Engineering Applications of Artificial Intelligence 56 (2016) 230–249

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

A framework to represent, capture, and trace ontology development processes

Marcela Vegetti[a], Luciana Roldán[a], Silvio Gonnet[a], Horacio Leone[a], Gabriela Henning[b,*]

[a] INGAR (CONICET/Universidad Tecnológica Nacional), Avellaneda 3657, 3000 Santa Fe, Argentina
[b] INTEC (Universidad Nacional del Litoral, CONICET), Ruta Nacional 168, Km 0, 3000 Santa Fe, Argentina

ARTICLE INFO

Keywords:
Ontological engineering
Design rationale
Ontology development process

ABSTRACT

This article presents OntoTracED, a comprehensive framework to represent, capture and trace ontology development processes. It has three components: (i) a conceptual model that defines the framework foundations, (ii) an ontological engineering domain model (OEDM), which specifies and describes design objects, as well as those operations that are particular to a specific ontology development methodology, and (iii) a computational support environment, named TracED(aaS). This contribution first offers an overview of the ontology development process characteristics and then describes the main features of each OntoTracED component. The framework capabilities are illustrated by means of a case study addressing the use of TracED(aaS) throughout the development of an ontology of industrial interest. It is shown that this proposal makes a strong contribution in the ontological engineering field since the whole ontology development process, its history, rationale, and all the intermediate products can be captured in an integrated fashion.

1. Introduction

Ontologies are currently applied in many fields like medicine, finance, education, and biology; similarly, its usage in the engineering domain has grown in recent years. In manufacturing, for example, ontologies have been developed to: i) formalize knowledge about product data and production processes (Panetto et al., 2012, Vegetti et al., 2011; Giménez et al., 2008, Marquardt et al., 2010); ii) define, describe, and standardize vocabulary, concepts and relations between system activities and components (Efthymiou et al., 2015; Borgo and Leitão, 2007; Usman et al., 2013); iii) design multi agent and knowledge based systems for process supervision, monitoring and control (Darmoul et al., 2013; Elhdad et al., 2013; Natarajan et al., 2012). In addition, several ontologies have been developed in order to reach interoperability between multiple Information and Communication Technologies (ITC) (Wu et al., 2013; Beydoun et al., 2014; Liao et al., 2016).

Despite the fact that the number of ontologies developed by the academic community has increased in the last decade, the adoption of this technology by industry is still limited, as opposed to what happens in other fields, like bioinformatics. In fact, there is an important gap between scientific studies and real world applications, though there is a consensus on the need for concrete applications in various engineering fields. The development and usage of ontologies in engineering sciences is particularly challenging due to various reasons:

(i) It requires the collaboration among experts in the specific field, as well as specialists in knowledge engineering, ontology engineering and computer science (Hai et al., 2011). However, these last ones are not always available and domain experts demand for tools helping them along the ontology development phases.
(ii) Ontologies are built to be reused or shared anytime, anywhere, and independently of the behavior and domain of the application that adopts them (Fernández-López et al., 1999). Building a consensual and rich ontology in any engineering domain is, however, not an easy task, as it requires the proper handling of the compromise between usability and reusability (Marquardt et al., 2010) and the agreement of different people on various aspects. Particularly, in most engineering domains, in which there are many pre-existing organizational information sources and vocabularies, as well as rapidly changing requirements, reuse is hard to achieve without appropriate methodological approaches and support tools.
(iii) Given the complexity of most engineering domains, as well as the numerous possible applications of a given ontology in a particular domain, such ontology cannot be expected to be complete from

* Corresponding author.
E-mail addresses: mvegetti@santafe-conicet.gov.ar (M. Vegetti), lroldan@santafe-conicet.gov.ar (L. Roldán), sgonnet@santafe-conicet.gov.ar (S. Gonnet), hleone@santafe-conicet.gov.ar (H. Leone), ghenning@intec.unl.edu.ar (G. Henning).

http://dx.doi.org/10.1016/j.engappai.2016.10.001
Received 2 February 2016; Received in revised form 12 July 2016; Accepted 3 October 2016

> scratch. On the contrary, its development must anticipate the possibility of future extensions and reuse, thus adding an extra difficulty. To our knowledge, with the exception of OntoCAPE (Morbach et al., 2007, 2009; Marquardt et al., 2010; Hai et al., 2011), a comprehensive ontology for the domain of computer-aided process engineering (CAPE), there is hardly any ontology in the engineering sciences which can be broadly used and which is actually applicable. Even for the authors of OntoCAPE (Marquardt et al., 2010), a project that took almost two decades, an ontology is never ready for use in the sense that there will always be the need for adaptations and refinements to match the requirements of an envisioned application. In fact, they assert that the OntoCAPE project exemplarily shows that ontologies are dynamic information systems and that there is still a lot of room for more elaborated ontology engineering support tools.

The previous statements, as well as the experience of our group in developing ontologies in various engineering domains for about a decade (Gonnet et al., 2007a; Giménez et al., 2008; Vegetti et al., 2011; Vegetti and Henning, 2015), allow us to conclude that ontology development needs to be properly supported. Ontology engineering is a new field that focuses on both the ontology development processes and the methodologies aimed at guiding such processes. An extensive state-of-the-art overview of ontology engineering methodologies can be found in Gómez-Pérez et al. (2004).

Depending on the scenario that is faced, different types of ontology development processes can be identified. In fact, according to the NeOn team (Suárez-Figueroa et al., 2012) there are nine scenarios for collaboratively building ontologies and ontology networks. Scenario number one considers the development of ontologies from scratch. There are scenarios for reusing and reengineering non-ontological resources and for reusing ontology design patterns. The other six scenarios consider combinations of reusing, reengineering and merging existent ontologies. These scenarios can be combined in different and flexible ways. However, any combination of them should include the scenario number one (ontology development from scratch) because it is made up of the core activities that have to be performed in any ontology development process.

In the last decade, ontology development processes devoted to build ontologies from scratch have changed from old-fashioned ones, performed by isolated knowledge engineers or domain experts, into collaborative processes, executed by mixed teams (Bernaras et al., 1996). In such teams, experts in knowledge acquisition and modeling, domain specialists, and experts in ontology implementation languages collaborate to build ontologies, according to well-established methodologies. In general, once a given development process is finished, the things that remain are mainly design products (e.g., requirements specification, competency questions, ontology class diagrams, implementations in specific languages, etc.), without an explicit representation of how these products have been obtained. In other words, the development process itself is not tracked and the design rationale (DR) is not captured.

