Using ontologies to support decentral product development processes

Patrick D. Stiefel[1], Christian Hausknecht[1] and Jörg P. Müller[1]

[1] Clausthal University of Technology, Department of Informatics
{patrick.stiefel, christian.hausknecht, joerg.mueller}@tu-clausthal.de

Abstract. Adaptive and open platforms for cross-organizational collaborative product development (CPD) need flexible architectures and network solutions as well as novel data integration concepts supporting distributed, decentralized collaboration. Previous approaches to solving this problem have largely ignored the requirement of providing interoperable formats for product model data that enable the support of collaborative product development activities. This paper proposes the use of ontology technology to address this problem; it presents the integration of ontology technology into existing model-driven approaches to product development, and evaluates the applicability of the approach by describing a use case with limited CPD platform functionality.

Keywords: Decentral and collaborative product development (CPD), Peer-To-Peer based collaboration, Ontologies in a distributed collaboration environment, Model-driven development of decentral organized information systems.

1 Introduction

Cross-organizational, collaborative Product Development (CPD) is the state-of-the-art approach to support knowledge sharing in multi-party cross-organizational engineering projects [1]. To support CPD processes, new collaboration platform technologies are required. They should provide an added value at product definition and execution stages by reducing collaboration complexity and obstacles.

There are many CPD platform design recommendations that mainly focus on data sharing and mapping. These recommendations address one of the key problems of product development: the integration of heterogeneous product model definitions [2]. What still remains is the question on how to accelerate the product development processes, especially in the early phases of the product lifecycle where the major challenge is to efficiently share rapidly changing product design approaches among development teams [3].

These requirements result in new design strategies for CPD platforms. Existing client-/server-based approaches are too inflexible to support loosely coupled, ad-hoc collaboration situations. Therefore, our research focuses on developing decentral information technologies to support CPD processes [4,5]. One of the results is the Product Collaboration Platform (PCP), an experimental peer-to-peer (P2P) software platform to support decentral product development processes. As presented in [1], we follow a model driven development (MDD) approach to design information systems

for decentral and collaborative product development (DeCPD). Based on Computation Independent Models (CIM) we employ an iterative process to develop different abstractions of IT models: starting from IT architecture models over platform specific models (PSM) to concrete software artifacts. We employ an agent-based approach to model the distributed collaboration logics, as is explained in Section 2.1.

When developing distributed systems for knowledge sharing there is a need to provide a common language. Product developers act in different languages and normally have unequal action courses and best-practices when doing their model design activities. A common global CIM process model constrains the user in his process sequence but it does not explain the objective of the collaboration. This results in a lack of interoperability. To deal with model-centric processes, we have to introduce a common formal language that defines the intent of each collaboration step. Ontologies give us the possibility to enhance decentral information knowledge sharing with semantics. Collaborations based upon a semantic data model provide the possibility to understand a collaborations partner design requirements (*specification*) and to answer in an understandable format (*proposal*) without changing local product data management strategies. Thus, ontologies are an important building block to achieve interoperability between distributed model repositories and DeCPD processes. Last but not least we are able to underpin the suitability of the model-driven approach [6].

The structure of this paper is the following: After introducing DeCPD and ontology concepts in Section 2, we will discuss some related work experiences in Section 3. Section 4 deals with DeCPD CIM level models to provide a technology independent need for concepts like ontologies. Section 5 with corresponding PSM models and introduces in our ontology concepts. In the last section we evaluate our approach by providing a reference scenario that we implemented in our Product Collaboration Platform (PCP). The scenario depicts on a simple LEGO™ building example.

2 Background

2.1 Model-driven decentral and collaborative product development

Each DeCPD process describes a distributed solution of a given product engineering problem (*specification*). Our approach is based on the Distributed Problem Solving developed for multi-agent systems [7, 8]. The DeCPD process provides a synthesis of the distributed partial solutions of the participants (*proposals*) to an overall solution satisfying the initial requirements set up by the initiator.

The engineering problem in our work can be represented as the search of a set of product model (PM) components matching the specification. Product engineers are normally confronted with this problem in the design phase of the product lifecycle. This view enables us to conceptualize a *PM Specification* by a query that describes features, which the target component has to fulfill; a corresponding *PM Proposal* represents one possible design solution.

