Evaluation of Configuration Knowledge in Industrial Manufacturing

Joachim Baumeister[1,2]

[1] denkbares GmbH, Friedrich-Bergius-Ring 15, D-97076 Würzburg
[2] University of Würzburg, Am Hubland, D-97074 Würzburg
joachim.baumeister@denkbares.com

Abstract. Current industrial information systems maintain and process immense amounts of knowledge to be used for the development, configuration, and production of artifacts. From a logical point of view these elements process knowledge abozt the same concepts albeit existing in different information systems. From a practical point of view, however, the systems are only loosely coupled and only have little linkage. These semantic gaps prevent a seamless analysis and evaluation of the entire knowledge, which in practice, is of significant importance for productional success. In this paper, we propose a general high-level ontology that integrates with the knowledge of the information systems and enables the analysis and evaluation tasks. We describe the basic concepts and properties of industrial knowledge and motivate a number of general evaluation tasks.

Keywords: semantic technologies, ontologies, plant engineering, quality assessment

1 Introduction

The manufacturing industry is currently moving fast from traditional manufacturing processes to digitalized processes. This transition also includes the full representation of so-called *digital twins* of the manufactured artifacts. Digital twins are the enablers for mass-customized products that deliver premium quality of single-slot products at the range and costs of mass production.

The representation and handling of these twins, however, typically ranges over a wide range of industrial process systems within the IT landscape. The different interfaces of these systems act as a barrier for the data sharing and linkage and thus prevent a full digital handling of the process. These systems also hold knowledge about configuration and manufacturing processes that need to be shared between the systems. Today, often a manual transition between the system borders is developed which is complex, error-prone, and time-consuming. Also a full knowledge process is not possible due to the system borders. This prevents...

- a consistent development semantics of the manufacturing knowledge; no semantic gaps between system transitions,

– a simple and lossless connection to existing information infrastructures within or outside the company, and
– a comprehensive and complete quality assessment of the knowledge (validation and verification).

Semantic technologies are a key enabler for archiving these goals [4]. They already proved to be capable to interconnect broad landscapes of IT systems by preserving the intended semantics of the delivered data. Semantic technologies, especially the use of standardized ontology languages, provide a declarative semantics of the manufactured products and their features. Also the reasoning about these products can be defined in a declarative and clear manner.

This paper will focus on the last issue described above, i.e., the quality assessment of distributed knowledge. We present an ontology for describing the typical artifacts of a producing company:

– products and their structure
– features and feature items of products
– relational knowledge for customization and configuration

Despite general approaches in knowledge-based configuration [5], the proposed ontology introduces concepts and constraint types that are tailored to the use in the manufacturing industry. That way, it is easy to understand and share between manufacturing companies. We also contribute a shallow integration approach that is able to process quality questions on the integrated and linked knowledge base.

The remainder of the paper is organized as follows: We first introduce a general ontology for configuration items, i.e., artifacts that are produced in industrial manufacturing. In Section 3 we briefly sketch, how different knowledge bases from distributed knowledge bases can be integrated into the presented ontology. We show uses cases of analysis and evaluation in Section 4. The paper concludes with a discussion of related and future work in Section 5.

2 Ontology for Configuration Items

A reusable ontology for industrial manufacturing needs to represent different products, that are defined by feature values. Different types of knowledge constrain the combinatorics of features but also of products, see Figure 1. In more detailed, the following key concepts need to be explicitly defined:

Product portfolio: A *product portfolio* includes all artifacts produced by a manufacturer. These artifacts are named *products* and are typically organized in a *hierarchy*. Products itself run through a life cycle yielding different *product versions*.

Features: A single product is characterized by distinct *features*. The collection of features for a produced product is often called a *configuration*. These features are sub-classified into different types, i.e., features with a numerical

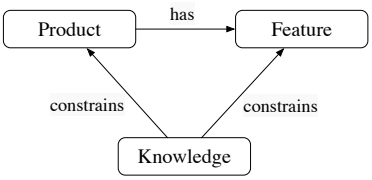

Fig. 1. Products have features; both are constrained by different types of knowledge.

feature value (e.g. `length`) versus features with discrete values (`color`). A larger number of features is organized in a hierarchy, for instance, in order to mark the origin or use of the feature. Typical origins of features within a company are the sales department, engineering group, and the production department.

