Problem Formulation

Currently companies are storing almost all of the information created via computers. The servers are saving, replicating and archiving the collected information non-stop. The amount of stored information today is simply to put; huge.

The storing of the information or size of the models are no longer the issues. The problem today is the meaning of the stored information.

In a large scale company there can be up to 100 different design support systems, versions and ad-hoc applications build in top of those which are used to create the information of the current product and/or production systems. All of the systems are using their proprietary data structures and semantics and none of those are truly able to share data beyond geometrical visualizations with each others.

This leads to several problems; firstly the communication between departments’ “designer domains” becomes time consuming since the models have to be created over and over again.

The second problem is that by every re-make and update the model actually loses information, because the second tier does not require all of the information created in the first phase. The result is that there are multiple sources contributing specific knowledge to several isolated models and revisions instead into “the master model”.

The third problem that follows these two is the re-use of the knowledge. At the moment the re-use of the existing information is quite impossible due to the several models which are meaningless from the knowledge-point of view.

For example the production planning knows that there are several stations, robots and grippers which can be re-used in the production of the next product family needing only a bit of modification. However, despite the amount of stored information, there are no up-to-date complete information set easily available of the interfaces and life-cycle-data and dimensions of the station and its components. The new station and the line are therefore designed starting with an empty layout.

There is a lot of potential for the re-use of the components. But once again the problem is that there is no definition, the ontology, which would explain what the system is, what its capabilities are and how long it has operated. And even if the ontology would exist, the modern CAD/CAPP systems which are currently the market leaders cannot utilize the existing information.

3 Problem Solution

The authors propose a following solution in order to take the first steps to obtain meaningful connected information, i.e. knowledge without requesting major changes on the used CAD/CAPP systems. Of course this approach will solve only partially the earlier discussed problems. The major effort is towards the “knowledge-intensive master model”.

3.1 Feature-Based Product Definition

Starting with the theoretical background, a product model is understood as a representation of product specific data, which describes a particular product or a product family. The computerized representation of product data requires a product data model, which defines the form and content, and a product model, which contains information specific to a particular product.

The product model contains a combination of single parts where different instances can be of the same type. Some of the parts are not always directly assembled to 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 to the main assembly. [3, 5]

Feature-based modeling and analysis is used to add meaning to the product data. Features can be either geometrical (such as edges, faces, holes, pockets, slots, ribs and pads) or non-geometrical product information (such as tolerances, material, density, handling requirements and color). The product model will include features from both of the categories.

In the PISA model the assembly specific feature information is divided into four layers; Features, FeatureSets, AssemblyFeatures and AssemblyFeatureSets. A feature can be a certain face or an edge. The feature sets are more complex combinations of recognized and meaningful faces. The assembly features include assembly process requirements. 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 base in relation to product knowledge [5].

The knowledge model defines how the Process Model, Reasoning Machine, Reasoning Results and Product Model are integrated. The product specific information is connected into the processes through the two classes: Reasoning Machine and Reasoning Results [6].

3.2 Definition of the CoreOntology
In the field of philosophy the word ontology is defined as “the structure of a being”. In past decade the word ontology has become very important to the realm of the Knowledge Engineering. Today ontology is seen as a valuable tool when different manufacturing enterprise systems are needed to interoperate. The ontology can capture the consensual knowledge in a generic way that can be re-used and shared across the enterprise applications [9].

In a global manufacturing environment there are multiple design support system clients, which have very different understanding of semantics or values. Even when operating in a same field, personal aspects of the same concepts cause conflicts in transferring knowledge.

The solution that ontological modeling provides is simplification and categorizing of the knowledge into a more typified but still flexible form. Each of the actors still may preserve their perception of the knowledge but by using common terms and rules to model the knowledge in the transferring process, huge advantages are achieved.

The client-independent knowledge still readable and understandable by the clients can be archived by using semantical mapping, in-which the semantical models of the clients exported data structures are transformed into a generalized knowledge model i.e. CoreOntology. Unfortunately, a great risk is involved within semantical mapping, 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 the knowledge model to another requires much flexibility from the mapping process, but can be reached within some limitations and by realizing that modeling everything is not required to increase productivity [7].

As stated above, modeling of all the information available is not needed, and wisely used interfacing provides very general but flexible knowledge model which operates as a medium for various reasoning processes.

The knowledge stored in this kind of model is not required to be “complete” or even fully compliant with the existing knowledge. The point is in process where each actor modifies data from their point of view and by doing so increase the value of the knowledge.

The sections of the CoreOntology are modeled with Protégé v3.4 and visualized with GraphViz v2.17.

3.2.1 Product Definition

Product section in the CoreOntology, introduced in figure 1, is used for describing and modeling the product specific information inside the knowledge base. The product ontology supports variable type of product structures with rich meta-information. Additional information, if needed, is presented as non-geometrical features of the object and the lifecycle related information.

![Fig.1. Product Definition in CoreOntology](image)

In the product ontology the Part, SubAssembly and Assembly class all have a connection to the Product class, while the Product class defines the geometrical and non-geometrical features related to the part, subassembly or product. This structure will allow more freedom to describe different detail levels of the current product.

