The Knowledge Engineering Review, Vol. 24:3, 309–326. © 2009, Cambridge University Press
doi:10.1017/S0269888909990087 Printed in the United Kingdom

Ontologies of engineering knowledge: general structure and the case of Software Engineering

MIGUEL-ANGEL SICILIA, ELENA GARCÍA-BARRIOCANAL, SALVADOR SÁNCHEZ-ALONSO and DANIEL RODRÍGUEZ-GARCÍA

Computer Science Department, University of Alcalá, Carretera Madrid-Barcelona, km. 33.6, 28871 Alcalá de Henares, Madrid, Spain; e-mails: msicilia@uah.es, elena.garciab@uah.es, salvador.sanchez@uah.es, daniel.rodriguezg@uah.es

Abstract

Engineering knowledge is a specific kind of knowledge that is oriented to the production of particular classes of artifacts, is typically related to disciplined design methods, and takes place in tool-intensive contexts. As a consequence, representing engineering knowledge requires the elaboration of complex models that combine functional and structural representations of the resulting artifacts with process and methodological knowledge. The different categories used in the engineering domain vary in their status and in the way they should be manipulated when building applications that support engineering processes. These categories include artifacts, activities, methods and models. This paper surveys existing models of engineering knowledge and discusses an upper ontology that abstracts the categories that crosscut different engineering domains. Such an upper model can be reused for particular engineering disciplines. The process of creating such elaborations is reported on the particular case study of Software Engineering as a concrete application example.

1 Introduction

Engineering has been defined as the *science of production* (Auyang, 2004). Other accounts of the term define it as the discipline(s) of acquiring and applying scientific and technical knowledge to the design, analysis, and/or construction of works for practical purposes. Vincenti (1993) reported on case studies regarding how design knowledge is developed from functional collections of information, concluding that engineering knowledge is different from scientific knowledge since it is characterized by 'knowing how to do or make things'. This and other accounts of engineering knowledge differentiate it from other kinds of knowledge essentially by an emphasis on *practice*. As such, the categories used in every engineering endeavor are mainly related to final artifacts and disciplined procedures for producing them or operating those artifacts.

Engineering requires intensive practice-oriented knowledge, thus it is practical to categorize it as vertical knowledge versus horizontal knowledge (Dias, 2007). Discipline related information, comprising engineering science theories and codes of practice, can be referred to as *vertical knowledge*. In addition, during a given design project, there will be *horizontal knowledge* that is generated by the design team. Also, the main concepts and ways of doing that crosscut different disciplines can be considered a third subset of engineering knowledge that we might call *upper (or common) vertical knowledge*. Common vertical knowledge excludes, by definition, scientific knowledge that is specific to the design of particular classes of engineering products, for example, physical laws models reported by Yoshioka *et al.* (2004) or common mathematical knowledge (Gruber & Olsen, 1994). As a consequence, such common knowledge retains only those essential elements that can be considered as characteristic of the way engineers work. This latter category is the main focus of the model described in the rest of this paper. The scattered literature on engineering knowledge models indeed provides concepts and relation definitions for many aspects that pertain to such common vertical knowledge and specific purpose. However, there is no reuse of foundational engineering elements between models. Here, we try to cover the gap for reference models in engineering.

There are many different branches of human activity that are referred to as ‘engineering’, some of them significantly disparate. However, all of them have a common substrate that differentiates them from other activities. The roots of such common grounds can be found in the history of the concept. Derived from the Latin word *ingenium* (meaning ‘natural capacity or invention’), the word *engineer* was probably born from another related word ‘engine’. The early engineer was largely concerned with making and operating engines of war. Engineering was thus originally divided into military engineering, which included the construction of fortifications as well as military engines, and civil engineering, related to non-military projects such as bridge construction. However, that distinction is no longer useful to discern the ontological differences and commonalities in the diverse engineering disciplines, since it rests on the uses of the artifacts produced (which are accidental rather than essential) and not on the methods, techniques or organizations employed to use them.

Regardless of the history of the term, engineers focus on creating ‘useful’ artifacts (as opposed to the ‘beautiful’ artifacts created by *artists*), and they do it by applying knowledge and procedures considered scientific (as opposed to ‘traditional techniques’ as used in *craftsmanship*). They do not, however, aim at producing knowledge in itself (as opposed to the principal aims of *scientists*). This leads to an importance on representing requirements (or intended uses) inside engineering models, and the importance to focus on how they evolve into products by using particular methods and techniques. From the rough sketch of characteristics we have just discussed, the engineering domain can be characterized as that of purposeful and planned actions to devise artifacts for specific practical objectives by using scientific and technical knowledge. Such definition captures the key characteristics of the engineering activity while retaining a high degree of generality. This will be the starting point for the rest of the discussion.

Several reports on models for engineering activities or elements can be found in the literature. However, they are fragmentary in their coverage of categories that could be applicable to different engineering disciplines, and their scope is tied to particular domains. This paper reports on an ‘upper level’ ontology for engineering and presents Software Engineering (SE) as a case study of its applicability. Upper ontologies provide definitions of high-level concepts such as time, quantities, topology and the like. Examples of upper ontologies are Sowa upper ontology (Sowa, 2000), *Dolce* (Gangemi *et al.*, 2002) or Cyc (Lenat, 1995). In this work, we use OpenCyc[1], the open source version of Cyc, as a continuation of our previous work (Abran *et al.*, 2006). Although other upper ontologies could be used, OpenCyc provides a set of open tools for developing applications and inferencing. However, it is possible to map different ontologies for interoperability purposes, eventually leading to some degree of standardization (Niles & Pease, 2001). An ‘upper ontology for engineering’ is not a replacement but an extension of upper ontologies situated between them, and the more specific engineering models for concrete disciplines.

The ontology is expressed in the OWL[2] (Ontology Web Language) language combined with SWRL[3] (Semantic Web Rule Language). These languages were selected because of (i) their open nature and (ii) the growing number of tools and technologies that use them. To the best of our knowledge, the ontology presented is the effort that aims at becoming an upper model for different engineering domains, even though parts of existing ontologies might be abstracted as generic models (e.g. Katranuschkov *et al.*, 2002; Darlington & Culley, 2008).

[1] http://www.opencyc.org/

[2] http://www.w3.org/TR/owl-features/

[3] http://www.w3.org/Submission/SWRL/

The intention of an upper ontology for engineering is to abstract out the details of concrete domains. Nevertheless, its utility will be fully appreciated only when extended and applied for particular purposes. SE is a good candidate as a case study for the applicability of such a general model to specific situations, as this discipline is recent in comparison to more traditional engineering disciplines. However, SE was, from the beginning, conceived as the translation of established engineering ways of practicing. For example, Parnas (1998) stated that '[…] the introduction of accredited professional programmes in software engineering, *programmes* that are modeled on programmes in traditional engineering disciplines will help to increase both the quality and quantity of graduates who are well prepared, by their education, to develop trustworthy software products'. There has been considerable debate on the 'engineering' character of SE, and its connection to an abstract engineering model helps in manifesting the differences and commonalities, complementing specific work in the area (Abran *et al*., 2006).

The rest of this paper is structured as follows. Section 2 provides a brief state of the art on existing knowledge representation models and knowledge-based systems in the different engineering domains, focusing on which of their elements can be considered as generic. Then, Section 3 provides an analysis of the domain of engineering and the main categories of knowledge that arise in engineering domains. Section 4 reports on the concrete case study that extends and refines the upper ontology model described for practical purposes in the field of SE. Finally, conclusions and potential future extensions and applications are reported in Section 5.