As described in the introduction modern approaches to collaboration platforms are needed to support the engineering problem described above. In designing a DeCPD collaboration platform, we follow a model-driven top-down approach. To model the underlying collaboration process we start with **C**omputation **I**ndependent **M**odels (**CIM**) that describe the functionalities of the platform on the functional level. We use the Business Process Modeling Notation (BPMN) to express CIM models in a language suitable for the audience (i.e. engineers).

The result of performing this process are decentral architectures at **P**latform **S**pecific **M**odel (**PSM**) level. We design Business Process Execution Language (BPEL) workflows with especial architecture elements to cover the requirements given by decentral information systems subject to our work. Our BPEL workflows are model-centric; they cover event-driven trigger patterns (such as publish-subscribe) from the P2P overlay network.

The focus of this paper is not on the development of MDD models but on their application combined with the use of ontologies that provide the flexible and scalable approach required for collaborative product engineering platforms. In doing so, we aim at

- supporting ad-hoc interconnections between world-wide distributed partners that often did not collaborate in the past and
- effectively distributing product models among participating engineers for load balancing reasons, and a more efficient execution of product development processes for the reasons of task sharing.

2.1 Ontologies

We propose using ontologies to represent product model data and metadata. We will give a short introduction into the core concepts of ontologies and related technologies.

We use the web ontology language (OWL[1]) which is standardized by the W3C. OWL is based upon the resource description framework (RDF[2]), which is also a W3C standard. RDF enables the linking between objects, so called RDF Resources. This linking can be described by a directed graph, where nodes represent resources and the edges represent named links. This graph structure is also called *triple*. OWL itself defines some important concepts upon RDF:

- **classes** are similar to sets as they group together individuals having the same properties. For example, a class represents “Lego building bricks” or “plates”.
- **properties** describe the connection between individuals or the assignment of data values to them. For example, an individual of the class *brick* could have a property *hasWidth* which holds a value of “24”.
- **individuals** are instances of classes like in class-based programming languages (Java, C++, C# e.g.).

OWL provides a wealth of additional constructs, but these are the most important ones and sufficient for understanding this paper. Fig. 1 shows a simple example: Two

[1] http://www.w3.org/TR/owl-features/ [19th February 2010]
[2] http://www.w3.org/RDF/ [19th February 2010]

individuals (*Tuc01* and *Tuc02*) belong to the class *Brick*. *Tuc01* has a property *isConnectedWith*, which uses *Tuc02* as endpoint. Furthermore it has another property *hasWidth*, which assigns the data value "24" to it.

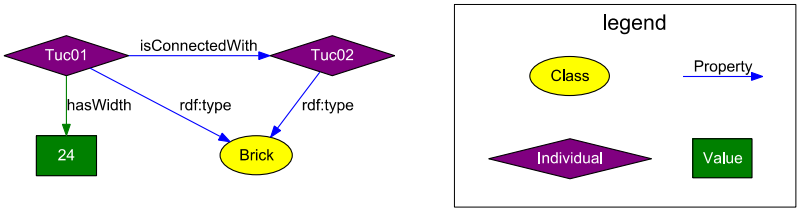

Fig. 1. OWL example

Real world ontologies are much more complex and much harder to understand and validate. For that reason, techniques are required to query triples. One query language to provide this functionality is SparQL[3], which is also a recommendation by the W3C. SparQL has much similarity with SQL. SparQL enables to select arbitrary subsets (subgraphs) from a given RDF graph that fulfill the users constraints (= graph pattern). Furthermore, a SparQL query specifies which elements from the result should be returned. A SparQL query has the general form:

```
SELECT ?subject ?predicate ?object
WHERE {
    ?subject ?predicate ?object.
}
```

After the `SELECT` statement, elements that should be returned by the query are defined, no matter if these are individuals, classes, or properties. Within the `WHERE` clause, we define the graph pattern that the query engine has to match.

```
SELECT ?item
WHERE {
    ?item   rdf:type            Brick.
    ?item   isConnectedWith     ?other_item.
}
```

The following example (see Fig. 1) shows the selection of an individual (from the given ontology), the type of which is *Brick* and that has a relationship to another item, specified by the property *isConnectedWith*. The following SparQL query will return the name "Tuc01".

[3] http://www.w3.org/TR/rdf-sparql-query/ [19th February 2010]

3 Related Work

There are existing research approaches related to the topic of CPD that use semantic descriptions. We observe that there are mainly two ways how ontologies are used in CPD: (1) as a formal description for product model content, or (2) as a structural base that enable a mapping between several product data formats.