The product ontology also contains information of the life-cycle phases of the current product. The life-cycle phase information will define whether or not the part or a product is allowed to be used in the simulation.

3.2.2 Process Definition

Manufacturing processes are described with process ontology, see figure 2. The process ontology is the key operator when combining the product and the system knowledge.

![Process Definition](image)
The process ontology defines the activities required by the product via skills and actors from the system ontology. The process class is the root class in this domain. Processes are defined, for instance, 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, screwing, gluing, insert and press-fit. The Action and SubAction classes are reserved for the most basic functions; rotate and translate and combinations thereof. [6]

3.2.3 System Definition

The third ontology section describes the resources as illustrated in figure 3. 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 production equipment, devices and related software, are defined via product ontology.

Currently the system ontology is linked to the process ontology by agent based skill classes for controlling operations in the manufacturing domain.

3.3 CoreOntology+OWL

At the beginning of the previous decade, ontologies were built using mainly AI modeling techniques based on frames and first order logic (FOL) utilizing languages such as Ontolingua and KIF. Since then, other knowledge representation techniques based on description logics (DL) were used to develop ontologies and new DL languages such as Ontology Interchange Language (OIL), DARPA Agent Markup Language (DAML+OIL) and web ontology language (OWL) have emerged [9].

Figure 4 illustrates the OWL-file which is created from the ontology.

3.4 Test Case

An example of the different clients using and contributing their knowledge can be a loose integration between process-planning and manufacturing simulation.

Fig. 2 Process Definition in CoreOntology

Fig. 3 System Definition in the CoreOntology

Fig. 4 OWL for defining the Activity and Capability classes

Fig. 5 Process Planning and Populated Simulation

In the example described here, see figure 5, process-planning has no verified information of the actual
time spend on the operations, but by entering the
process-planning knowledge to KB. The simulation
tool will retrieve specific information from the KB
and populates the simulation environment in order
to validate time-cost estimation and complete the
existing model with the results.

The ideology behind this example is to give a
method to each entity to save the vital information
from their point of view and to retrieve some
knowledge in which they would not have access
otherwise.

The case example will require the Process
Planning client to create a middleware for exporting
and importing the knowledge back and forth to the
KB. The requirements for the simulation tool are
similar. Also the simulation environment needs to
parse the information into actual 3D simulation.
Some of the required reasoning will later be done
inside the KB.

### 3.5 An Interface to CoreOntology and
Access to the KB

The knowledge stored in the KB can be accessed
through a designated middleware using interfaces.
The figure 6 lists the get() functions defined at the
moment.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getProducts()</td>
<td>Returns all the products in the Knowledge base.</td>
</tr>
<tr>
<td>getSubAssembly(parent)</td>
<td>Returns all the subassemblies directly under the parent object.</td>
</tr>
<tr>
<td>getParts(parent)</td>
<td>Returns all the parts of the parent object.</td>
</tr>
<tr>
<td>getParameter(object, key)</td>
<td>Returns a parameter identified by key variable and owned by object.</td>
</tr>
<tr>
<td>getActivities()</td>
<td>Return all the activities related to object / feature / feature.</td>
</tr>
<tr>
<td>getRequiredResources()</td>
<td>Returns resources required.</td>
</tr>
<tr>
<td>getRequiredSkill()</td>
<td>Return skill required.</td>
</tr>
<tr>
<td>getProcesses()</td>
<td>Returns the top level activities, e.g., Manufacturing of X.</td>
</tr>
<tr>
<td>getWorkActivity(activity)</td>
<td>Returns the next activity in a sequence. If no next defined, returns the previous activity.</td>
</tr>
<tr>
<td>getPreviousActivity(activity)</td>
<td>Returns the previous activity in a sequence. If no previous defined, returns the next activity.</td>
</tr>
<tr>
<td>getChildActivities()</td>
<td>Returns collection of all child activities of entered activity.</td>
</tr>
<tr>
<td>getParentActivity()</td>
<td>Returns the parent activity of entered activity.</td>
</tr>
<tr>
<td>get.ActivityTarget(activity)</td>
<td>Returns the objects that activity target, e.g., part in moving activity.</td>
</tr>
<tr>
<td>getResources()</td>
<td>Lists all the resources in KB.</td>
</tr>
<tr>
<td>getActivities()</td>
<td>Lists all the activities in KB.</td>
</tr>
<tr>
<td>getRequiredResources()</td>
<td>Returns collection of resources required to use the resource.</td>
</tr>
<tr>
<td>getRequiredSkill()</td>
<td>Returns collection of skills required to use the resource.</td>
</tr>
<tr>
<td>getPosition(resource)</td>
<td>Returns the position of the resource in production line, e.g., returns the next area position in the layout.</td>
</tr>
<tr>
<td>getLeaf(resource)</td>
<td>Returns sub-leaf objects of selected location resource, e.g., cell.</td>
</tr>
<tr>
<td>getFeatures()</td>
<td>Returns all the features of a feature set.</td>
</tr>
<tr>
<td>getFeature(feature)</td>
<td>Returns all the features related to feature.</td>
</tr>
<tr>
<td>getFeatures()</td>
<td>Returns all the features in Knowledge Base.</td>
</tr>
<tr>
<td>getPreviousPosition(resource)</td>
<td>Returns the previous area position in the layout.</td>
</tr>
<tr>
<td>getFeatureType()</td>
<td>Returns the type of the feature.</td>
</tr>
<tr>
<td>getFeatureCollection()</td>
<td>Returns the collection of lifecycle engines in Knowledge Base.</td>
</tr>
<tr>
<td>get.Transition()</td>
<td>Returns the lifecycle engine transition.</td>
</tr>
<tr>
<td>get.Activity(Specification)</td>
<td>Returns the lifecycle phases that are related to the specification.</td>
</tr>
<tr>
<td>get.Parameter()</td>
<td>Returns all the parameters related to the object.</td>
</tr>
<tr>
<td>get.ActivityPhase()</td>
<td>Returns the current lifecycle phase of the object.</td>
</tr>
<tr>
<td>get.ActivityPhase()</td>
<td>Returns the next lifecycle phase in the process.</td>
</tr>
<tr>
<td>get.ActivityPhase()</td>
<td>Returns the previous lifecycle phase in the process.</td>
</tr>
<tr>
<td>get.ActivityPhase()</td>
<td>Returns the lifecycle engine related to the activity.</td>
</tr>
<tr>
<td>get.ActivityPhase()</td>
<td>Returns a single document.</td>
</tr>
<tr>
<td>getGeometry()</td>
<td>Returns a geometry of a single feature.</td>
</tr>
<tr>
<td>getGeometry()</td>
<td>Returns a geometry of a single feature.</td>
</tr>
<tr>
<td>getGeometry()</td>
<td>Returns a geometry of a single figure.</td>
</tr>
</tbody>
</table>