2 Survey of knowledge representation for engineering domains

This section briefly references existing work on ontologies covering engineering concepts. Only those works that report on ontologies, of which a part could be considered generic, are included. One initial question regarding this survey is what is considered engineering knowledge. For example, Dias (2007) explored the philosophical dimension of the choice of 'connexionist' and 'cognitivist' approaches to knowledge representation, emphasizing the role of the former as an account to represent Polanyi's (1966) notion of tacit knowledge. However, we are concerned here with ontological representations, which, by definition, model conceptual domains. Therefore, we will not discuss the application of connexionist or pattern-finding models here. Other classifications of engineering knowledge use the domain as a criterion, even though such distinction is not useful for our purposes. On the contrary, the facets described by Colombo *et al.* (2007) are indeed useful in discerning categories of engineering knowledge and will be mentioned in the rest of the paper. Concretely, the types considered are:

- *Structural* knowledge: about the components which comprise the design object and their relations.
- *Behavioral* knowledge: about the behavior of the design object, that is, about the ways the device responds to changes in its environment and/or in its own state. This type of knowledge describes components in terms of the physical quantities that characterize their state and the physical laws that rule their operation.
- *Teleological* knowledge: about the purpose and the way the design object is intended to be used. This type of knowledge describes the goals assigned to an artifact and enables designers to translate market requests into specific expected behavior of the artifact.
- *Functional* knowledge: about the behaviors and goals of the artifact itself.

This typology is useful for understanding the four dimensions present in every engineering domain. Top and Akkermans (1994) divide their full ontology in three parts (sub-ontologies) namely functional components, physical processes and mathematical constraints. The first one can be considered a combination of structural and functional knowledge, whereas the other two are in fact representations of scientific knowledge.

If we focus on particular but common engineering areas, the most important aspect is requirements elicitation. Darlington and Culley (2008) report on the process of building an ontology for supporting design requirement elicitation in engineering. Their approach includes three different ontologies: *engineering design requirements*, *product finish* and *machine motion*. The last one is specific to some particular kinds of engineering and, in comparison with the first one, is largely general to different ontological domains. Katranuschkov *et al.* (2002) also reported on a 'core engineering ontology' covering common categories, but the main part of their model is oriented to the interchange between CAD (computer-aided design) applications.; Teleological knowledge, as defined by Colombo *et al.* (2007), is also connected to requirements.

Many applications of knowledge-based systems in concrete engineering domains address specific problems or applications, but only a fraction of them use ontologies as the representation paradigm. For example, Blomqvist and Ohgren (2008) report on the method of constructing an ontology in the automotive supplier domain, dealing mostly with the artifacts. Ontologies in classical engineering fields also include models of physical objects, phenomena and laws, and they usually become integrated in CAD/CAE (computer-aided engineering) environments. For example, Yoshioka *et al.* (2004) report on a system in which the engineer builds a functional hierarchy from functional specifications and then maps the functional hierarchy into physical features that are building blocks for conceptual design. Lin and Harding (2007) report on an ontology for manufacturing, from which a few concepts such as `Project` and `Process` can be abstracted, even though their definition is primitive. Ahmed *et al.* (2007) reported on an ontology-development methodology for engineering design that can be used for extending an upper engineering ontology to particular cases.

In the specific domain of SE, Abran *et al.* (2006) reported on a formal ontology covering the concepts of the discipline. Their effort was based on the so-called Guide to the *Software Engineering Body of Knowledge* (SWEBOK)[4]. This work provides general principles for SE ontologies, some of which have been reused here. Other works propose the use of ontologies to bridge engineering tools; for example, Peachavanish *et al.* (2006) report on ontology-based interoperability for entities, actions and role definitions between GIS (Geospatial Information Systems) and CAD applications. Those are pairwise mappings for concrete engineering objects and actions that could be useful in evaluating the generality of a general model for engineering. Finally, a comprehensive survey of ontologies in the software domain can be found in Coral *et al.* (2006).

3 Main categories in engineering models

The first step for the development of the upper model presented here was setting the requirements that serve as evaluation criteria. One of the ways to determine the scope of ontologies is to elaborate a list of questions that an application using the ontology should be able to answer. These are often called competency questions (Gruninger & Fox, 1995). From this idea, we first stated a competency scope and then derived competency questions from a definition of the domain being modeled. This section identifies the main ontological categories derived from the definition of engineering. Those categories are later included into a general framework. The definitions are based on available ontologies, which are referenced by prefixes to the names of ontology elements; for example, those definitions taken from OpenCyc are prefixed by '`oc_`'[5].

3.1 The competency scope

The competency scope for engineering an upper ontology is stated as follows: 'providing support for the management and accountancy of the products and activities in engineering work, covering all the aspects involved in engineering as a general problem solving activity operating under technical and organizational constraints'. In consequence, competency questions addressed include support for both horizontal and vertical knowledge. The statement of scope determines the ambit of the ontology, although some clarification regarding the coverage of 'organizational

[4] `http://www.swebok.org/`

[5] The resulting OWL + SWRL files can be found at `http://www.cc.uah.es/ie/ont/engineering`

Table 1 Competency questions for the upper engineering ontology

ID	Competency question
q1	Which are the artifacts produced and used in engineering and how can they be classified?
q2	Which is the configuration of an engineering artifact?
q3	How are artifacts' functions defined?
q4	How are requirements defined and how they relate to each other?
q5	How are requirements connected to the artifacts or elements produced?
q6	What is a specification in engineering?
q7	What is a model in engineering?
q8	Which is the nature of the relation of engineering artifacts, documents, specifications and models?
q9	What are methods in engineering?
q10	How can engineering work be evaluated?

constraints' is still required. Here, the domain of organizations is considered out of scope, and constraints are only considered as they affect technical processes. This entails that the model does not include categories related to organizational structures or processes that surround engineering work. For example, this work can be considered complementary to a Knowledge Management ontology (Sicilia *et al.*, 2006) in the sense that the latter manages the dissemination, evaluation and transfer of knowledge about engineering products and engineering experience. However, they cover separate domains of work activity.

Note that the focus is on building a model that could be embedded in applications and tools to improve engineering productivity and effectiveness. In consequence, philosophical issues are considered only when they impact on practical issues. This aligns with the postulates stated by Quine (1948), that is, whatever ontology best serves the epistemological role in science merits adoption.

3.2 A definition and related competency questions

To ensure that the competency scope matches the widespread notion of engineering supported by ABET[6], we adopt ABET's definition of engineering:

> The creative application of scientific principles to design or develop structures, machines, apparatus, or manufacturing processes, or works utilizing them singly or in combination; or to construct or operate the same with full cognizance of their design; or to forecast their behavior under specific operating conditions; all as respects an intended function, economics of operation and safety to life and property.

From that definition, and considering the discussion in the previous section, a collection of competency questions covering different aspects was elaborated. These questions are summarized in Table 1.