3.1 Ontologies as formal product model description

Kim [9] proposes the usage of ontologies for adding a semantic description to product model representations so that it possible to specify the meaning of design challenges. The author introduces an ontology-based specification of an assembly design, so called AsD. Based on this AsD, a product engineer can describe the topology of assemblies and their joining relations, mainly focusing on spatial, geometric characteristics. With computational reasoners it is possible to infer the meaning of an assembly connection. This helps developers understand the meaning of design decisions within a CPD process.

Similar to Kim's approach, Liang [10] studies the description of connections between LEGO™ objects due their given geometric structure. He also implements an ontology using OWL, that describes these connections (he calls them "ports") in order to use it as a tool to find possibilities for connecting assemblies or to prove their validity.

In [11], Mostefai describes a generic and extensible product ontology especially designed to use in the area of mechanics. This is mainly used to facilitate a common understanding among different people such as engineers with their CAD/CAM experience, production planners, IT technicians, etc. If all these people accept a "common ontology", they can contribute to a unified product model.

3.2 Ontologies as a support for product model data exchange

Another use case for ontologies in CPD is the need to develop a platform-independent interchange format for product data. In [2], Patil, Dutta and Sriram propose an ontology-based framework to handle this task, called "Product Semantic Representation Language" (PSRL). They claim that the traditional solutions based upon industry standards for product data exchange (e.g. STEP, **S**tandard for **t**he **E**xchange of **P**roduct Model Data) are limited by the fact, that the semantics are ignored, which may lead to loss of information at transformation processes. They further claim that a better mapping solution can be achieved with PSRL that expresses semantics of a product model and not just plain parameters and values.

To conclude, what we did not find in existing ontology approaches yet is to use ontologies as semantic description in a decentrally organized product engineering process. That is what we need in our work: The cross-organizational process state has to be determined from the product model and that would be impossible when not using ontologies to define semantic information and process coherences upon flat

product model data. As a result, in our approach; we can allocate IT modules the capability to interpret intermediate collaboration results and to make the right decisions until the final result is reached.

4 DeCPD CIM level models

In this section we give an overview of several model design aspects at CIM level. This should help to understand the main motivation when designing a DeCDP platform.

4.1 CIM: Process aspect

Using BPMN we build generic, cross-enterprise business processes (CBPs) to describe global DeCPD processes. Each CBP defines participating roles, public processes, and a collaboration protocol, carefully matched to the requirements given in a product modeling environment.

To build CBPs that describe DeCPD processes we specified the following generic public processes:

- *GenerateSpecification* is the process to specify a query that describes the searched component characteristics in a *PM Specification*, while *GenerateProposal* is the counterpart of the process mentioned before, which is used by participants to describe a target component in a *PM Proposal*.
- *PublishProposal* makes a *PM Proposal* available to selected collaboration partners. At CIM layer, it is not specified how this is realized from a technical point of view; thus, e.g., on PSM layer the IT experts could provide a distributed database solution for storing *PM Proposal* information.
- By contrast, *PublishSpecification* distributes a *PM Specification* among the developers involved in the collaboration. Suppose that the product model handling was realized using databases; in that case the *PM Specification* would be a SQL-type query that should deliver answers to the collaboration initiator when executing it after several participants have provided valid proposals.
- The process *Search* describes the possibility to search for existing *PM Specifications / Proposals* in a distributed collaboration environment. The search mechanism has no bearing on the execution of a PM Specification query (perform a SQL query e.g.), but it means to find their physical location in the collaboration network. Beyond each search process, there is a complex request-acknowledge and routing method to assure that only trustworthy partners get requested *PM Specifications* and/or -Proposals. These methods are part of our research and will not be discussed in this paper.
- In some environments, it is useful to have a *Notify* process. In technical terms, this corresponds to event-triggered messaging. Independent from any technical realization, it means the participant to register on a particular event, like e.g. the event “newProposal”, whereby he gets informed about every public changes made in this collaboration instance.

- Last but not least a process *Analyze* is needed, that encapsulates the user's decision on how to proceed a collaboration based on the incoming proposals.

Generic DeCPD processes differ in the following three characteristics. The **PM distribution** (specifies how physical *PM Specification* and/or -Proposal data resources are dedicated to collaboration participants), **hierarchies** (describes the maximum levels of sub-collaborations, in the case of a 0-level collaboration there is no subdivision) and **iterations** (allows the compilation of versions and variants).