The expertise and knowledge of each ontology development team member, the activities he/she executed, and the decisions made during the development process might be of great importance in future projects. In addition, provided the design of an ontology is an incremental and iterative process, further issues need to be considered. In an iterative evolution, design decisions, constraints and assumptions made at a given iteration step must be taken into account in following iterations. However, current tools supporting ontology development processes do not capture this type of information. In consequence, the process trace is lost and any new ontology development process would start from scratch.

Design rationale is an explanation of why an artifact is designed the way it is (Lee, 1997). In the ontology domain, DR encompasses background information on the ontology development process, including the justifications of design decisions, records of the design alternatives that have been considered and tradeoffs that have been evaluated. DR information is very important for the reuse of development processes, as well as for the assessment of these processes. In general, current ontology development methodologies do not adequately document the DR. Therefore, most of this knowledge remains hidden in the experts´ mind.

Research on DR capture, representation, and application has had an extensive tradition in several engineering design domains. However, few efforts have been made in the ontology engineering field and they have only focused on the representation of argumentations during design discussions. On the other hand, proposals coming from different engineering fields still have some drawbacks, which are pointed out by Zhang et al. (2013). An important one is the fact that the DR representation lacks integration with the artifacts produced during the design process. Furthermore, most representations do not provide the ability to reason and intelligently retrieve knowledge. Additionally, there are two issues that still need to be addressed: (i) how to capture DR information during design processes without affecting the designers´ activities, (ii) how to reuse the rationale that is captured.

In order to tackle the challenges presented above, this contribution proposes ONTOTracED, a framework aimed at representing, capturing and tracing those ontology development processes that start from scratch. ONTOTracED takes into account the particular concepts of the ontological engineering domain and the possible operations that can be applied on the instances of these concepts. Its goal is to capture (i) the requirements that are considered, (ii) the activities and actors that generate each design product, and (iii) the rationale behind each adopted decision, thus supporting ontology development and future reuse. Furthermore, it also offers an explicit mechanism to manage the different ontology versions that are created along a given development process.

The remainder of this article is organized as follows. Section 2 presents a description of the ontology development process, the steps and the methodologies that support the complete process. Section 3 introduces the OntoTracED framework, its components and fundamental characteristics, while Section 4 presents a computational environment, named TracED(aaS), which implements the underlying models. The application of the framework to a case study is addressed in Section 5. Finally, Section 6 presents the conclusions and new lines of research that are derived from this work.

2. Ontology development process

Ontology development processes have improved from the early phases of non-systematic work to the current activities performed in the framework of Ontological Engineering. This is a relatively new field of study concerning ontology development processes, ontology life cycles, methods and methodologies for building ontologies, as well as tool suites and languages that support them (Gómez-Pérez et al., 2004). A series of methods and methodologies for developing ontologies has been reported in literature in the last two decades, of which an extensive state-of-the-art overview can be found in Gómez-Pérez et al. (2004). In addition, Cristani and Cuel (2005) have proposed a framework to compare ontology engineering methodologies and have evaluated the established ones accordingly.

The first contributions in the field, due to Gruber (1993), Grüninger and Fox (1995), Uschold et al. (1998) and Uschold and Gruninger (1996), introduced the grounds for many subsequent proposals. Gruber's work discussed some basic ontology design criteria related to ontology quality, as well as to the development methodology. Gruninger and Fox introduced a development methodology based on Competency Questions. Methontology (Fernández-López et al., 1999), which is an ontology development process, introduced an ontology lifecycle based on evolving prototypes and specific techniques to address each activity in the methodology. Yure et al. (2009) proposed

On-To-Knowledge, which emphasizes knowledge management. Other approaches, often related to industry or research projects, include the methods used for building CyC, SENSUS (Swartout et al., 1997) and KACTUS (Bernaras et al., 1996). These methodologies mainly include guidelines for the construction of single ontologies, going from their requirements specification up to their implementations. On the other hand, the Neon methodology (Suárez-Figueroa et al., 2015) suggests pathways and activities for a variety of scenarios including, among others, the development of ontologies from scratch, the reuse and merge of ontological and non-ontological resources, as well as the reuse of ontological design patterns.

Although many ontology development methodologies have been proposed in recent years, no one is yet emerging as a clear reference (De Nicola et al., 2009). In this proposal, the methodology introduced by Uschold et al. (1998) has been adopted, since its phases – specification, conceptualization and implementation – are included in all the other methodologies. In fact, the first and second phases are present in all the methodologies with the same or different names. On the other hand, implementation may be a complete phase in itself in certain methodologies, it may be executed together with the conceptualization step in others, or it can be totally absent. The scope of each of these steps, schematically shown in Fig. 1, is briefly explained in the following paragraphs..

The first phase, named **requirements specification**, identifies the scope, purpose and intended use of the ontology. Additionally, in some methodologies, this phase also refers to a first identification of concepts and relationships existing in the domain, as well as the possible ontologies to be reused. Some methodologies (Grüninger and Fox, 1995, Sure et al., 2009, Suárez-Figueroa et al., 2015, De Nicola et al., 2009) ground this phase on the definition of competency questions that help identifying requirements and the scope of the ontology. Thus, from each competency question the ontology developer extracts the proper terminology. A first pre-glossary of terms is pulled-out from competency questions and their answers.

The **conceptualization** phase is related to the capture of concepts, relations and their semantics from the domain in order to build the ontology from them. This stage organizes and converts an informally perceived view of a domain into a specification using a set of intermediate representations that can be understood by domain experts and ontology developers. Assumptions and constraints in the interpretation of such terminology are also defined in an informal way. There are different approaches to represent concepts and relations at this phase: i) natural language (Debruyne and De Leenheer, 2014), ii) intermediate representations such as tables (Gómez-Pérez et al., 2004; Yure et al., 2009) or UML diagrams (De Nicola et al., 2009), and iii) predicates in first order logic (Grüninger and Fox, 1995). The first and third approaches imply the absence of an implementation phase and the addition of implementation-related activities to the conceptualization one. In general, when a methodology considers an intermediate representation language, a formalization step is required to obtain a formal ontology.