Relational knowledge: The values of some features are dependent on the assignment of other feature values and/or product types. One example is the calculation of a total product weight that is dependent on the single weights of the included features. Another example is the prohibition of a distinct feature value combination, e.g., a roof antenna for convertible cars.

In the following, we will describe these elements in a more formal manner. For the definition of the manufacturing product ontology we used a derivation of the SKOS ontology [10], which already defines a number of basic concepts for general knowledge organization systems. We derive the class `LinkedConcept` from `skos:Concept` to describe the linked characteristics of concepts between a number of information systems. Also, linked concepts have a property `linkedSystem` pointing to the original information system and `linkedId` storing the original identifier of the concepts.

2.1 Product Portfolio

The product portfolio organizes all produced artifacts of a manufacturer in a loose hierarchical structure, see Figure 2. The class `ProductPortfolio` wraps all products and is derived from `skos:ConceptScheme`. A distinct product acts as a root of the portfolio using the property `topConceptOf`. The hierarchy of the portfolio follows the composite pattern. It is created by instances of `Product`, that itself can have successors of `Product` by using the property `superProduct` (a derivation of `skos:broader`). There exists special types of Products to represent the group of concrete products in model families and model lines (also known as series).

The life cycle of products is modeled by different product versions. That way, a `ProductVersion` instance is assigned to a concrete product to represent the life cycle version of this product. Each product version has a start date and an end date, that defines the validity of this version. Often, different model years

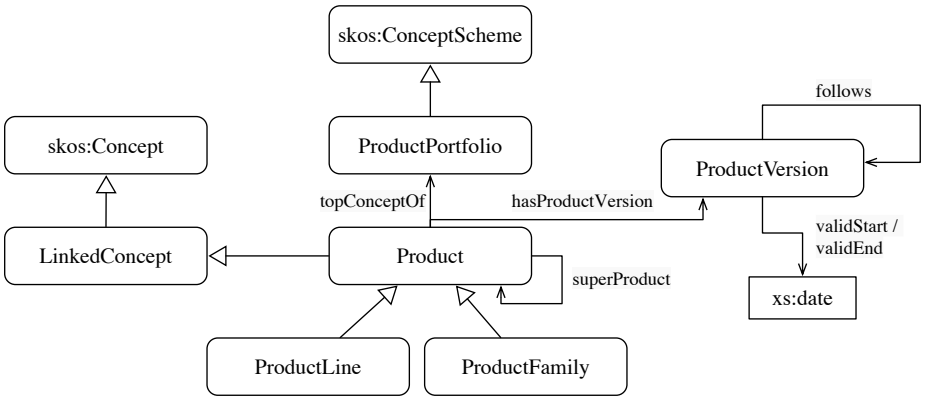

Fig. 2. Classes and properties of the generic product portfolio.

are represented by different product versions. Here, the particular calendar years are the valid dates for the product versions.

2.2 Features, Assignments and Configurations

A product is characterized by its possible features. More precisely, a concrete product version defines a collection of features, that are available in this version. It is worth noticing, that all available features are described for a product and that the combination of some features may be not possible in a concrete instance (e.g., feature electric engine vs. gasoline engine in a automotive setting). Figure 3 shows that `Feature` instances are organized hierarchically by the `superFeature` property, a derivation of `skos:broader`. Different types of features are characterized by the value type it can be assigned to. We define `NumericalFeature` storing numerical float values (e.g., length of a car)and `ChoiceFeature` having a predefined list of possible choice values (e.g., color of a car). Consequently, we introduce properties for assigned choice values to Features (`possibleValue`) and initial values (`initValue`, sometimes used as defaults). For choice values it is possible to state an order between the different values by the property `valueOrder`, e.g., for a feature *doors* the ordered choices *two*, *four*, and *five*.

An assignment between a value and a feature is captured by the corresponding class `assignment`, that provides the property `hf` (has feature) to reference the feature instance and the property `hfv` (has feature value) to reference the value. Please note, that the value class need to correspond to the assigned feature class.