The DeCPD process sequences in CIM Business Process Diagrams (BPDs) depend on the exact values of these three parameters. To evaluate the necessity of ontologies in the field of DeCPD we reduce complexity by setting up a 0-level collaboration between one initiator und a set of participants. *PM Specifications* are distributed among all participants via broadcast and the *PM Proposals* are interchanged via point-to-point transfer between initiator and corresponding participant. To maintain relevance to real world scenarios at product development we do consider iterations. Fig. 2 shows an example BPD for only one participant.

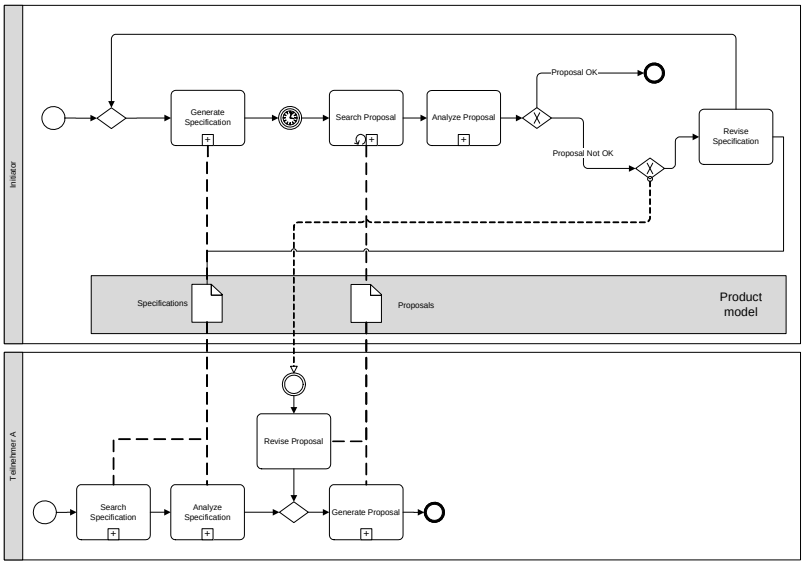

Fig. 2. BPD for o-level collaboration with iterations

4.2 CIM: Data aspect

As described in Section 3 we need a data model that fits on the process described above. We defined the meta-model as shown in Fig. 3.

Fig. 3. DeCPD data meta-model

A concrete instantiation of the meta-model and its resulting complexity depends on the DeCPD characteristics (hierarchies and iterations) of a DeCPD *project* and therefore mainly on the amount of sub-collaborations (*sub-projects, sub*$_P$) and the maximum number of *PM Specification-* and *PM Proposal* variants (*vars*/*var*$_P$), respectively versions (*vers*/*ver*$_P$). As a *variant* we describe all *PM Specifications* / Proposals that belong to one single component and that are published in parallel. Note that each variant has got only one valid *version*.

To conclude, due to the process limitation that no sub-projects are allowed (cp. Section 4.1), we can concentrate on the collaboration process itself and not on the difficulties that affect us when putting together the *sub-results* from *sub-projects*.

4.3 CIM: Service aspect

In describing DeCPD services and their behavior at CIM level, we distinguish between two dimensions: *Service Execution* and *Service Coordination*.

At execution dimensions we distinguish between local and distributed *service execution*. In local *service execution*, each collaboration partner hosts all needed DeCPD core services on its own; thus, the availability of needed services is ensured at any time (= local *service execution*). In the case that DeCPD core services are distributed between participants we have to provide additional service discovery methods at run time. Furthermore we distinguish between *central and decentral service execution*. In the central case, only selected partners provide a choice of core services, whereas in the decentralized case core services are distributed between participants.

Considering the coordination dimension we distinguish between *central and decentral coordination*. *Central coordination* needs a coordinator that controls the cross-enterprise service workflow; *decentral coordinated workflows* do not need a coordinator, but decentral, model-centric mechanisms that make a control flow possible.

To summarize, we decided to use a *decentralized coordinated collaboration* strategy with *local DeCPD core service execution*. Using local *service execution* we do not need to concentrate on strategies for a distributed service search that is as complex as decentral product model data exchange and provides nearly the same challenges. Rather we prefer not using a coordinator and control the workflows by querying the state of the distributed product model.

5 Ontologies for a concrete PSM level data model

The generic data model at CIM level has to be concretized at PSM level by specifying specific data elements needed to fulfill the requirements of a DeCPD. In this section we describe on how to use an ontology approach to specify the PSM data model.

5.1 Ontologies to describe components and connections

One of the core concepts of DeCPD is the decomposition of an overall product development problem (CIM: project) into several sub-problems (CIM: sub-projects). Central element of the CIM data meta-model is the sub-project related component.