Question `q1` deals with *what is designed or developed* by engineering. In the case of mechanical parts, mereological formulations (Colombo *et al.*, 2007) describe the structural knowledge of the artifacts that are tangible and solid (e.g. as those represented by `oc_SolidTangibleArtifact`). However, other artifacts such as software pieces (e.g. instances of `oc_ComputerCode`) posses a structure that does not entail any kind of significant topological relation, but relations such as *dependencies* that can only be understood in the context of program execution. In consequence, the categorization of artifacts needs to be flexible enough to cope with a variety of entities. Question `q2` is closely related to `q1` and concerns the alternate combinations of components that lead to a useful product. This goes a

[6] ABET (Accreditation Board for Engineering and Technology) certifies college and university programs in engineering and other disciplines (`http://www.abet.org/`).

step beyond structural knowledge to deal with the management of versions and configurations of different artifacts that end up in composed artifacts with different functions.

Questions q3 and q4 are, respectively, related to functional and teleological knowledge. Question q5 synthesizes the requirements traceability problem. Question q6 deals with the expression of design, and q7 covers the concept of model, which is pervasive in engineering. Then, the relation of artifacts produced, the documents that realize them, and the expressions of the design are all stated by question q8. Question q9 is related to the 'disciplined ways of working', that is, methods and techniques, which drive actual engineering activity. Finally, question q10 focuses on the essential scientific characteristic of engineering: activities and processes need to be subject to contrast and evaluation. This includes not only quantitative measures on the characteristics of the products, but also general criteria, principles and evaluation techniques.

Looking back at ABET's definition, there are some aspects that have been deliberately considered out of scope. For example, the 'creative' nature of engineering is a complex phenomenon for which there are no widely accepted theories, so it is left aside. Also, the operation of the products of engineering is only considered in the cases that they require engineering skills. Non-central issues such as safety and privacy can be introduced in the ontology as specific kinds of requirements.

Some of the competency questions can still be broken into more specific ones. For example, q2 is related to configuration, which entails the maintenance of versions in general.

The rest of this section discusses the main ontological commitments[7] of the upper engineering ontology. The discussion starts from the individuals that perform engineering actions, and continues with the origin of engineering work, followed by the results. After dealing with the inputs and outputs, the rest of the section discusses general ways of performing engineering work.

3.3 The engineers

First, the Engineers that perform the actual work can be characterized as a subset of the class oc_IntelligentAgent defined in OpenCyc as follows: 'An agent is an intelligent agent if and only if it is capable of knowing and acting, and capable of employing its knowledge in its actions.'

Engineers usually work in teams, and as a consequence the term oc_Organization can be reused. The belonging of agents to an organization is modeled in a generic form by the oc_groupMembers predicate. Then, a first definition of EngineeringOrganization as subsumed by oc_Organization can be implemented through an OWL restriction in the form oc_groupMembers some Engineer. A basic model of organizations is provided by the oc_subOrganizations predicate, which relates organizations to its subordinates, and allows us to model different kinds of organizations, groups or teams.

An important concept in most engineering domains is oc_Project, which are actually collections of oc_PurposefulActions. The oc_Project concept can be used as the root of a separate ontology for projects, which could be modeled as an extension by codifying the knowledge in the *Project Management Body of Knowledge* (PMBOK[8]). Projects include engineering activities as well as other activities such as training and deployment. Abels *et al.* (2006) discuss PROMONT, a project management ontology covering the DIN 69901 standard that could be used as the basis for such an effort.

3.4 The problems: requirements

Requirements correspond to teleological knowledge, as the expression of intended uses of the artifact to be designed. All design activities are aimed at translating functional requirements into structures that are able to realize them, and several relations between functions and components or parts (as HAS-A or PERFORMS-A) have been proposed by Colombo *et al.* (2007). In addition, predicates connecting requirements to functions, such as the predicate NEED-A can be stated.

[7] Terms and predicates in the ontology are typed in Courier font.

[8] http://www.pmi.org/

An initial concern is the status of different kinds of requirements. Lin *et al.* (1996) distinguish between *external* and *internal* requirements, the latter being posed by the engineering team. We deal only with external requirements, since the latter pertain to the decomposition of the initial design. In a different direction, Darlington and Culley (2008) provide a comprehensive classification of `DESIGN_REQUIREMENTs`, including economic, product, functional, ergonomic, legal and other requirements. There is, however, a need for particular definitions for each of them, for example, legal requirements might be defined by a required connection to `Legal_sources` according to the LKIF (Legal Knowledge Interchange Format) legal core ontology (Hoekstra *et al.*, 2007). These different definitions can be elaborated as extensions of a generic ontology, and an important observation is that most kinds of requirements can be translated into constraints on a particular kind of requirement, namely *functional requirement*. For instance, a performance requirement may be represented as a constraint on a particular functional requirement: the shape or disposition of an object can be constrained by a safety requirement, a legal requirement might constrain the exposure of certain information in a software application, etc.

In any case, the ontology deals only with *specifications*, since the requirements situate at a subjective level for the needs of individuals. In consequence, the specifications serve as the unique elements used in engineering activities. Functional requirement specifications in turn denote function specifications, that is, they are constrained by the OWL `oc_thingSpecified some FunctionSpecification` constraint. We maintain both specification entities because the former is a user requirement whereas the second is not specific to the subjectivity of users, but instead an *internal* requirement.

Furthermore, a distinction between `Functions and UserRequirements` needs clarification as it deeply influences the traceability of engineering elements and activities. Requirements are first associated to a concrete engineering *process* (rather than concrete products, that might have not been conceived at the time of requirement formulation). On the contrary, functions are the final uses of the devised artifacts. For instance, in OpenCyc, the primary function of a given object type is connected to a type of situation role as in the following example:

(`typePrimaryFunction Telephone AudioCommunicating deviceUsed`)

Given that the artifacts (`Telephone`) and the role they play (`deviceUsed`) are not necessarily determined *a priori*, specifying the situation is essential. The `oc_SituationSpecification` concept is flexible enough to cover this role. The synonym `FunctionSpecification` is then connected to the category of `UserFunctionalRequirementSpecifications`, and a generic `constraintRequirement` predicate can be used as a general-purpose relation covering relationships between requirements, and thus, also giving support to competency question `q4`. Note that the subspecification predicates of OpenCyc can be reused for the decomposition structure of user requirements.

The connection of requirements to engineering processes is implemented by relating them to `oc_Plans`, which are normative specifications for actions to be performed. Such plans can be later connected to projects or concrete actions, but this latter aspect is not further analyzed here. Figure 1 shows the subordination of requirements to plans, which in turn are connected to independent function specifications (`fs`). Then, functions are mapped to artifacts (or artifact types, `at`) that realize those functions. This mapping provides basic support for automating the search of knowledge to be reused during the initial stages of a project, when functional requirements are established.

To illustrate the above discussion, let us consider the following example about *mechanical seals*. A generic requirement specification that could originate the engineering of such artifact is: 'join systems or mechanisms together by preventing leakage'. This results in a `UserFunctionalRequirementSpecification` instance, which we will identify as `noLeakageInOilJunctions`. This instance will be connected to the `FunctionSpecification` instance `joinOilConductionWithoutLeakage`. This is a concrete requirement that is connected to a more generic `joinWithoutLeakage` generic function specification, as a way to abstract out the common aspects of different requirements. Doing so provides an opportunity to share design knowledge that could be reused in other contexts, which is acknowledged to be a driver in innovation processes (Hargadon, 2002).