Ontologies can be represented with different knowledge modeling techniques and can be implemented in various kinds of languages. However, not all of them can represent the same knowledge, with the same degree of formality and granularity. Gómez- Pérez et al. (2004) have stated that AI-based approaches combining frames with first order logics or description logics are suitable for modeling heavyweight ontologies, while software engineering approaches and databases allow the representation of lightweight ontologies. In spite of their diversity, techniques share structural similarities and have common modeling elements, such as:

- *Classes*: They represent a collection of entities that share a common set of characteristics. Certain languages call them concepts or frames. *Classes* can be hierarchically organized by means of subsumption relationships.
- *Relations*: They are associations between classes. Different languages call them properties, slots, roles, or associations.
- *Individuals:* They are entities that belong to a particular class. They are also called instances or members of this class.

The activities at this conceptualization phase require close cooperation between domain experts and ontologists, because both of them need knowledge about the domain and important ontological distinction patterns (Neuhaus et al., 2013). In order to facilitate this task, several authors proposed the use of Ontology Design Patterns (ODPs), the ontological engineering version of software engineering design patterns. ODPs identify and specify abstractions that are above the level of single concepts and relations. These abstractions address a recurring design problem that arises in specific design situations and provide a solution for it (Buschmann et al., 1996). Ontology design patterns in their current interpretation were introduced by Gangemi (2005) and Blomqvist and Sandkuhl (2005). Blomqvist defines the term as "a set of ontological elements, structures or construction principles that solve a clearly defined particular modeling problem" (Blomqvist, 2009). ODPs can be grouped into six types, each addressing different modeling problems. Examples of these patterns are n-Ary Relations, Value Partition (or just Partition), Normalization, Negative Property Assertion, among others. A list of several proposed patterns can be found in the ODPs public catalog (ODPsOrg, 2015; ODPS, 2015).

Once a conceptual model is built, it is necessary to transform it into a formal one, which could be represented in an ontology implementation language. Such transformation is the objective of the **implementation phase**. It is important to notice that the same conceptual model might result in several implementation models (Hartmann et al., 2006), because distinct applications might require different kinds of reasoning services, and special-purpose languages to support them. Languages to specify ontologies can be classified into two groups: (i) AI-based languages, such as KIF (NCITS, 1998) and Flogic (Kifer et al., 1995), and (ii) Web-based languages. The knowledge representation paradigm underlying the first group is based on first order logics, frames combined with first order logics, and description logics. Languages belonging to the second group, such as RDF (Gandon and Schreiber, 2014), RDF- Schema (Brickley and Guha, 2014) and OWL (W3C-OWLwg, 2012), have been developed by the W3C (World Wide Web Consortium) and they have established the foundations for the Semantic Web (Shadbolt et al., 2006).

Regardless of the adopted approach, the development of ontologies is a challenging process that requires supporting tools. In the last decade, such a process has been transformed from a one which has been traditionally performed by isolated engineers or domain experts into a process that has been collaboratively executed by mixed teams (Palma et al., 2011). Creating and designing an ontology requires the collaboration of domain and ontology engineering experts. In order to arrive at a consensual model of a domain, expressed by means of an ontology, the participants of the engineering project must discuss their different viewpoints efficiently (Gangemi et al., 2007). So, discussions are an important part of collaborative ontology engineering activities and capturing them would reflect the design rationale behind the

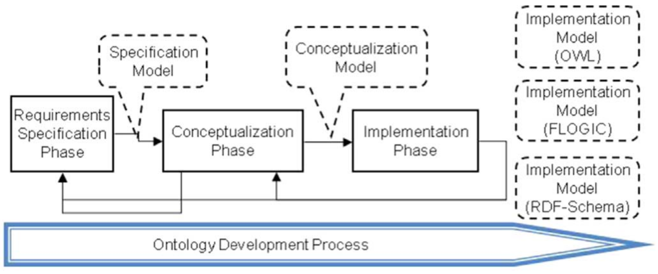

Specification Model
Conceptualization Model
Implementation Model (OWL)
Implementation Model (FLOGIC)
Implementation Model (RDF-Schema)
Requirements Specification Phase
Conceptualization Phase
Implementation Phase
Ontology Development Process

Fig. 1. Ontology development phases.

ontology. The captured design rationale might be useful for understanding the decisions made in previous activities when developing an ontology from scratch, but it may also be important in other scenarios, such as the reuse or the reengineering of ontological and non-ontological resources, or the alignment with other ontologies.

The exchange of arguments and the capture of the design rationale behind design decisions constitute a major part in collaborative ontology building. The argumentation process is used in several domains, like medicine (Doumbouya et al., 2015), social agents (Heras et al., 2014), engineering design (Baroni et al., 2015; Gonnet et al., 2007b), and ontology design (Hauagge Dall Agnol and Tacla, 2013), among others. In ontological engineering some proposals capture the argumentation process using IBIS (Kunz and Rittel, 1970), which is the most accepted model of argumentation. Among others, Human-Centered Ontology Engineering Methodology (HCOME) (Konstantinos and Vouros, 2006), DIstributed Loosely-controlled and evolvInG Engineering process of oNTologies (DILIGENT) (Pinto et al., 2009) and Cicero (Dellschaft et al., 2008), are examples of proposals that use IBIS model as part of the ontology development process.

HCOME and its environment HCONE (Human-Centered Ontology Engineering Environment) support the development of ontologies in a collaborative decentralized way. Each ontology works (creates or merges ontologies) in a private space. Private ontologies can be put in a shared space, which is accessed by all involved ontologists, in order to discuss ontological decisions. The discussion is captured using the IBIS argumentation model. Once an agreement is reached, the ontology is moved into an agreed space.

The DILIGENT methodology proposes an ontology that deals with argumentation in collaborative workflows. This proposal incorporates certain elements to the IBIS model in order to accelerate issue resolution. This argumentation model, along with the one proposed by Potts and Bruns (1988), constitutes the basis of the argumentation model that underlies the Cicero tool. This tool is an extension of a semantic mediawiki (Krötzsch et al., 2006) that combines the general structure for representing discussions from the DILIGENT argumentation framework with the idea of annotating ontology elements and changes with the corresponding discussions. The Cicero tool has been developed within the scope of the NeOn project and it is integrated with the NeOn toolkit ontology editor. Therefore, when a user working with the editor adds a concept or a property into an ontology, he/she can link this new concept or property with an element of the argumentation framework. Although the Cicero tool gives support to capturing the discussion during the ontology development, the discussion and the generated ontology model run on different paths.