A concrete instantiation of a product is manifested by an instance of `Configuration`. Basically, a configuration stores a collection of features with concrete values, i.e., `Assignment` instances. Please note, that the assignments need to reference to features of the same product version. Also, a `Configuration` instance

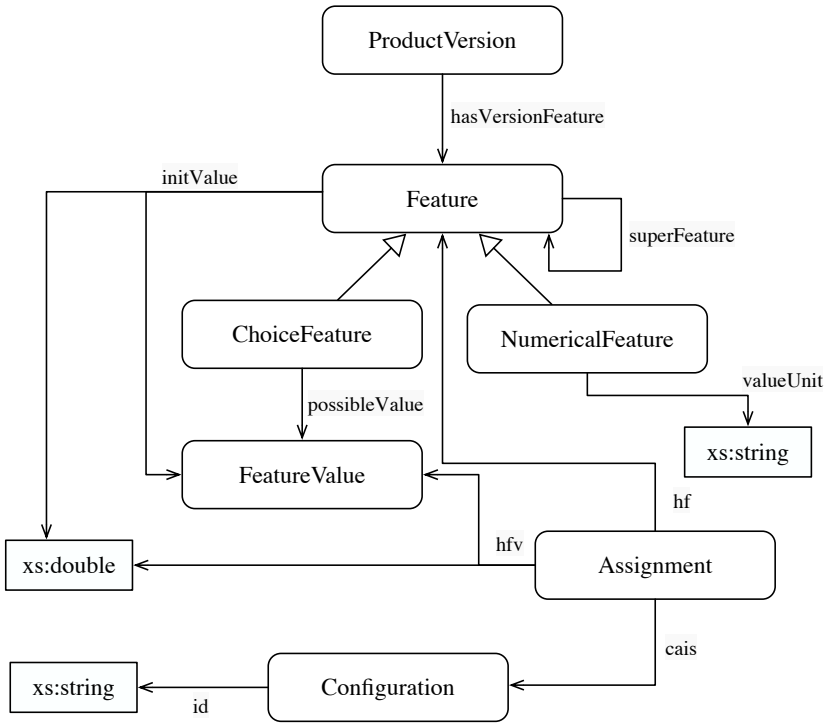

Fig. 3. Classes and properties defining the features, an assignment to feature values, and a configuration.

usually refers to an ID that distinguishes it from other configurations. Often, this identifier is called serial number or machine number.

2.3 Relational Knowledge

The features of products are constrained by relational knowledge. As shown in Figure 4, relational knowledge refers to a `Condition` instance, that defines the requirements for the execution of this particular knowledge element. These requirements vary between the specializations of `RelationalKnowledge`: For instance, the condition can state invalid combinations between features and their values (`InvalidValues`). Furthermore, relational knowledge is used to derive specific values of features based on the given condition; the derivation is possible for choice features (`SymbolicDeduction`) and for numerical features (`Calculation`). A `SATConstraint` defines a (often complex) condition, that need to be either positively or negatively satisfied. For a simplified analysis and exchange of relational knowledge, all involved features are linked explicitly by the relation `involved`. We also define sub-properties `conditionedFeature` and `derivedFeature` for,

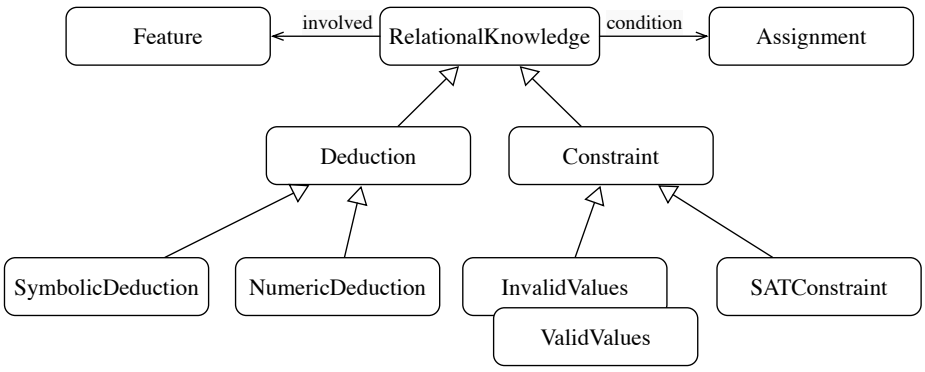

Fig. 4. Classes and properties defining the relational knowledge, mostly between feature values.

depending of the specific type of the relational knowledge, the features used in the condition but also in the conclusion are linked by these properties.