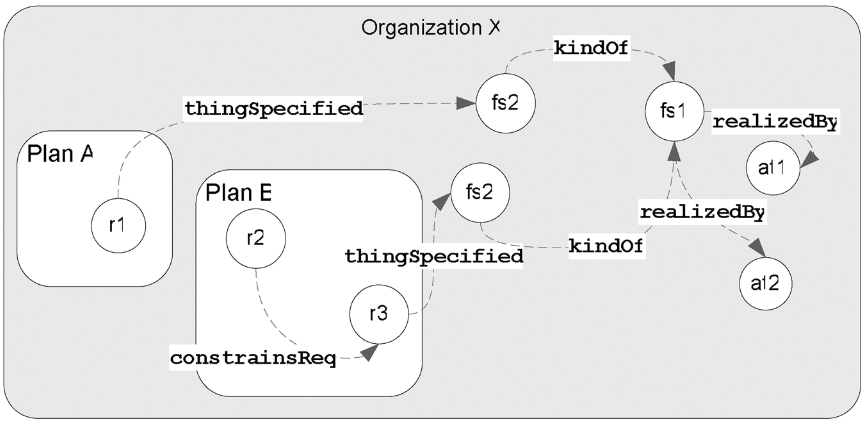

Figure 1 Requirements, function and artifacts

Although the requirement will be tied to a particular plan of an organization, this will not be discussed here[9] as it belongs to the project management scope.

Lin *et al.* (1996) modeled some basic properties of requirements, such as decomposition, that are considered in the upper ontology described here. An interesting definitional characteristic of their model is the expression of requirements as logical expressions. However, these properties do not apply to all possible requirement specifications, so in our model they are left to the rules layer provided by SWRL, which can be attached in a modular way to the ontology as described in the next section.

Other important aspects of requirements in the domain of engineering are the following:

1. Requirements need to be *documented.*
2. Requirements need to be related to the design artifacts they are origin or constraint of.

Aspect (1) raises a general ontological commitment—the differentiation of the documents and the specifications themselves. This is realized in OpenCyc with the categories of `oc_InformationBearingThings` and `oc_PropositionalInformationThings`, providing support to the propositional content and to the thing that conveys it, respectively. For instance, a requirements document can be broken into several documents, but the propositional content is unique irrespective of its form (digital or hardcopy). When speaking about the engineering process, the important part is the propositional content. As the concrete things have some degree of arbitrariness in formatting, they are only important for cataloging processes specific to each project.

Even though aspect (2) can be generically realized by linking `UserRequirementSpecifications` to `oc_Artifacts`, the concrete semantics of these links are left to sub-properties which will specify particular constraints. Traceability (`q5`) from requirements to artifacts starts by such links, but other predicates are required to link the chain of artifacts produced by the different steps in the engineering process. This will be studied in the following sub-section.

3.5 The activities

Engineering is basically an *artifact-producing activity* performed by engineers. At this level, engineering can be seen as a flow of activities. Ideally, every activity, its doers and the artifacts used, changed or created should be represented. This consideration does not take into account the ways of carrying out the activities (the *methods*) but only of the representation of the activities actually enacted. In fact, this is the recording of actual, real empirical experience of engineering as a human activity. In OpenCyc, activities can be represented as `oc_Action` instances. These actions are defined as ‘the collection of `oc_Events` that are carried out by some “doer” (see `oc_doneBy`). Instances of `oc_Action` include any event in which one or more actors effect some

[9] A complete example can be found in the Web of the ontology

change in the (tangible or intangible) state of the world, typically by an expenditure of effort or energy.' An `oc_Event` is in turn 'a dynamic situation in which the state of the world changes; each instance is something one would say, "happens"'. Furthermore, engineering activities are in fact `oc_PurposefulActions`. Each instance of `oc_PurposefulAction` is 'an action consciously, volitionally and purposefully done by at least one actor'. Then, `EngineeringAction` instances define the scope of horizontal knowledge, and are able to represent the day-to-day activities in engineering projects.

There are several `situationRoles` for `EngineeringActions` to be mentioned as characteristic of engineering activities of any kind, including those that are not carried out by specific techniques or methods:

- The actions are decomposed into `oc_subEvents`.
- The events are `oc_performedBy` some concrete professional roles.
- The `oc_inputs` and `oc_outputs` for the activity.
- The collection of `oc_objectActedOn` that are altered or affected by each activity.

The `actionRole` predicate can be defined to place specific constraints on particular classes or `engineeringActions`. For example, actions that belong to the type 'design mechanical part structures' can be constrained to use concrete kinds of CAD programs as `Tools`. Also the responsibility for approving a change in the configuration of a product can be specified as pertaining to a change authority (a group or an individual).

Following the above example, a given company may undertake an `oc_Project` in which the main of the `oc_outputs` is a new seal (an instance of `ArtifactType`). This is the highest possible abstraction level, and it is compatible with a refinement of the project as a complex activity into more concrete actions.

3.6 The results: artifacts

Artifacts are the main products of engineering, but it is important to clearly differentiate engineering artifacts. The basis of our model of artifacts is the OpenCyc definition for the concept: 'Each instance of `oc_Artifact` is an at least partially tangible thing which was intentionally created by an `oc_Agent` (or a group of `oc_Agents` working together) to serve some purpose or to perform some function. In order to create an instance of `oc_Artifact`, it is not necessary that an `Agent` creates the matter out of which the `Artifact` is composed; rather, an `oc_Agent` can create an instance of `oc_Artifact` by assembling or modifying existing matter.' That definition is actually a superset for the category of `EngineeringArtifacts`, which are defined as those artifacts coming from engineering actions.

Non-essential characteristics from artifacts are the following:

- Identifiability; the capability of uniquely identifying the elements produced in engineering.
- Traceability; that is, recording the history of relations of input–output between engineering artifacts.
- Identification of functionality; the capability of tracking any artifact back to the requirement that originated it.

Identity (Welty & Guarino, 2001) of engineering artifacts is essentially dependent on the engineering process that devises or creates them. This entails the introduction of particular engineering processes (horizontal knowledge) in the model.

The traceability in the production of artifacts is an important element in engineering models, for which the transitive `tracesTo property` can be used. The following rules in SWRL provide two examples for such property in which ?*x* can be used as a parameter with the artifact that we want to trace, including identification of functionality.

```
    Artifact-EngineeringProduct(?x)
and Artifact-EngineeringProduct(?y)
and Artifact-EngineeringProduct(?z)
and tracesTo (?x, ?y)
and tracesTo (?y, ?z)
then tracesTo(?x, ?z)

    Artifact-EngineeringProduct(?x)
and Artifact-EngineeringProduct(?y)
and UserRequirementSpecification (?r)
and tracesTo(?x, ?y)
and tracesTo(?y, ?r)
then tracesTo(?x, ?r)
```

The structural model defined by Lin *et al.* (1996) corresponds to this mereology of parts that is generically realized by the `oc_properParts` predicate. The predicate can be extended for more specific meanings in different engineering contexts. Furthermore, proper parts predicates can be refined to model both structural and functional knowledge. Structural knowledge, for instance, is captured in the case of physical artifacts by the `oc_properPhysicalParts` specialized predicate.

Another important concept is *specifications*, which are pervasive at the normalization level (standards, recommendations and the like) as well as a way to interface different activities in the project life cycle. Specifications at the normalization level are modeled by the general `Standard` concept, which can be specialized into different sub-concepts such as `ISOStandards` (those `definedBy` the ISO organization), and product standards (those defining an `ArtifactType`, which in OWL is expressed as the restriction `oc_thingSpecified some ArtifactType`).