C-ODO is another tool developed as part of the NeOn project. It consists of a framework formalized in OWL, which can be used as a requirement language for describing social level aspects of ontology design (Gangemi et al., 2007). The framework defines six layers, including one for argumentation and another one for ontology design rationale.

All the aforementioned proposals deal with the capture of arguments supporting discussions among ontology developers. Additionally, they provide support for capturing the decisions that ontologists have made during the ontology development process. Some of them, such as HCOME and DILIGENT, are independent from methodologies or editor tools. Other ones were developed within the scope of the development project of other tools, or were intended to be integrated in an existent environment. However, none of them focuses on the capture of fine-grained operations, such as the addition of competency questions, addition/removal of concepts, relations or individuals, and modification of properties of these ontological elements. Moreover, none of them provides a mechanism to capture and trace how informal competency questions are formalized or which group of concepts, individuals and relations are used to answer one of these questions.

Although there are some initial proposals to capture argumentation, in general, once a specific ontology development process concludes, the things that remain are isolated design products (e.g., the requirements specification, competency questions, ontology class diagrams, the ontology implementation in a specific language, etc.). However, these products are preserved without an explicit representation of how they were obtained; thus, the history and rationale behind the development process have not been captured. In fact, there is no trace of the activities that originated the products, the requirements that were imposed, the actors performing each activity, or the rationale behind each decision.

In order to overcome these drawbacks, this article introduces an operational-oriented approach to represent and capture the ontology development process. Some preliminary results of the proposal have been presented in Vegetti et al. (2012), which considers an ontology development process that is based on the ontological categories defined by the Unified Foundational Ontology (UFO) (Guizzardi, 2005). In this contribution we extend and generalize these preliminary results, proposing a comprehensive framework based on the three-phase methodology depicted in Fig. 1. The main differences between this proposal and the one of Vegetti et al. (2012) are: i) the specification of a new Ontology Engineering Domain Model (OEDM), which is based on a general methodology, ii) the capability of extending the OEDM to fit any particular ontology development methodology, iii) a new architecture for the computational environment that implements the framework, which was developed using cloud computing technologies. In addition, new and more complex case studies, pertaining to the engineering domain, are tackled in this contribution.

3. A Framework to capture and trace the ontology development process

In order to override the limitations that were mentioned in Section 2, this contribution proposes a comprehensive framework to represent, capture and trace ontology development processes.

Fig. 2 shows the main components of the ONTOTracED framework that includes: (i) a *Conceptual Model,* which is able to represent generic design processes; (ii) an *Ontological Engineering Domain Model (OEDM),* which specifies the concepts that are required to describe ontology development processes, and (iii) a computational support environment, named *TracED(aaS) (Trace Engineering Design as a Service)*, that implements both the conceptual model and the OEDM to enable the capture of specific ontology design processes, along with their associated products..

The supporting *Conceptual Model* is based on an operational-oriented approach that envisions the ontology development project as a

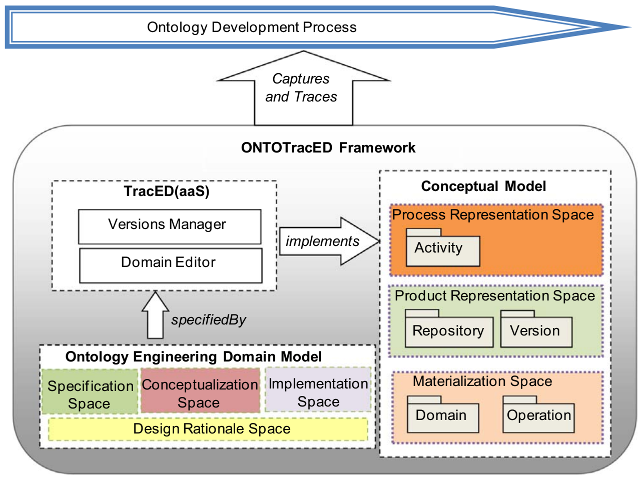


Fig. 2. Components of the proposed framework.

sequence of activities that operates on the products of the development process. The proposal defines two representation spaces to model generic design process concepts: the *Process* and *Product* spaces. In addition, a third component (the *Materialization Space* in Fig. 2) is included to fully specify a flexible model that is able to represent and capture design processes pertaining to specific engineering fields.

The *Ontological Engineering Domain Model* (OEDM) component can represent and capture specific ontology development projects, based on building blocks that define the products obtained, as well as the activities carried out during this type of processes. This representation includes the modeling elements most commonly found in current methodologies for guiding ontology development processes. Examples of these modeling elements are the competency questions, concepts, and relations. The OEDM component is organized in four different representation spaces. Three of them specify building blocks and activities related to the specification, conceptualization, and implementation phases of the ontology development process. The other one allows representing design rationale objects and operations.

TracED(aaS) is the computational environment that implements the conceptual model and enables to define the OEDM and incorporate it to the framework. *TracED(aaS)* is based on TracED (Roldán et al., 2010), which was conceived for capturing and tracing engineering designs. The major components of *TracED(aaS)* are the *Domain Editor* and *Versions Manager*. By using the *Domain Editor*, the OEDM has been specified in *TracED(aaS)*. Furthermore, the editor allows this model to be further specialized, if required. On the other hand, the *Versions Manager* keeps track of the execution of a design project, as it will be shown in the following sections.

3.1. Conceptual Model

The *Conceptual Model* component provides the framework foundations. This component is organized in *Process Representation*, *Product Representation* and *Materialization Spaces*, which are explained in this section. The *Process Representation* space models the activities being performed during an ontology development process and it is specified by the *Activity* package (Fig. 3). In particular, when tackling the development of an ontology, typical tasks are: the incorporation of concepts and relations into the ontology, the definition of constraints on a specific concept, the analysis of whether a group of concepts, relationships and constraints satisfies a formal competency question, the evaluation of the ontology, decisions on alternative concepts and relations, etc. As Fig. 3 shows, such activities are represented in the model with the *BasicActivity* or *CompositeActivity* classes, depending on whether the task is atomic or it can be decomposed into a set of subactivities..