For the ontological representation of conditions of relational knowledge and the representation of deductive knowledge in particular, we refer to the rule language SWRL [8]. Albeit the syntax of SWRL is not easy to comprehend for untrained users, it provides a common standard for representing equations and conditions. In the context of our work, however, we focus on the shallow representation of conditioned features and derived features, since this knowledge is easy to interchange between systems and is sufficient to answer a number of analysis and evaluation questions.

3 Linking Distributed Knowledge

Based on the general ontology defined in the previous section, we give an example of a meaningful syncing strategy for an industrial use case. As we motivated in the introduction, the knowledge of a producing company is typically dispersed over a number of information systems. For the analysis of the summed knowledge base, the knowledge elements of the particular systems need to be synced and linked.

In an exemplary situation, a company runs a product data management system (PDM) in the engineering department, a customer relations and pricing system in the sales department (CRM/CPQ), and a service information system (SIS) in the after sales department. Also an enterprise resource planing system (ERP) is connected for managing the master data of the company. The knowledge included in these systems considers the same products, features, and interrelations.

Figure 5 shows the proposed approach, where the local knowledge bases are linked with an integration ontology to be used for all analysis and evaluation

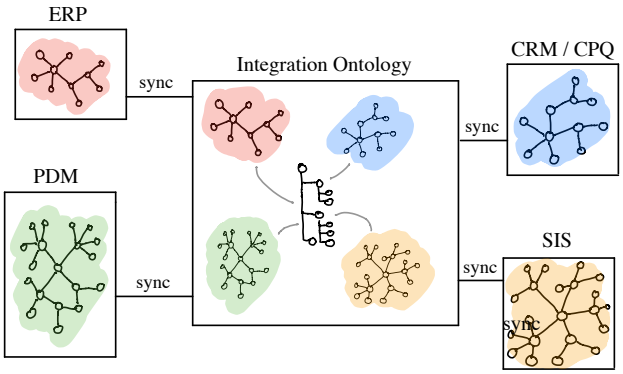

Fig. 5. Linking knowledge of existing information systems with an integration knowledge base.

tasks. In a canonical mapping scheme we represent all elements in the integration ontology, but introduce distinct instances for the elements with local namespaces in each system. For example, a specific product `p` referenced in all information systems, we also introduce three corresponding instances `erp:p`, `pdm:p`, `cpq:p`, and `sis:p` of `Product`. The identity semantics between all p instances is represented by the sub-properties of `skos:mappingRelation`, such as `skos:exactMatch` [10]. In consequence, we arrive at one integration ontology and one ontology for each connected information system.

This procedure may produce redundant instances at first sight, but allows for a independent development and use of the different information systems and the simple combination of different knowledge versions of each information system.

It is worth noticing, that the integration ontology does not include and map the entire depth of the knowledge bases it is linked to, but integrates only that levels of knowledge that are used in the evaluation task. In consequence, knowledge included in the integration ontology can be used for analysis but may be not executable as in the original system.

4 Use Case: Analysis and Automated Verification

In most cases, the installed information systems provide mechanisms to maintain, analyze, and evaluate the knowledge included in their systems (*local evaluation*). In a distributed scenario, however, the analysis and verification of the combined knowledge represented by all systems is not possible (*global evaluation*). This combined consideration of knowledge is made difficult because of the different semantics and missing general interfaces of the particular information systems. With the proposed ontology, a shallow analysis and verification of the knowledge becomes feasible for a wide range of participating information systems. We show

some examples that motivate that even the shallow investigation of the combined knowledge is valuable for maintaining a steady health state of the knowledge.

In the context of analysis and evaluation of knowledge cases we distinguish *verification* and *validation* tasks [6]. The *validation* of knowledge basically considers the correct output of the system for reasonable inputs and—in our scenario—requires the deep semantics of all participating knowledge systems. The *verification* of knowledge investigates the correct construction of knowledge with respect to possible anomalies. Here, a shallow analysis of the linked knowledge bases is possible and valuable for the development and maintenance of the entire system. Corresponding to the anomalies star classification, e.g., described in [2], we distinguish *circularity*, *redundancy*, *inconsistency*, and *deficiency* of knowledge. In the following, we discuss a selection of verification methods that are simple to implement, but valuable for supporting the distributed development. We focus on the introduction of a number of deficiencies, i.e., bad smells [3] in the design of the ontology.