The challenge at PSM level is to specify components in an interoperable way, that means to cover aspects of different business domains like design, functional requirements, economic or of course geometric and topological parameters. To generate such a component description, all relevant parameters must be representable in a meaningful and feasible way. This is where we apply ontologies due to their flexibility and power of expression.

As known from many programming languages we provide a common domain specific base for the fundamental expressions. This ontology, that we call *BaseOntology* (*BaseOnt*), depends on the domain of the collaboration and must define appropriate *classes*, *properties* and even *individuals* if needed. For sure the scope of our *BaseOnt* is limited to the field of product development, although it contains essential concepts that could be use in other application domains. The *BaseOnt* must be available for all participants in a DeCPD process.

As we chose the LEGO™ system as domain for our example scenarios (cp. Section 6), our *BaseOnt* provides a special vocabulary for a collaboration in this product development area and we define the following five superclasses:

- **Component**. Holds the subclasses *Project*, *Assembly* and *Part* that are constructional superclasses themselves. Derived from *Part*, we define LEGO™ elements such as *Brick*, *Plate*, *Lego Technic Part*, and others. In the example, these special *Part* classes represent all the different kind of LEGO™ building bricks that can be purchased on the market.
- **Connection**. Within the *Connection* classes it is possible to describe the type and the implementation of connections between *parts*, assemblies and perhaps other important stuff. This is especially needed to ensure, that the different *PM Proposals* connects well during the synthesis phase of the collaboration.
- **Requirement**. This class helps to specify the various kinds of requirements concerning a component. We distinguish between functional (movability or flexibility) and non-functional requirements (price, weight, or material).
- **Resource**. Using this class, ontology elements can be linked to external data storages such as files or database tuples. Generally, an external data source contains more detailed information than provided by the ontology.
- **ValuePartitions**. This is a helper superclass to construct enumeration classes that are needed to restrict the ontology user when defining values.

Using these predefined elements, a participant can describe a component by generating needed *individuals* derived from the given classes. By using properties individuals can be tied together or atomic values can be assigned to them. Sometimes the problem may occur that the *BaseOnt's* classes are not adequate to build up an individual. In this case the missing elements can easily be added to a custom *project ontology* (*PrOnt*), cp. Section 5.3.

Next to the simple description of components, connections between them have to be represented. We provide a solution for this requirement by defining special

connection classes in our *BaseOnt*. So the connection modality (from physical point of view), the involved components and additional semantic information (why the connection is modeled the way it is) can be described in the same way as shown above.

5.3 Project and component queries

For describing the project and its components that should be developed during collaboration we use so called *project queries* (*PrQ*) that work upon a special *project ontology* (*PrOnt*) which extend the *BaseOnt* with project specific stuff.

In a *PrOnt* we define the main component that represents our project subject as a new class by deriving from the *Project* class using the *subClassOf* built-in property of OWL. Additionally components are defined by deriving from the *Assembly* class. In order to represent the topological structure, we create an individual for each new type we defined before and link them together via the *isChildOf* property of the *BaseOnt* like building up a *bill of material* (*BOM*). To declare which components should really be developed in collaboration we mark each individual with a boolean value using the *BaseOnt's isExposed* property. In summary this results in an ontology structure that holds all the information about types and topological structure of the collaboration.

To extract these information from the *PrOnt* we use a project independent *PrQ*, which is of course defined in the W3C standardized SparQL query language. The *PrQ* basically searches for types of those individuals, which are children of another individual that represents a type being a subclass of the *Project* class. The query returns the name of the component, if it should be developed in the collaboration and the associated project type as shown below:

```
PREFIX lego: <http://tuc.de/ontologies/lego_base.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?components ?value ?root_type
WHERE {
    ?root       lego:hasChild   ?x;
                rdf:type        ?root_type.
    ?root_type  rdfs:subClassOf lego:Project.
    ?x          lego:isExposed  ?value;
                rdf:type        ?components.
    ?components rdfs:subClassOf lego:Assembly.
}
```

In either case it might be the initiator that generates the first revision of the *PrQ* and the *PrOnt*. Dependent of the knowledge a collaboration initiator owns when specifying a new *PrQ*/*PrOnt* composition he might not be able to describe the components. This case might be suitable in real word scenarios when developing new products. So we must differentiate between a project partitioning made by the initiator and those made by the participants themselves.