In the proposed model, the execution of an activity is guided by one or more *requirements,* which specify the functional and non-functional characteristics that a development product must satisfy (e.g., in the ontology development domain, the concepts have to preserve the ontological commitment of the domain being modeled). Therefore, the ontology development process is interpreted as a series of activities

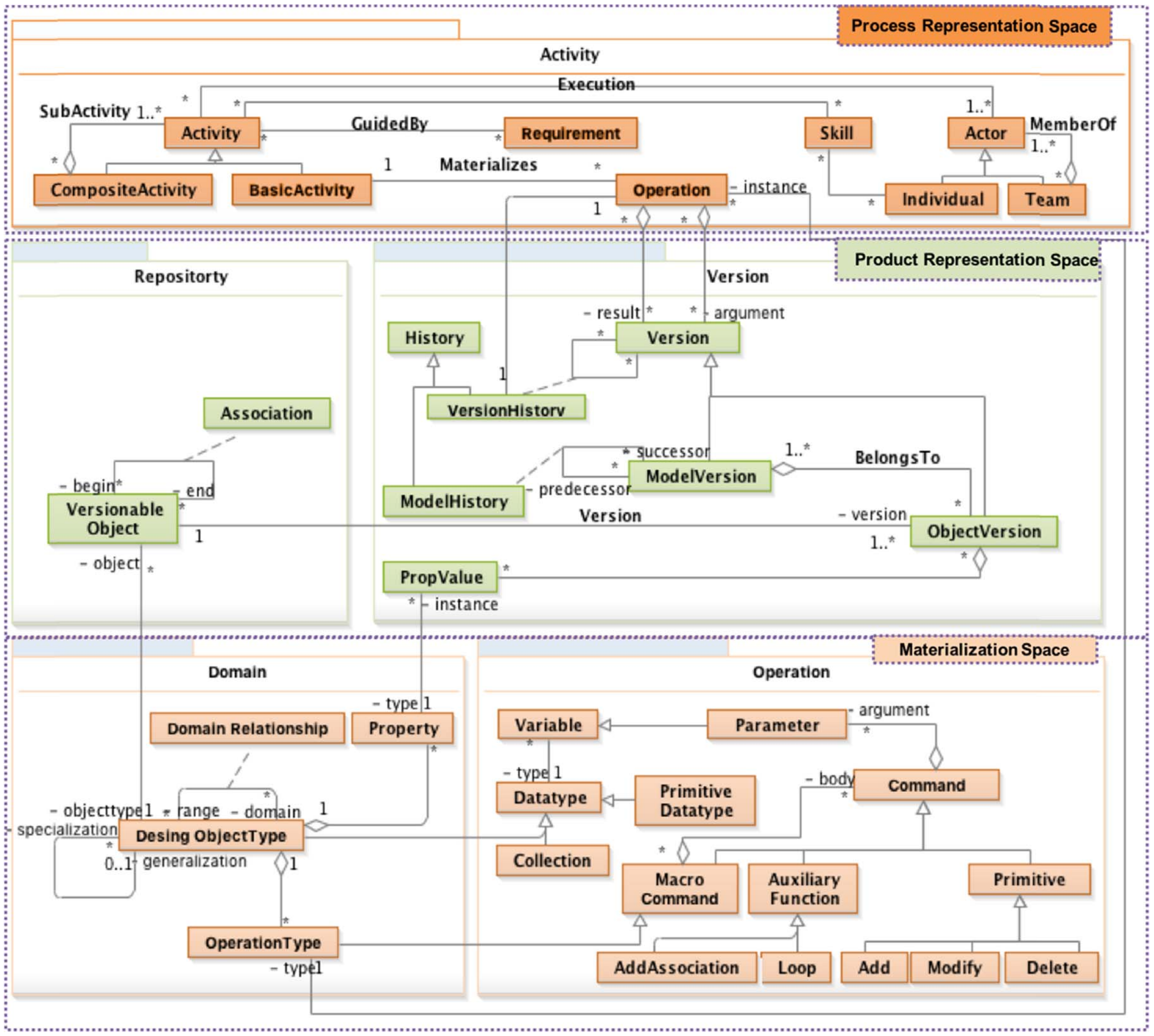


Fig. 3. Conceptual Model: process representation, product representation and materialization spaces.

led by *requirements* that are performed by *Actors*. An *Actor* may be either an *Individual* (a human being or a computational program) or a *Team*. Teams allow representing composite skills that are needed for carrying out activities. Each basic activity performed by an actor during an ontology development process is represented by the execution of a sequence of *operations*, which transforms the design objects. The operations that can be applied are domain dependent. So, it is necessary to define the allowed types of operations, as well as the modeling elements, for each specific domain.

In this proposal, the execution of an activity (materialized through a sequence of operations) transforms a design object, which thus may evolve into multiple versions. In order to represent this evolution, each *design object* is specified at two levels: the *Repository* and the *Version* packages (Fig. 3), which constitute the *Product Representation Space*. The *Repository* keeps a unique entity for each *design object* that has been created and/or modified due to the natural progress that takes place during a development project. Any entity kept in the repository is regarded as a *versionable object*. Furthermore, relationships among the different versionable objects are also maintained in the repository (*Association* class in Fig. 3). On the other hand, the *Version* level keeps the different versions resulting from the evolution of each design object, which are called *object versions*. The relationship between a *versionable object* and any of its *object versions* is captured by the *Version* association. Therefore, for a given *design object*, a unique instance is kept in the repository, and all the versions it assumes along the design process belong to the version level. Fig. 3 also includes the *Design ObjectType* class, which allows representing the various kinds of modeling elements pertaining to particular domains.