Figure 6. Get() functions of the interface

The interface is consist of two kinds of methods:
fundamental methods and reasoning methods. The fundamental methods are simple methods only
getting and setting instances or parameters from the
existing knowledge model. These methods are used
to simplify the access to the knowledge provided/requested by the clients.

The fundamental methods are listing methods,
retrieval methods and setting methods. Listing
methods return the array of objects feasible for a
given filter. Retrieval methods can get individual
objects or parameters from the knowledge model
and methods for setting do the opposite.

The reasoning methods are meant to support
client's activities by giving easy access to the
already reasoned information. These methods are
mainly for retrieving information.

### 3.6 KB and A Client for Viewing the
Knowledge Stored in CoreOntology

A Knowledge system platform should be a neutral
system, which is independent from any specific
applications. This is indispensable in order to fulfill
the objective of the product and process knowledge
system. The system platform encapsulates all
specific parts of a control system in order to provide
a neutral interface to the application software. The
system software, which provides the functionality
needed to support openness for the application
software, should be located on the top of the
platform. Knowledge acquisition is also accessed
by humans, as part of a job that doesn’t require
previous experience of knowledge based systems [2,
8].

For storing and accessing to the product
knowledge a certain set of knowledge management
tools is needed. The authors propose an approach to
form a distributed knowledge base and control tools
based on Service-Oriented Architecture (SOA) and
Event-Driven architecture introduced in [6].

The KB has three main goals; the first is to act,
in the simple case, as a PDM (Product Data
Management) vault. The second goal is to provide a
standardized interface between different design
support system clients. The clients are able to share
information more efficiently since the native
formats are mapped into the CoreOntology. For the
clients the only requirement is to provide the
interface to the KB.

The third goal is to provide knowledge reasoning
and parsing based on the defined rules, mating
conditions and earlier solutions saved on the KB.
An example of reasoning is that the product has a
set of screws in the product structure, which will set
requirements for the process and the system
definition.

In order to inspect the knowledge stored into the
KB, a viewing client was developed, see figure 7.
The client uses web services to get the product information. At the moment the reading functionality is implemented. At the next stage of the development this client will also be able to modify the stored information and create new instances.

Fig. 7 KB client for viewing the product information

4 Conclusion
In summary, the ontological approach presented in this paper aims to provide a solution for the problem of knowledge representation and sharing between the product, process, and system design domains. The Knowledge Base will have three main goals; to serve as an interface between different applications, to perform reasoning and to store knowledge in structured format.

The authors are confident that the ontologies together with reasoning methods form the most efficient way to acquire the knowledge and share it internally as well as externally between different clients.

Nevertheless the reality is that the system providers are not agreeable to export the knowledge created in their proprietary systems with the possible competitors, even the customers are requesting the feasible knowledge distribution.

Another major challenge is to provide a robust and acceptable solution for knowledge capture from different sources. Overcoming the challenges in inter-operability and re-use of components and declarative knowledge are crucial for the further development of the knowledge-based system. 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 definition of declarative knowledge structures) assumed by the various components. [2, 6]

References:
[9] PABADIS PROMISE, Development of Manufacturing Ontology PABADIS PROMISE Ontology (P2 Ontology), 2006