In reality a one-step *PrQ* breakdown is unrealistic. According to the supply pyramid a given *PrQ* will be specified more precisely from tier to tier. So a *PrQ* distributed by a OEM will be republished in a further collaboration by a 1st tier

supplier and so on. Regarding our data meta-model in Section 4.2 it means that projects are recursively subdivided into sub-projects that are connected together by the components interfaces in principal.

The distribution of *PrQ's* is in general only practicable between participants from maximally two supply pyramid layers due to their interest in collaborate. The question is also to allow and/ or how to handle supplier rivals.

After having distributed a project description the initiator hopes to find participants interested in developing selected components. Desired requirements of a *PM specification* are specified in a *component specification query* (*CSQ*). In contrast to the *PrQ*/*PrOnt* the person that describes a *CSQ* (= the initiator in general) is not the one who will also provide the *CompOnt* (= one of the participants). The component is still to be defined (cp. Section 5.1.2).

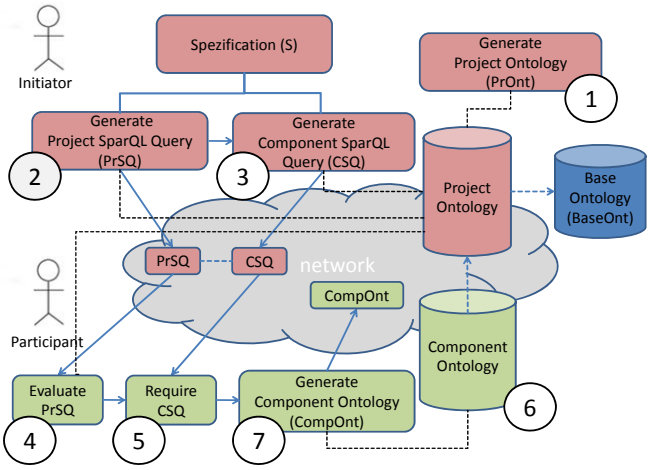

Fig. 4. DeCPD query routing

In the following we will take a look at the query routing that fulfills the requirements of the DeCPD process shown in Fig. 2. We consider a collaboration with only one initiator and an arbitrary amount of participants. Each sub-collaboration (again with one initiator and several participants) would adhere to the following base procedure:

- **Step 1:** The *BaseOnt* is extended to the *PrOnt* by specifying additional components that are not yet contained.
- **Step 2:** The project describing *PrQ* is generated. Executed on the *PrOnt* each *PrQ* returns the list of components to construct during collaboration.
- **Step 3:** The initiator generates a separate *CSQ* (= *PM Specification*) in that the component requirements are specified.
- **Step 4:** A participant receives the *PrOnt* and the *PrQ*. Based on that he can decide whether to provide participate at collaboration or not.

- **Step 5:** Interested participants require a *CSQ* of the component they are interested in from the initiator.
- **Step 6:** The *PrOnt* is extended to the *CompOnt* (= *PM Proposal*) by again specifying additional components.
- **Step 7:** The *CompOnt* is submitted to the initiator and evaluated by executing the *CSQ*.

5.2 Component proposals

A *PM Proposal* at PSM level should describe a concrete implementation of a *PM Specification*. Therefore a *ComponentOntology* (*CompOnt*) is provided. Dependent on the used domain specific CAD models an individual mapping of relevant attributes and features into the *CompOnt* is needed. The mapping is again realized by generating and combining individuals using *PrOnt's* classes and properties. Additionally the *CompOnt* should provide links to the ontology's source CAD files. During evaluation of a *PM Proposal* the original CAD file is still a useful and important representation.

6 Evaluation: Sample use case

As proof of concept we implemented an example use case that combines the PSM service concepts with the ontology based data model approach. In the sample use case, we simulate the collaborative construction of a LEGO™ bulldozer (as illustrated in Fig. 5).

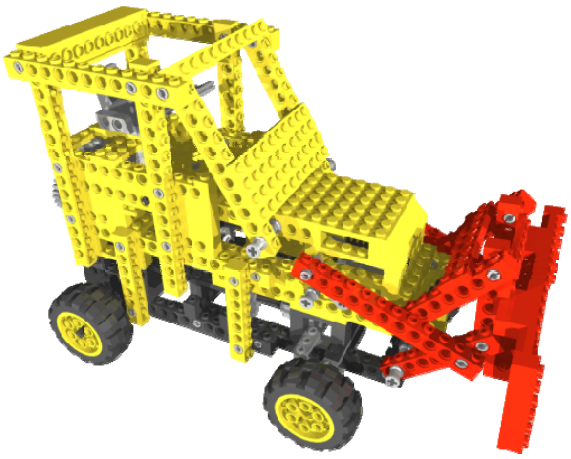

Fig. 5. LEGO™ bulldozer to evaluate ontologies at DeDCP

The bulldozer represents the data element project and consists of three components that are specified by the initiator: A chassis, a body and a shield.