Specifications, in general, relate to any thing through the `oc_thingSpecified` predicate. The term `oc_ModelArtifact` provides the appropriate semantics for the concept of model: 'a collection of artifacts; a subset of `oc_VisualInformationBearingThing`. Each element of `oc_ModelArtifact` is a tangible object designed to resemble and/or represent some other object, which may or may not exist tangibly.'

In OpenCyc, Information Bearing Things (IBTs) are defined as 'a collection of spatially-localized individuals, including various actions and events as well as physical objects […] each instance of `oc_InformationBearingThing` (or "IBT") being an item that contains information (for an agent who knows how to interpret it)'. As information bearing objects, engineering models are IBTs as well, so that their contents can be represented in a propositional form, through the predicate `oc_containsInfoPropositional-IBT` IBT Propositional Information Thing (PIT), that links to a PIT. PITs can be in themselves microtheories (internally consistent sub-ontologies in OpenCyc), thus allowing the definition in logical terms of the actual contents of the model. In OWL, this relation can be represented by subclasses of the model outside of the upper ontology for engineering. These subclasses provide full representational capabilities for specific kinds of artifacts and are connected to some `oc_Artifact` or `ArtifactType` by a `hasModel` predicate. This could, for example, be applied to develop systems that represent UML (Unified Modeling Language) diagrams[10] through logics, which will enable a degree of increased automation. This also holds for specifications. Figure 2 depicts an example of this kind of mapping, in which two models with different viewpoints are attached to the same product.

The separation of models and the actual artifacts or artifact types support incompleteness in the design process, which is a key characteristic of abstraction-intensive activities as engineering. Indeed, model-driven engineering is seen as a way to tackle with complexity in some disciplines (Schmidt, 2006), and this entails progressive refinement of models. As a consequence, attribute constraints, such as those proposed by Lin *et al.* (1996), can be associated to models and enables: (a) checking the consistency of different views (partial models on the same product), by checking that the constraints are compatible; and (b) checking the compatibility of the models produced

[10] `http://www.uml.org/`

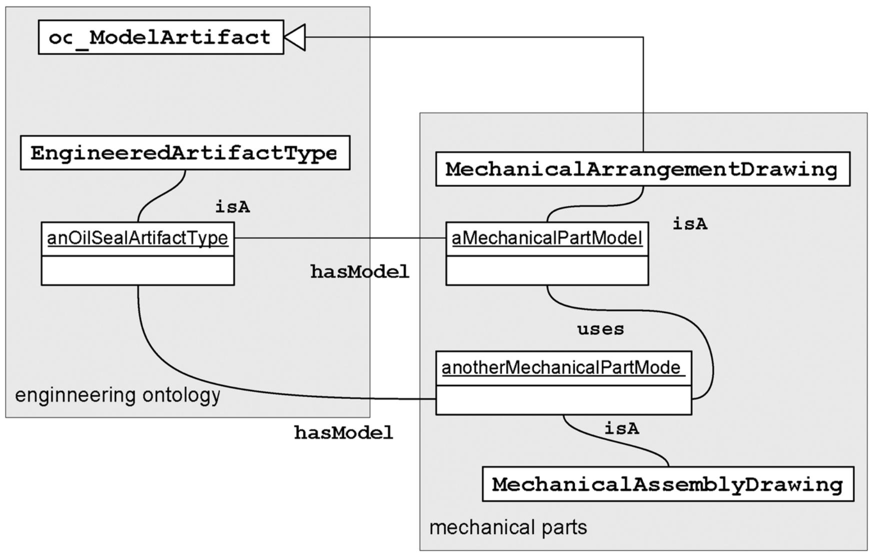

Figure 2 Requirements, functions and artifacts

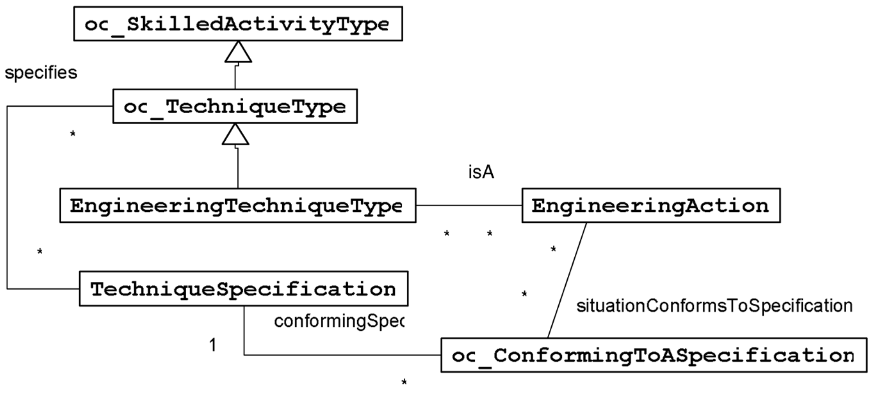

Figure 3 Techniques, technique specifications and engineering actions

during engineering work, and final attributes of the artifacts. This latter case is useful for automating the control on design and production.

3.7 The way of doing: techniques and methods

Specifications and models capture an important portion of the way engineers work. An ontologically different concept related to activities in engineering is that of 'methods' for activities, that is, the specification of 'blueprints' for potential courses of activity or 'ways of doing'. These specifications have an intrinsically prescriptive character, so they should not be specified as actions, but rather as specifications. Figure 3 depicts the definition of techniques and skilled activity types in relation to OpenCyc definitions.

The `oc_Actions` are particular kinds of `oc_Events` that can be related to `oc_Specifications` by `oc_conformingToASpecification` instances. The conformance of events or any other kind of situation is a common pattern in engineering, since normalization and reuse of engineering artifacts is a key characteristic of engineering work in modern industry.

Techniques are usually considered specific applications of a *method*, or more specific ways of carrying out some courses of action. Methods can then be specified as subsumed by oc_SkilledActivityType and be subject to contain or mandate some techniques. Specifications and the relation to engineering actions can be modeled in the same way as techniques depicted in Figure 3.

The oc_competentAtSkill predicate allows us to relate individuals to competencies. This can be used to link human resource management to the engineering ontology, but it is outside the scope of the latter. Ontologies of competencies, as the one discussed by Monceaux *et al.* (2007), could be used in this context.

The Methods are considered collections of techniques plus some additional elements as common notations or principles. The difference between technique and method resides in that the latter are of a finer grain and concrete, while the former may contain several techniques. Methods may share some techniques, and they might also share notational frameworks or common principles.

3.8 The contrast: indicators and criteria

Many aspects in engineering deal with contrasting the quality of either the artifacts produced or the methods used, which is in the scope of competency question q9. Some of them are part of specific engineering activities, often called testing or evaluation. However, there are others that fall into the category of normative knowledge such as principles and measurement.

Principles can be classified as those specific to the supporting sciences, and those specific to the activities and products of engineering. The former include any mathematical or physical principle used during design. For instance, the *bending moment* principle for building strong structures in civil engineering. However, the representation of these kinds of knowledge is outside the intended competency of the ontology discussed here, and existing ontologies can be used for that purpose (Stahovich *et al.*, 1993; Gruber & Olsen, 1994). Principles in engineering disciplines take the form of general rules on ways of doing, and their practical formulation requires the use of measures and their interpretation. This is the rationale for introducing indicators as a key characteristic of engineering contrast.