4.1 Bad Smell: Uneven Twins

We observe two matching features f_1 and f_2 in different models. Both features have defined values

$$f_1 \in M_1 \ \wedge \ dom(f_1) = \{v_{1,1}, v_{1,2}, \ldots, v_{1,n}\} \ and$$

$$f_2 \in M_2 \ \wedge \ dom(f_2) = \{v_{2,1}, v_{2,2}, \ldots, v_{2,m}\} \, .$$

The features are defined to be exact matches of each other, i.e., $exactMatch(f_1, f_2)$.

We report an design anomaly (bad smell), when there exists a feature value $v \in dom(f_1)$ that has no value $v' \in dom(f_2)$ with an exact match. That way, both features are to be defined to be exact matches (twins) but define values that are not known in the counter feature. A common reason for such a anomaly is a missing match relation, that was not defined by the knowledge engineers. Sometimes, however, the change of the semantics of values without the broadcasting to the other information systems are a further reason.

The following SPARQL statement shows a possible query for identifying uneven twins `f1` and `f2`.

```
SELECT  ?f1 ?f1_value ?model
WHERE { ?f1 a co:Feature ;
            co:possibleValue ?f1_value ;
            co:inModel ?model .
            MINUS { ?f2 a co:Feature ;
                        co:possibleValue ?f2_value .
                    ?f1 skos:exactMatch ?f2 .
                    ?f1_value skos:exactMatch ?f2_value .
                    FILTER (?f1 != ?f2) }
}
```

4.2 Bad Smell: Knowledge Twins

We define relational knowledge in different models that derive feature values for the same features with same or intersecting feature sets in the condition, i.e.,

$$k_1 : \{a_1, \ldots, a_n\} \rightarrow \{b_1, \ldots, b_m\} \ \wedge \ k_2 : \{c_1, \ldots, c_p\} \rightarrow \{d_1, \ldots, d_q\},$$

where a_i, b_i, c_i, d_i are assignments $f = v$ of values v to features f. Two sets of assignments A_1, A_2 are *intersecting*, when there exists at least one assignment in each set, that have matching features and feature values, i.e.,

$$for\,(f_1 = v_1) \in A_1, (f_2 = v_2) \in A_2 : exactMatch(f_1, f_2) \wedge exactMatch(v_1, v_2)$$

Please notice, that the property `exactMatch` is reflexive and thus a feature also has an exact match to itself.

We observe a knowledge twin, when there exists an intersection between the sets of assignments in the condition and an intersection between the sets of the knowledge action, i.e.,

$$\{a_1, \ldots, a_n\} \cap \{c_1, \ldots, c_p\} \neq \{\} \ \wedge \ \{b_1, \ldots, b_m\} \cap \{d_1, \ldots, d_q\} \neq \{\}$$

The following SPARQL statement shows a simplified query for detecting knowledge twins `k1` and `k2` in different knowledge models, here the sales model and the engineering model.

```
SELECT  ?k1 ?k2
WHERE { ?k1 a co:RelationalKnowledge ;
            co:conditionedFeature ?f1con ;
            co:derivingFeature ?f1der ;
            co:inModel sam:salesModel .
        ?k2 a co:RelationalKnowledge ;
            co:conditionedFeature ?f2con ;
            co:derivingFeature ?f2der ;
            co:inModel enm:engineeringModel .
        FILTER (?k1 != ?k2)
        FILTER EXISTS {
           ?f1con skos:exactMatch ?f2con .
           ?f1der skos:exactMatch ?f2der . }
}
```

4.3 Incompatible Concept Matching

A (semantically) identical feature exists in two different information systems and is matched, i.e., $f_1 \in M_1 \ \wedge \ f_2 \in M_2 : exactMatch(f_1, f_2)$. Due to a design decision the type of the feature in M_1 is modified but this is not broadcasted to the other information system.

For example, the feature *fuel tank* is a choice feature in the sales system with only two possible values *40 liters* and *80 liters*. Since the engineering department needs to compute with the actual capacity of the tank, they change the

feature class from symbolic to numerical. The exact match relation still exists. In consequence, we arrive at an incompatible concept matching.