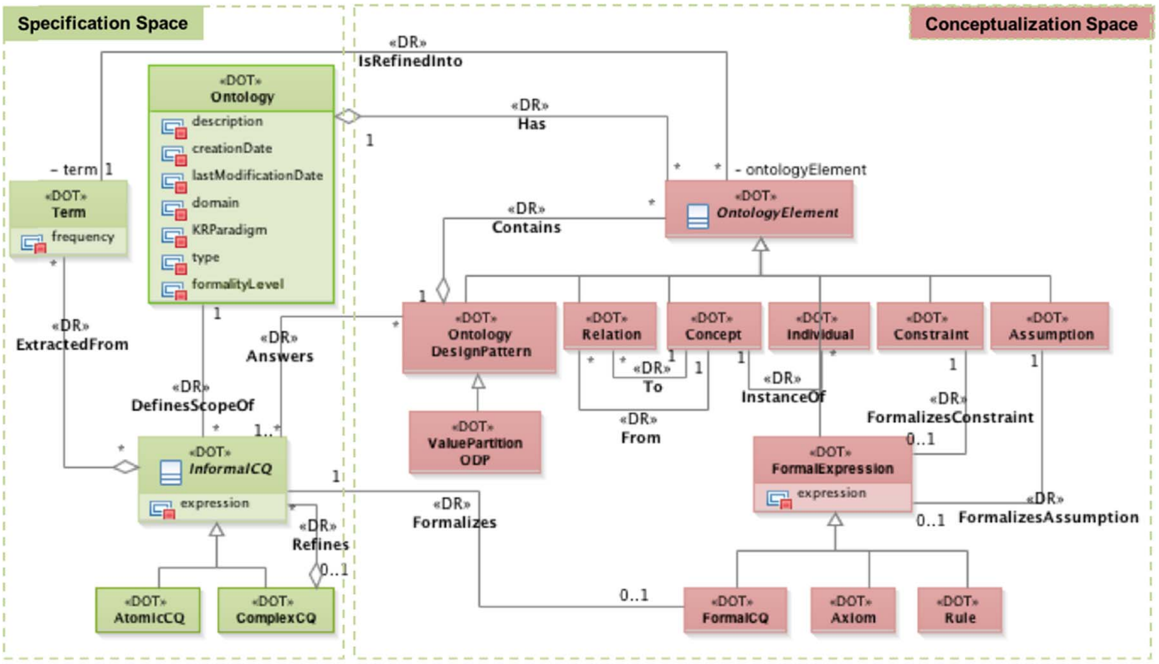
The version package also includes the *ModelVersion* concept, which represents a set of design objects within the context in which a given design activity is carried out. Its aim is to provide a snapshot description of the state of a certain design process at a given moment. According to the proposed representation, a new *model version* m_n is generated when a *basic activity* is executed. Since each *basic activity* is *materialized* by a sequence of operations, named ϕ, the new *model version* m_n is the result of applying such sequence to the components of the previous *model version* m_p. This *predecessor model version* m_p corresponds to the context where the activity was performed and the successor one (m_n) represents the resulting context. In order to represent engineering design evolution, a model version has zero or more successor model versions (noted by the * cardinality at successor role of *ModelHistory* association shown in Fig. 3).

Each transformation operation applied to a model version incorporates the necessary information to trace the model evolution. This information is represented by *VersionHistory* relationships between the object versions to which the operation is applied (arguments) and the ones arising as the result of its execution (results, Fig. 3).

The *Materialization Space* is defined by the *Domain* and *Operation* packages (see Fig. 3), which permit ontological engineers to specify the building blocks and operations of particular engineering design domains. In the context of the OntoTracED framework, this space has allowed defining the ontological engineering domain model. In addition, this space can be employed to specify different engineering design domain models like chemical engineering (Gonnet et al., 2007b) and software engineering (Roldán et al., 2010), among others.

The *Design objects types* (Fig. 3) represent the various products of the development activities. Typical design object types are models of the artifact being conceived (e.g., in the ontology development domain: class diagrams, implementations in specific ontology languages, etc.) and specifications to be met (i.e., competency questions, quality attributes, etc.). Design object types may relate among themselves by specific domain relationships (*DomainRelationship* association class in Fig. 3), and they can be organized in generalization-specialization hierarchies. Design object types are described by a set of properties. Moreover, each design object type is related to a set of operation types that may be used to transform such design object.

The *Operation* package enables the specification of operation types and their implementations in a specific computational environment (*TracED(aaS)* in this case). This package defines the primitive operations *add*, *delete* and *modify* and enables the specification of other operations that are applicable into the specific design domains (the ontology development domain in this work). When an *operation* is specified, it is necessary to define both its *parameters* and its *body*. The *body* comprises some already defined commands that are available for being used in other operation specifications. They can be primitive (such as *add*, *delete*, or *modify*), *auxiliary function* commands, or previously defined operations.



DOT: Design Object Type - DR: Domain Relationship

Fig. 4. Partial view of OEDM$_{ah}$: specification and conceptualization spaces.

The proposed conceptual model supports registering the design decisions as they are made (by means of capturing the executed operations), along with their impact on the ontology model in terms of changes (the results of the operations). This is a fundamental step towards the development of a knowledge base that guides the ontology developer in the different activities of an ontology development process, setting the grounds for learning, future reuse, and providing means for detecting potential conflicts. This conceptual model is based on the preliminary work of Gonnet et al. (2007b), which presents a thorough formalization based on situational calculus and first order logic. Due to limitations of space no further details on this issue are included in this contribution.

3.2. Ontological engineering domain model

As it was mentioned in Section 3.1, the *Domain* and *Operation* packages (Materialization space) of the underlying conceptual model let specify modeling elements and operations that are suitable for particular domains. This section presents the use of these packages in the specification of the *Ontological Engineering Domain Model*, which is a description of the constructive elements, artifacts and operations that arise during the execution of ontology design processes regarding a specific ontology development methodology. It can be defined as follows:

$OEDM_i = \{D_i,\ R_i,\ O_i, DR_i\}$

where D_i is the set of design object types used by the ontology development team during the design of an ontology, regarding a specific methodology i. R_i is the set of relationship types that enables to link elements in D_i and O_i is the set of operations that handle the design object types in D_i. Finally, DR_i is the set of concepts that enables the capture and tracing of the design process and its rationale.

As it has been previously introduced, there are several methodologies for building ontologies. However, there is no agreement on a methodology for ontology development (Neuhaus et al., 2013). In spite of their diversity, most methodologies share structural similarities and have comparable modeling elements, which are considered to be part of the proposed domain model.

The following paragraphs introduce $OEDM_{ah}$, which is the domain model for the ad-hoc three-phase ontology development methodology introduced in Section 2. $OEDM_{ah}$ is organized into representation spaces, one per each phase in the ad-hoc methodology that is shown in Fig. 1, and one for representing the design rationale. The first part of this section introduces the design object types (D_i) and domain relations (R_i) that belong to specification and conceptualization spaces. Then, ideas about the possible implementation space are presented. After that, design object types for representing design rationale (DR_i) are introduced. Finally, the last part of this section explains how the operations (O_i) are defined in the $OEDM_{ah}$ spaces.