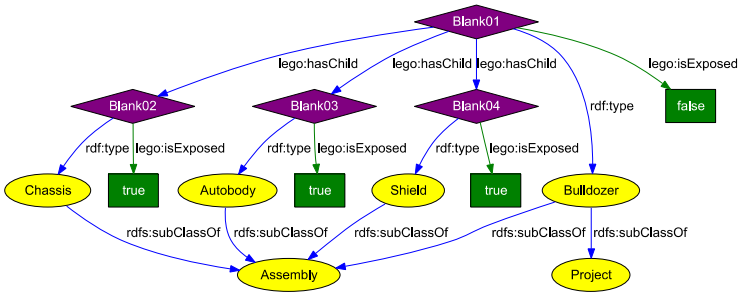

Fig. 6. Project Ontology for the sample use case

As listed in Section 5.3, the **first step** is to use the *BaseOnt* and extend it to the *PrOnt* (cp. Fig. 4, Step 1) by adding project-specific components such as a class for the bulldozer, the chassis, the body and the shield and mark the bulldozer as the project class. For the topolocigal structure we define individuals for each type using the *hasChild* property and mark them via the *isExposed* property to identify which component is a *PM Specification*. Mind that not each component of the underlying bill-of-material has to be developed within the collaboration.

With a special SparQL Query (= *PrQ*) that is developed in **step 2**, the exposed classes can be determined when running the query. Let us assume that in our reference scenario all three components should be developed by collaboration participants.

For each exposed component the initiator has to implement a specification as SparQL query in the **third step** (cp. Fig. 4, Step 3). In summary there are three requirements for this assembly, one restricting the dimension, a second demanding the movability, and the third limiting the costs. The schema of this query type is quite generic and can be used for arbitrary projects and not only for describing LEGO™ toys.

```
PREFIX spec: <http://tuc.de/ontologies/spec_bulldozer.owl#>
PREFIX lego: <http://tuc.de/ontologies/lego_base.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?item
WHERE {
    ?item           rdf:type                spec:Chassis;
                    lego:hasRequirement     ?req_dim, ?req_mov,
                                            ?req_costs.
    ?req_dim        rdf:type        lego:Dimension;
                    lego:hasDepth   8;
                    lego:hasWidth   ?width
                    FILTER(?width >= 20 && ?width <= 30).
    ?req_mov        rdf:type                lego:Moveability;
                    lego:hasAxisRelation    lego:XValue,
                                            lego:ZValue;
                    lego:hasMoveability     lego:LinearValue.
    ?req_costs      rdf:type        lego:Costs;
                    lego:hasCosts   ?price FILTER(?price <= 45.99).
}
```

Based upon these requirements provided in a *CSQ*, a developer can now start to implement a *PM Proposal*. This is done by generating a *CompOnt* that follows the structure of the *CSQ*. Therefore a collaboration participant's work is to map his final parameters from his proposed product model into the *CompOnt*.

To check if the *CompOnt* is a valid proposal, both participants and the initiator need to execute the *CSQ* upon this *CompOnt*. If the result of a *CSQ* is an individual of a requested component (cp. individual *Tuc01* in Fig. 8 below) the proposal is valid, if not it is invalid and there may remain some unfulfilled requirements.