Indicators are models purposefully prepared to evaluate parts of processes or products. They involve the Measurement of one or several attributes that may be later aggregated into a contrastable value. While measurements directly relate to attributes, indicators do not. Since indicators are very specific of concrete kinds of artifacts or activities, the upper model does not provide further additional detail. The computing from measures to indicators, if needed, is modeled through rules, and measurement can be expressed by reusing existing measurement ontologies (Kim *et al.*, 2005).

3.9 Other elements

Another important characteristic of engineering is that of being *tool-intensive*. This concept is captured by the oc_Tool term, representing 'user controlled devices that help to manipulate or alter other objects or the immediate environment in some way'. However, contemporary engineering is increasingly based on software tools, and they do not fit in the oc_Tool definition, tied to solid, rigid things. A Tool concept subsuming this and a SoftwareTool category has been used to cover the spectrum of tools used in engineering.

4 The ontology of Software Engineering: concepts and structure

Many engineering definitions emphasize the disciplined and systematic artifact creation. However, the material object produced by every engineering discipline is not necessarily of a similar nature. The case of SE is particularly relevant in the illustration of such differences, since software as an artifact is acknowledged as a very special piece of human work. The special nature of software was attributed by Brooks (1987) to 'complexity' as an essential characteristic.

This section provides a case study on how ontologies for particular engineering disciplines can be built by extension to the upper engineering ontology discussed so far. The idea of disciplinary

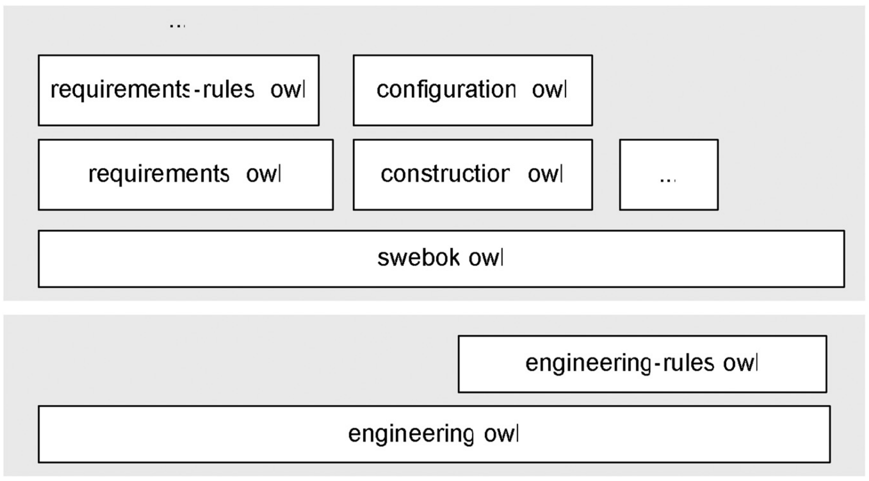

Figure 4 Arrangement of ontology fragments for reusability

extensions is that of producing modular ontology fragments. Figure 4 depicts the concrete arrangement for the SE extension.

The principles of the decomposition are two-fold. On one hand, SWRL rules are separated from OWL definitions to allow for versions that support or not the SWRL reasoning. On the other hand, there should be a separation of the domain-specific part that follows some kind of standard convention. Consensus-reaching approaches to ontology engineering are deemed appropriate for the crafting of representations of the concepts of some concrete domain. Nonetheless, in some domains the engineer can find pre-existing processes of consensus-reaching on conceptual frameworks. This is the case of SE, in which the Guide to the SWEBOK project is the result of a considerable effort on the collaborative production of a subset of the knowledge of the discipline that is, as of today, subject to little controversy in the community of researchers. The SWEBOK project aims at defining the boundaries of SE, which motivates the fundamental organization of the Guide. The material that is recognized as being within this discipline is organized into Knowledge Areas (KAs), including requirements, construction and configuration, among others. In what follows, the principal aspects of the upper engineering ontology presented so far are discussed as applied to the field of SE.

4.1 Software requirements

Typically, user-prescribed functionality is mapped in design time to software components, which expose functionality to other components. Then, required functionality can be defined as the functions at the level expressed in functional requirements, emphasizing that we stay at the level of final user functionality. Even in the case that functional requirements are expressed for a software without a user interface (e.g. a software framework), the distinguishing characteristics of required functionality in the sense provided herein is that it was stated as functional requirements in the context of engineering. That is, user requirements are *performative*, in the sense that they trigger development activities.

Requirements are concerned with several kinds of engineering actions, namely: elicitation, analysis, specification and validation. These four classes apply to any engineering domain and are yet identified at the level of the upper ontology for engineering discussed. In fact, elicitation techniques such as *use cases* (Bittner & Spence, 2002) can be used in other engineering areas, even though they were developed as specific to software systems.

A `SoftwareRequirementSpecification` is a `UserRequirementSpecification` that specifies a property that must be exhibited by a software system to solve a problem in the real world. The differentiation of requirements in different disciplines can be expressed in terms of OWL class constraints. In the case of SE, such constraint would be the following.

```
eng:tracesTo only swebok:SoftwareEngineeringArtifact
```

Where `swebok:SoftwareEngineeringArtifact` is subsumed by the generic `eng:Artifact-EngineeringProduct`. Since traces are transitive, any software requirement will finally trace to a SE artifact.

Usually, two requirement categories are considered: functional and non-functional. The former can be mapped to `UserFunctionalRequirementSpecification`. The latter can be defined by exclusion through a disjoint axiom with the former, and the `constrainsRequirement` can be used to make explicit their connections.

Functional requirements are, in many projects, modeled as use cases, which requires a specific model schema as depicted in Figure 2. Use cases are thus a unit to check the validity of the software system built, and as such they can be used to automatically check the coverage of test cases. The `mapToUseCase` links `UserFunctionalRequirementSpecifications` to `UseCaseSpecifications`, which are subsumed by the former. In this way, inferences on traces are enabled. Use cases are in turn specified by scenarios, and these scenarios contain interactions between objects in the conceptual UML model. The following is a simple SWRL formulation of such coverage checking.

```
    UseCaseSpecification(?uc)
and hasScenario(?uc, ?s)
and contains(?s, ?o)
and hasClass(?o, ?c)
then eng:tracesTo(?uc, ?c)

    eng:tracesTo(?uc, ?c)
and test:hasUnitTest(?c, ?t)
then partiallyCovered(?uc)
```

Even in the case that the class ?*c* (a specific type of engineering artifact) does not have the same name or it is split up into several classes, the transitive trace relationships between classes will ultimately trigger the coverage inference. More detailed rules could be used to test the particular calls in use case scenarios.

4.2 *Software as an artifact*

The view of software that needs to be addressed here is actually that of software specification. There has been a considerable debate on the formal/informal form of such specifications—a review of the main original issues can be found in Colburn (2000: 129). However, the discussion here concerns a given specification, assuming it correctly captures the real world entities being modeled and the right user needs. In addition, software has a dual nature concerning its form of expression and its form of execution. But since we only deal with the inputs and expected outputs of software behavior (to be developed yet, functioning software or only specifications), the discussion is not relevant. From the viewpoint of the coming discussion, we will start from some definitions on the OpenCyc ontology. Figure 5 provides a fragment of the OpenCyc ontology expressed in UML. It depicts a selection of concepts in OpenCyc and their relations (predicates) relevant to our present discussion.