Specification space's design object types

The left part of Fig. 4 presents the main building blocks of the $OEDM_{ah}$ specification space. In this figure, the *DOT* and *DR* stereotypes are used to indicate that the elements are instances of the *DesignObjetcType (DOT)* or *DomainRelationship (DR)* classes defined in the Materialization space, respectively (Fig. 3)..

The *Ontology* design object type represents the ontology artifact that is under design. Some attributes of this design object type are creation date, last modification date, the domain described by the ontology, and its type and formality level. This metadata is only an example of the capabilities of the model for representing information about an ontology. This metadata can be extended by taking concepts from some existent vocabularies, such as the Dublin Core (DC) metadata standard (ISO, 2009), the Ontology Metadata Vocabulary (OMV) (Hartmann et al., 2005), among others.

A given ontology is built to address a set of requirements; in fact, it is evaluated against its corresponding requirements specification. Requirements can be defined through appropriate competency questions, which define the scope of the ontology (*DefinesScopeOf* relationship in Fig. 4). Therefore, the $OEDM_{ah}$ includes a set of design object types relative to competency questions. The *InformalCQ* design object type represents the competency questions that are written in an informal way, using natural language, and that are defined by ontologists during the specification phase. Informal competency questions can be specialized into atomic ones (*AtomicCQ* in Fig. 4), which represent simple competency questions, and complex ones (*ComplexCQ* in Fig. 4), which can be expressed in terms of simpler ones.

Competency questions participate in most of the ontology design methodologies and they are the starting point in the identification of the ontology terminology. To represent the terms to include in the pre-glossary derived from competency questions, the *Term* design object type is defined in the OEDM domain. The definition of the $OEDM_{ah}$ should also specify possible relationships (R_{ah}) among design object types.

Conceptualization space's design object types

During the conceptualization phase, ontology developers build and formalize the ontology. Two main design object types are proposed in the conceptualization space: the *OntologyElement* and *FormalExpression*, which allow ontology developers to represent the ontology building blocks and the elements for formalizing ontologies, respectively. Both object types are then specialized into more specific types.

The *OntologyElement* object type is an abstract concept that represents the different kinds of entities that define the terminology. This abstract design object type is specialized (see Fig. 4) to represent simple modeling entities, such as *Concept*, *Individual* and *Relation,* as well as more complex ones such as *OntologyDesignPattern*. In addition, the proposal includes the *Assumption* and *Constraint* object types, which represent natural language expressions that impose restrictions on concepts and relations.

OntologyDesignPattern is a composite design object type that is defined in the conceptualization space to represent ontology design patterns (ODPs). This object type can be specialized to represent the diverse existent ODPs. In particular, Fig. 4 shows the *ValuePartitionODP*, which describes how to model a partition; i.e., a named concept which is divided into several disjoint concepts. Other patterns can be defined according to the domain requirements.

Buschmann et al. (1996) state that a pattern is made up of a context, where the pattern is useful or valid, a problem that arises within such context, to which this pattern is connected, and finally, a proven solution for the problem. In the Ontology Engineering field, the problem is stated by means of competency questions, which define the scope and the type of questions that the pattern could answer. Therefore, each pattern is related to a set of competency questions by means of the *Answers* association in Fig. 4. Since ODPs are general reusable solutions to commonly occurring problems within a given context, they constitute a tool for documenting well-proven and reusable design experience. Therefore, it is interesting to codify these patterns by means of a set of design object types and complex domain operations that facilitate their application in ontology design processes, in order to enable experience reuse.

An ontology can be a simple domain taxonomy or it can model the domain in a deeper and formal way, thus imposing some restrictions on the domain semantics. To represent this kind of formalization, special design objects and operations should be incorporated into the $OEDM_{ah}$. In particular, *FormalExpression* is an abstract concept that generalizes the different kinds of formalizations that can be applied to constraints, assumptions and informal competency questions.

FormalExpression is specialized into *Axiom*, *Rule* and *FormalCQ* design object types. *Axiom* and *Rule* represent formal expressions that allow ontologists to: i) explicitly define the semantics of an ontological concept by imposing constraints on its value and/or its interactions with other concepts; ii) verify the consistency of the knowledge represented in the ontology, and/or iii) infer new knowledge from the explicitly stated facts. The *Formal Competency Question* (*FormalCQ*) object type represents a specification in a formal language of an informal competency question that was initially identified.

Implementation space

As it was previously mentioned, the model generated in the conceptualization phase can be used to produce several alternative implementations in different implementation languages. Non-functional requirements such as decidability, completeness, computational complexity, reasoning paradigms (open vs closed world), among others, require a translation from the conceptual model of the ontology to an implementation model. Such translation may produce a loss of some domain knowledge (Gómez-Pérez et al., 2004).

The modeling elements used to describe the implementation model of an ontology depend on the language chosen for its codification. The Ontology Definition Metamodel (ODM) (OMG, 2014) can be used to define the building blocks for the implementation models, if OWL is selected as the implementation language. However, the extension of object types to capture the design of implementation models is out of the scope of this work.

Design rationale space's object types

The model introduced up to here is significant to capture ontology design knowledge. The definition of a domain in terms of the design objects and the operations that manipulate them makes explicit the activities that can be executed during ontology development processes. In addition, the “execution” of domain operations materializes the design decisions of the ontologist at each point in the ontology development process. All the decisions made during a design project are captured by the model, which keeps the performed operations, their effective arguments, and the resulting products, thus saving a great portion of the applied development knowledge. In this way, both sources of knowledge contribute partially to the explanation and rationale of the ontology design process. However, to enrich and extend the knowledge about the development process, it is necessary to incorporate explicit design rationale elements into the domain model. In this section, taking advantage of the flexibility of the proposal, we enlarge the OEDM with a new set of design object types and their applicable operations, which are aimed at capturing design rationale. Fig. 5 presents the specific design object types that were added to the OEDM$_{ah}$. They are associated with the design object types that were already introduced in Fig. 4..

The proposal incorporates three concepts from the IBIS methodology (Kunz and Rittel, 1970): *Issue*, *Position* and *Argument*. According to this methodology, an issue represents a problem to be solved. A position is an answer to an issue that helps solving the issue. Each issue may have many positions. An argument (*Argument* in Fig. 5) is a statement that supports or opposes a particular position. Every argument points to at least one position and every position points to at least one issue.