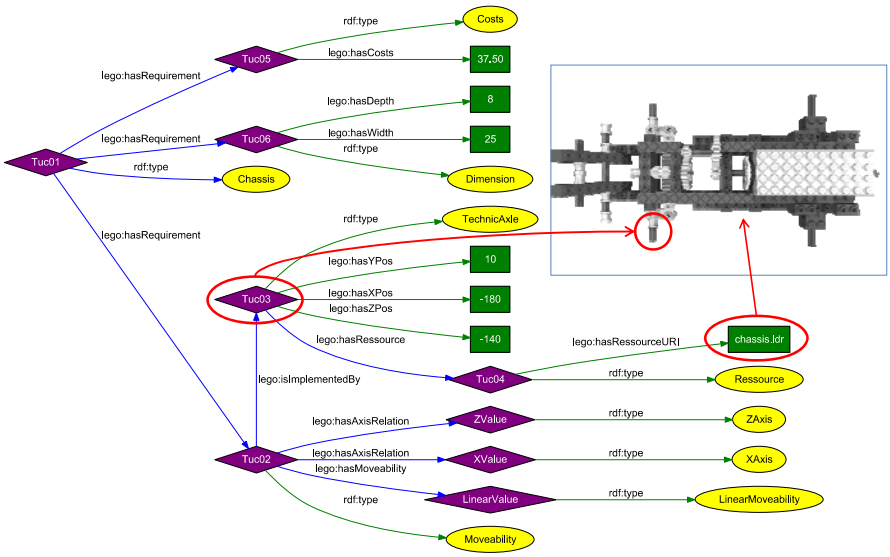

Fig. 8. Extract from component ontology for a CAD representation of a LEGO™ bulldozer chassis (LDraw).

When the initiator has received enough valid *PM Proposals* (*CompOnts*) for each component, he can assemble them into a *result ontology* instance (*ResOnt*), which represents the package solution. In other words, it means that assembling all corresponding CAD model files will result in a final CAD model like the one shown in Fig. 5.

7 Conclusion and outlook

As mentioned in Section 3, there is a lack of using ontologies to support distributed collaboration processes like those addressed in this paper. The development and application of an ontology especially designed for decentral organized product development enables the full realization of DeCPD platforms. We are now able to complete our existing experimental PCP implementation, which was presented at the CeBIT trade fair in Hannover in 2009, by integrating the missing semantic data

description module. With it we have the possibility to interpret relevant information from the state of the distributed product model that allows us to follow a decentrally developed process model.

Beside the integration of the ontology concept into our PCP, we have to evaluate the applicability in real world engineering scenarios that is on CIM layer, i.e., independent from any IT realization. At the same time existing methods and protocols for query distribution and routing have to be evaluated and expanded to hierarchically designed DeCPD processes containing sub-collaborations. In that case we get confronted with the problem of describing interfaces between product models that could perhaps be realized in a similar way. This problem is yet unsolved. Finally, a more practical requirement which we shall address in future work is to provide automatic mappings from CAD files into the ontology approach.

References

1. Li, W.D. and Qiu, Z.M. (2006): State-of-the-Art Technologies and Methodologies for Collaborative Product Development Systems. In: International Journal of Production Research, 44(13), pp. 2525-2559.
2. Patil, L., Dutta,D. and Sriram, R. (2005): Ontology-Based Exchange of Product Data Semantics. In: IEEE Transactions on automation science and engineering,Vol.2, No.3, pp. 213-225.
3. Li W.D., Ong S.K. and Nee A.Y.C. (2006): Integrated and Collaborative Product Development Environment. Technologies and Implementations. World Scientific Publishing Co. Pte. Ltd. Singapore.
4. Stiefel, P. D.; Müller, J. P. (2007): ICT interoperability challenges in decentral, cross-enterprise product engineering. In: Gonçalves; Ricardo J., pp. 171–182.
5. Stiefel, P. D.; Müller J.P. (2008): Realizing dynamic product collaboration processes in a model-driven framework: Case study and lessons learnt. In: K.-D. Thoben, K. S. Pawar, & R. Gonçalves, eds., 14th International Conference on Concurrent Enterprising, 23-25 June 2008, Lisbon, Portugal.
6. Stiefel, P.D and Müller, J.P. (2009): A model-based software architecture to support decentral product development processes. In: Proceedings of the Eighth Workshop on eBusiness, Web2009, Arizona; to be published.
7. Müller, J. P. (1996): The Design of Intelligent Agents – a Layered Approach. Lecture Notes in Artificial Intelligence, Volume 1177, Springer-Verlag.
8. Smith, Reid G. (1981): Frameworks for Cooperation in Distributed Problem Solving. In: IEEE Transactions on systems, man, and cybernetic, Volume 11, Seiten 61-70.
9. Kim, K.-Y. et al. (2006): Ontology-based assembly design and information sharing for collaborative product development, In: Computer-Aided Design, Vol. 38, pp. 1233-1250.
10. Liang, Vei-Chung and Paredis, C.J.J. (2003): A port ontology for automated model composition. In: Proceedings of the 2003 Winter Simulation Conference.
11. Mostefai, S. et al. (2004): Effective Collaboration in Product Development via a Common Sharable Ontology. In: Journal of Computational Intelligence, Vol.2, No. 4, pp. 206-212.