The following SPARQL queries for a feature `f1` that has an exact match to a feature `f2` with class `f2Type` that is different from all classes of `f1Type`.

```
SELECT  ?f1
WHERE { ?f1 a co:Feature ;
            skos:exactMatch/rdf:type ?f2Type ;
        MINUS { ?f1 a ?f2Type . }
}
```

4.4 Bad Smell: Same Surface

There exist two features f_1, f_2 that have similar names, but there exists no matching relation between them. This simply smell is a indicator for incomplete knowledge, typically occurring for new elements in the (distributed) knowledge base.

The following SPARQL statement implements a simplified version of this smell, where we query for features `f1` and `f2` that have an identical label. A more refined query would look for similar strings.

```
SELECT ?f1 ?f2
WHERE {  ?f1 a co:Feature ;
             rdfs:label ?f1Label .
         ?f2 a co:Feature ;
             rdfs:label ?f1Label .
         FILTER (?f1 != ?f2)
         FILTER NOT EXISTS { ?f1 skos:exactMatch ?f2 }
}
```

5 Conclusions

For the production of industrial artifacts typically a number of interacting information systems are used. These information systems show a large overlap of concepts they are storing and working with. Nevertheless, these concepts are in principle not linked or matched to each other. This especially makes is difficult when evaluation of the distributed knowledge comes into place.

We introduced a simple ontology for the description of industrial production artifacts. The ontology covers the product world, features, and versions of artifacts as well as a shallow interpretation of configuration knowledge. In consequence, we introduced some evaluation methods to motivate the general use of the ontology.

In the literature, we see a couple of approaches for describing industrial production knowledge in a semantic manner. For example, Badra et al. [1] introduce an ontology for the representation of configuration knowledge and show a use case for Renault automotive. In comparison, they did not cover the issue of

knowledge distributed over a couple of information systems. In Soinien et al. [9] a configuration ontology is described. An elaborate model for the representation of industrial production is introduced, as above, the distribution of knowledge is not covered. The Volkswagen Vehicle Ontology [7] gives a description of features and components of the Volkswagen group. From the perspective of this paper, it focuses on the representation of the sales model, but does not explicitly define knowledge distributed over different information systems.

In the future, we are planing to extend the ontology description with respect to possible relational knowledge. Most importantly, the ontology needs to be mapped to existing information systems already in use in a typical industrial setting, i.e., ERP and CRM systems such as SAP and Salesforce. Also, we plan to define and implement an extended catalog of evaluation methods tailored to investigation of distributed knowledge bases.

References

1. Badra, F., Servant, F.P., Passant, A.: A semantic web representation of a product range specification based on constraint satisfaction problem in the automotive industry. In: Proceedings of the 1st International Workshop on Ontology and Semantic Web for Manufacturing. pp. 37–50. Heraklion, Crete, Greece (2011)
2. Baumeister, J., Seipel, D.: Verification and refactoring of ontologies with rules. In: EKAW'06: Proceedings of the 15th International Conference on Knowledge Engineering and Knowledge Management, LNAI 4248. pp. 82–95. Springer, Berlin (2006), http://ki.informatik.uni-wuerzburg.de/papers/baumeister/2006/EKAW06_baumeisterSWRL.pdf
3. Baumeister, J., Seipel, D.: Anomalies in ontologies with rules. Web Semantics: Science, Services and Agents on the World Wide Web 8(1), 55–68 (2010), http://dx.doi.org/10.1016/j.websem.2009.12.003
4. Coskun, G., Streibel, O., Paschke, A., Schäfermeier, R., Heese, R., Luczak-Rösch, M., Oldakowski, R.: Towards a corporate semantic web. In: International Conference on Semantic Systems (I-SEMANTICS '09). pp. 602–610. Graz, Austria (2009)
5. Felfernig, A., Hotz, L., Bagley, C., Tiihonen, J.: Knowledge-based Configuration: From Research to Business Cases. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edn. (2014)
6. Gómez-Pérez, A.: Evaluation of ontologies. International Journal of Intelligent Systems 16(3), 391–409 (2001)
7. Hepp, M.: http://www.volkswagen.co.uk/vocabularies/vvo/ns (volkswagen vehicles ontology)
8. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language - Combining OWL and RuleML, W3C Member Submission . http://www.w3.org/Submission/SWRL/ (May 2004)
9. Soininen, T., Juha Tiihonen, T.M., Sulonen, R.: Towards a general ontology of configuration. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 12, 357–372 (1998)
10. W3C: SKOS Simple Knowledge Organization System reference: http://www.w3.org/TR/skos-reference (August 2009)