Figure 5 shows that there is a class of objects—`oc_ProgramSpecifications`—to determine the expected behavior of `oc_ComputerPrograms`. The OpenCyc concept `oc_ProgramSpecification` is defined as '[…] not a computer program itself (i.e. lines of code), but an abstract characterization of how a program should behave. For instance, a sorting program can be specified by requiring that the program's output be a list of the same elements as the input such that no element follows an element that is greater than it.' Functional specifications are obviously subsumed in that category. A problem arises with the granularity of what a 'program' is considered. `oc_ProgramSpecification` instances are not limited to 'single, discrete programs',

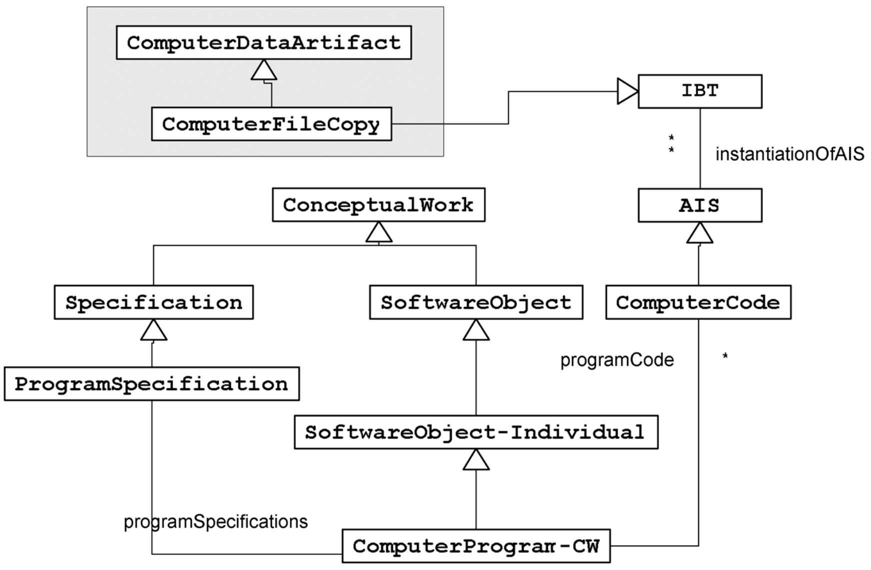

Figure 5 Software and specifications of software

thus, the mapping of computer programs (as conceptual works) and specifications is actually conventional and covers several cases:

1. A specification covering a number of programs (e.g. in the case of a protocol specification).
2. A specification covering a single program.
3. A specification which is only part of a program functionality.

The OpenCyc concept `oc_ProgramStepSpecification` serves to cover only a part of a program to deal with case (3) that is not clearly covered in `oc_ProgramSpecification`.

There is also a relevant distinction that appears in Figure 5 between those computer programs considered `oc_ConceptualWorks` and the `oc_ComputerCode` (source or binary) that realizes them. In turn, computer code is actually an abstract information structure—an abstract individual comprising abstract symbols and relations between them, as defined in OpenCyc—that has as instantiations IBTs, that is, computer file copies containing the computer code. For example, some given software like the statistical package StatGraphics 7 can be modeled as an instance of `oc_ComputerProgram-CW` (a specific kind of `oc_ConceptualWork` in the form of a computer program). Then, each of the distributions for different platforms can be specified by the following: 'the code in which an instance of `oc_ComputerProgram-CW` is expressed constitutes an instance of `oc_AbstractInformationStructure` that can be related to the program it expresses using the predicate `oc_programCode`'. In summary, `oc_ComputerCode` instances are realizations of `oc_ComputerProgram-CW`, but we are concerned with the specifications of the latter, which in turn are instances of `oc_ProgramSpecification`. Since conceptual works are not artifacts, and the upper ontology for engineering is expressed in terms of artifacts, the configuration of a software product as baseline under configuration management can be considered a special kind of `ArtifactType`, which is realized in the copies released.

4.3 Metrics and principles on software design

`SoftwareEngineeringPrinciples` are generic statements on the main guiding criteria for software development. *Modularization* is considered one of the key principles of software design, as acknowledged in the SWEBOK Guide. For the principle to become subject to automation, the use of concrete metrics is mandatory (Fenton, 1991). For example, Martin's instability[11] and

[11] `http://www.objectmentor.com/resources/articles/stability.pdf`

abstractness metrics can be used to assess the quality of the design. A possible rule for checking one of the criteria for the category represented by a software package could be:

```
SoftwarePackage(?p) and AbstractnessMeasure(?am) and measures(?am, ?p)
                    and InstabilityMeasure(?im) and measures(?im, ?p)
                    and value(?am, ?x1) and value(?im, x2)
                    and swrlb:lessThan(?am, 0.1)
                    and swrlb:lessThan(?im, 0.1)
                    then isPotentiallyRigidCategory(?p)
```

The *abstracteness* measure is computed as $\frac{number-of-abstract-classes-in-p}{number-of-classes-in-p}$ and the instability requires counting dependencies between source code elements. `oc_ComputerCode-Source` can be used to trace the dependencies in the source code artifacts for the rule above. Since dependencies are a particular kind of relationship between computer code instances that has a counterpart at the model-level in the UML language, they can enter the support system via source code analysis performed at the level of programming tools such as *Netbeans*[12]. The outcome of the evaluation comes in the form of predicates that specialize `indicatorInterpretation`, so that they can be clearly distinguished from factual information.

5 Conclusions and future work

Engineering can be characterized as a tool-intensive activity in which scientific and technical knowledge are used for the production of useful artifacts. As a consequence, a considerable number of software tools have been developed to support different aspects of concrete engineering domains. For example, CAD tools typically help in the design process of mechanical artifacts, while CASE (computer-aided software engineering) tools help the management of software elements.

Formal ontologies are a way to model domains through formal logics that enable rich and complex models with inference support. As such, the applications of ontologies to engineering are diverse and may cover many different aspects of engineering activity, supporting both design and management. Ontologies have started to be used for the purpose of building engineering tools in different domains. However, to date, no general, upper level model for engineering activity has been elaborated. This paper reports on a first attempt in that direction, which can be used as a point of departure for further elaboration, and as a template for ontologies tied to specific domains. The benefits of having a common model is that it factors out common characteristics, enabling reuse of knowledge representations. Reuse is a key element in ontology engineering since it helps in increasing the quality of models that otherwise could neglect important facets of commonsense knowledge. Physical things, mereology, topology activities and causality are examples of typical ontological dimensions that can be found in generic upper ontologies (Batres *et al.*, 2007). In the domain of engineering, some of these concepts are narrowed in more specific terms, and others are characteristic (but not exclusive) of engineering, such as artifacts, methods and models. A general ontology for engineering thus becomes a useful tool as a mechanism to support reuse. The general engineering ontology described in this paper builds on existing open commonsense ontologies, by mapping upper concepts to concrete ontology elements that could be used in applications for reusing these large ontologies.

The ontology described can be used for the development of applications supporting different aspects of the engineering process, for example, configuration management, modeling tools, etc. The case studies provided include traceability of requirements and inferred information from indicators, as examples of potential uses that can be generically embedded in the declarative form of rules. Organizations managing their processes with ontology-enabled tools would benefit from a flexible infrastructure prepared for inference and partial automation of processes (Sicilia & Lytras, 2005).

[12] `http://uml.netbeans.org/`

Future work will progress in two main directions, namely refining common semantics and building engineering domain specific extension for given applications. Eventually, some of the results of the latter could be re-factored to a generic version useful for several (or all) of the engineering domains.

Acknowledgements

This work has been partially supported by project SERE-OC (Resource selection in e-learning based on competency ontologies), code CCG06-UAH/TIC-0606 funded by the University of Alcalá and the CAM (Comunidad Autónoma de Madrid).

References

Abels, S., Ahlemann, F., Hahn, A., Hausmann, K. & Strickmann, J. 2006. PROMONT—A project management ontology as a reference for virtual project organizations. OTM Workshops, Lecture Notes in Computer Science, **4277**, 813–823. Springer.

Abran, A., Cuadrado-Gallego, J. J., García-Barriocanal, E., Mendes, O., Sánchez-Alonso, S. & Sicilia, M. A. 2006. Engineering the ontology for the SWEBOK: issues and techniques. In *Ontologies for Software Engineering and Software Technology*, Calero, C., Ruiz, F. & Piattini, M. (eds), Springer, 103–122.

Ahmed, S., Kim, S. & Wallace, K. M. 2007. A methodology for creating ontologies for engineering design. *Journal of Computing and Information Science in Engineering* **7**(2), 132–140.

Auyang, S. 2004. *Engineering—An Endless Frontier*. Harvard University Press.

Batres, R., West, M., Leal, D., Price, D., Masaki, K., Shimada, Y., Fuchino, T. & Naka, Y. 2007. An upper ontology based on ISO 15926. *Computers & Chemical Engineering* **31**(5–6), 519–534.

Bittner, K. & Spence, I. 2002. *Use Case Modeling*. Addison Wesley Professional.

Blomqvist, E. & Ohgren, A. 2008. Constructing an enterprise ontology for an automotive supplier. *Engineering Applications of Artificial Intelligence* **21**(3), 386–397.

Brooks, F. 1987. No silver bullet: essence and accidents of software engineering. *IEEE Computer* **20**(4), 10–19.

Colburn, T. 2000. *Philosophy and Computer Science*. M.E. Sharpe.

Colombo, G., Mosca, A. & Sartori, F. 2007. Towards the design of intelligent CAD systems: an ontological approach. *Advanced Engineering Informatics* **21**(2), 153–168.

Coral, C., Ruiz, F. & Piatinni, M. 2006. *Ontologies for Software Engineering and Software Technology*. Springer-Verlag.

Darlington, M. J. & Culley, S. J. 2008. Investigating ontology development for engineering design support. *Advanced Engineering Informatics* **22**(1), 112–134.

Dias, W. P. S. 2007. Philosophical grounding and computational formalization for practice based engineering knowledge. *Knowledge-Based Systems* **20**(4), 382–387.

Fenton, N. 1991. *Software Metrics: A Rigorous Approach*. Chapman & Hall.

Gangemi, A., Guarino, N., Masolo, C., Oltramari, A. & Schneider, L. 2002. Sweetening ontologies with DOLCE. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, 166–181.

Gruber, T. & Olsen, G. 1994. An ontology for engineering mathematics. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning, Gustav Stresemann Institut, Bonn, Germany*, Doyle, J., Sandewall, E. & Torasso, P. (eds). Morgan Kaufmann, 258–269.

Gruninger, M. & Fox, M. S. 1995. Methodology for the design and evaluation of ontologies. In *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing*, IJCAI-95, Montreal.

Hargadon, A. B. 2002. Brokering knowledge: linking learning and innovation. *Research in Organizational Behaviour*, volume 24. Elsevier, 41–85.

Hoekstra, R., Breuker, J., Di Bello, M. & Boer, A. 2007. The LKIF Core ontology of basic legal concepts. In *Proceedings of the Workshop on Legal Ontologies and Artificial Intelligence Techniques (LOAIT'07)*, Casanovas, P., Biasiotti, M. A., Francesconi, E. & Sagri, M. T. (eds). http://www.ittig.cnr.it/loait/LOAIT07-Proceedings.pdf

Katranuschkov, P., Gehre, A. & Scherer, R. J. 2002. An engineering ontology framework as advanced user gateway to IFC model data. In *Proceedings of ECPPM 2002—eWork and eBusiness in Architecture, Engineering and Construction*, Turk Z. & Scherer R. J. (eds). A.A. Balkema, 269–276.

Kim, H., Sengupta, A., Fox, M. & Dalkilic, M. 2005. A measurement ontology generalizable for emerging domain. *Journal of Database Management* **18**(1), 20–42.

Lenat, D. 1995. Cyc: a large-scale investment in knowledge infrastructure. *Communications of the ACM* **38**(11), 33–38.

Lin, K. & Harding, J. A. 2007. A manufacturing system engineering ontology model on the semantic web for inter-enterprise collaboration. *Computers in Industry* **58**(5), 428–437.

Lin, J., Fox, M. & Bilgic, T. 1996. A requirement ontology for engineering design. *Concurrent Engineering: Research and Applications* **4**(4), 279–291.

Monceaux, A., Naeve, A., Sicilia, M. A., García-Barriocanal, E., Arroyo, S. & Guss, J. 2007. Targeting learning resources in competency-based organizations. In *The Semantic Web: Real-World Applications from Industry*, Cardoso, J., Hepp, M. & Lytras, M. (eds). Springer, 143–167.

Niles, I. & Pease, A. 2001. Towards a Standard Upper Ontology. In *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, Welty, C. & Smith, B. (eds), Ogunquit, Maine, October 17–19.

Parnas, D. L. 1998. Software engineering programmes are not computer science programmes. *Annals of Software Engineering* **6**, 19–37.

Peachavanish, R., Karimi, H. A., Akinci, B. & Boukamp, F. 2006. An ontological engineering approach for integrating CAD and GIS in support of infrastructure management. *Advanced Engineering Informatics* **20**(1), 71–88.

Polanyi, M. 1966. *The Tacit Dimension*. Doubleday.

Quine, W. V. 1948. On what there is. *Review of Metaphysics* **2**, 21–38.

Schmidt, D. C. 2006. Model-driven engineering. *IEEE Computer* **39**(2), 25–31.

Sicilia, M. A. & Lytras, M. 2005. The semantic learning organization. *The Learning Organization* **12**(5), 402–410.

Sicilia, M. A., Lytras, M., Rodríguez, E. & García-Barriocanal, E. 2006. Integrating descriptions of knowledge management learning activities into large ontological structures: a case study. *Data and Knowledge Engineering* **57**(2), 111–121.

Sowa, J. 2000. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks/Cole.

Stahovich, T. F., Davis, R. & Shrobe, H. 1993. An ontology for mechanical devices. In *Proceedings of the AAAI-93 Workshop on Reasoning About Function*, Washington, DC, 137–140.

Top, J. & Akkermans, H. 1994. Tasks and ontologies in engineering modeling. *International Journal of Human–Computer Studies* **41**(4), 585–617.

Vincenti, W. G. 1993. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. Johns Hopkins University Press.

Welty, C. & Guarino, N. 2001. Supporting ontological analysis of taxonomic relationships. *Data and Knowledge Engineering* **39**(1), 51–74.

Yoshioka, M., Umeda, Y., Takeda, H., Shimomura, Y., Nomaguchi, Y. & Tomiyama, T. 2004. Physical concept ontology for the knowledge intensive engineering framework. *Advanced Engineering Informatics* **18**(2), 95–113.