Within the scope of this proposal, competency questions are issues; thus, they are the problems that the ontology should answer. A *Position* represents a set of ontology elements (*Concepts*, *Relations*, *Individuals, Design patterns,* etc.) that contributes to answer a competency question. Each *Position* may be supported by (*Supports* relationship in Fig. 5) or objected by (*ObjectsTo* association in Fig. 5) one or more arguments.

During the development process, the ontology developers have to decide which position is selected and which positions are rejected, as answers to a competency question, based on the stated arguments. To represent these decisions, the *Decision* class is linked to the supporting argument and related to the selected or rejected position through *Selects* or *Rejects* associations, respectively.

Operations definition

A domain model not only defines the type of products (design objects) whose versions should be kept in a design project, but also the design operations to manipulate them. Domain operations are intended to transform the design products of the OEDM (generating new versions, and maybe, eliminating other versions). When a sequence of operations is applied to a given model version, it materializes a design decision. Since the *add*, *delete* and *modify* primitives are not enough to adequately represent high-level design decisions, it is necessary to provide the model with capabilities for defining and executing complex operations. Operations are defined by instantiating the operation model described in Section 3.1.

The operation model is flexible enough for defining operations with different abstraction levels. Therefore, they can be as basic as *addConcept*, *deleteConcept*, and *addCompetencyQuestion*, or as complex as the operations that apply an ontology design pattern. As an example of the operations that are defined in OEDM$_{ah}$, the functional specifications of some of them are presented in Appendix A. Functional specifications provide an outline of how operations would be defined, using a computational tool that implements the proposed model.

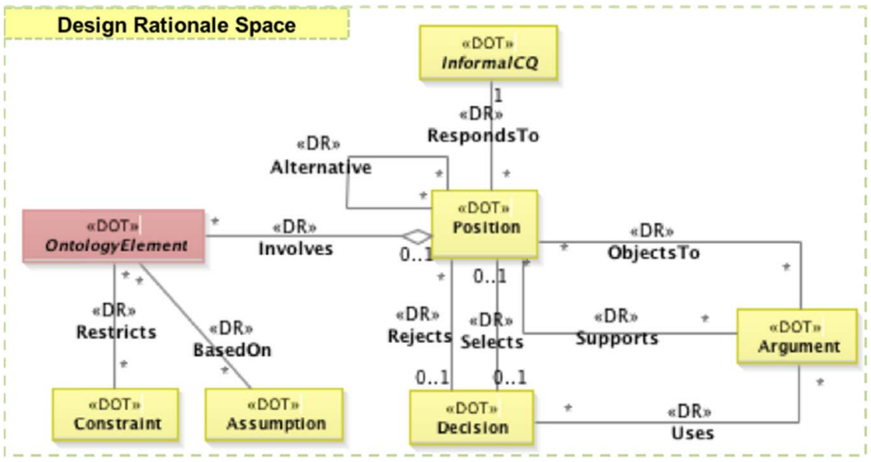


Fig. 5. Partial view of OEDMah: design rationale space.

4. TracED(aaS)

TracED(aaS) (Trace Engineering Design as a Service) is the cloud-based computational environment that implements the conceptual model and enables the definition of the OEDM$_{ah}$ domain model, thus materializing the ONTOTracED framework. As its name indicates, it provides services for tracing engineering design projects. The major components of TracED(aaS) are the *Domain Editor* and *Versions Manager*. By making use of the *Domain Editor*, a domain expert can specify different engineering Ontological Engineering Domain Models. The *Versions Manager* enables the execution of each ontology development project, and captures its evolution based on the *operations* that are accomplished and the instantiation of the *design object types* that have been specified in the Ontological Engineering Domain Model.

Fig. 6 shows the main interface of TracED(aaS) and its main panes. At the top of the interface the option menu is located. Next to it, the title bar, which indicates the model that is being edited in the edition pane, is placed. In this pane, the OEDM$_{ah}$ domain model is presented (see Fig. 6). On the left, it is possible to see the project tree. Each project involves the definition of at least one domain model, using the *Domain Editor*, and one or more design models, which are managed by the *Versions Manager* component. In particular, in Fig. 6, the project manager shows two projects: the private *ontotraced* project and a public one. The edition pane presents some of the design object types introduced in Fig. 4 and Fig. 5, along with their properties and operations. Properties are defined and edited by working with the properties tab of the design object type specification window, where the designer can indicate whether a design object type is abstract or concrete, assign a concept description and create or modify properties (instances of the Property class, Fig. 3).

Additionally, the environment allows defining domain associations as design object types. Hence, they can be reified, properties can be defined for them, and their object versions can be maintained. Specifically, designers can create binary links between design object types, and these associations are instances of *DomainRelationship* (Fig. 3). In each association, one end assumes the role of domain, and the other one, the role of range. Both association roles are qualified by multiplicity attributes. Furthermore, it is possible to define these links as composition, aggregation, or association. They have the same semantics as their respective relationships in UML (OMG, 2015).

The *Domain Editor* provides features for specifying operations having implementations that are based on the Operation Model (Fig. 3). Fig. 7 depicts the TracED(aaS) interface to define an operation. This interface has two tabs, one that lets the user define the operation arguments and another one that allows creating the body of the operation. In particular, Fig. 7 shows the definition of the *refineCQ* operation according to the functional specification that was presented in Fig. A.2 of Appendix A.

The *Version Manager* component is the core of TracED(aaS). It enables the execution of ontology design projects. When a new design project is created, an existing domain has to be selected. The *Version Manager* is the tool that the designer employs during the development of a project, by considering design object types and operations defined in the specific domain. Therefore, the evolution of a project is based on the execution of these domain operations and the instantiation of the selected design object types. Additionally, the *Version Manager* has features that allow TracED(aaS) keeping information about: (i) predecessor and successor model versions (if any exists) of each model version; (ii) history links that save traces of the applied sequences of operations, which originated new model versions; and (iii) references to the set of object versions that arose as a result of each operation execution. Furthermore, TracED(aaS) allows users reconstructing a model version by applying the whole operation sequence, from the initial to the current model version. This capability will show a view of the ontology design decisions in a chronological order, to better understand the decision-making process.

It is important to remark that TracED(aaS) was developed as a proof of concept demonstration. Therefore, this tool is not meant to replace traditional support environments. On the contrary, in the future TracED(aaS) should be integrated with existing ontology development tools. In this way, it would perform the capture of all the applied operations by working in a background mode, without being

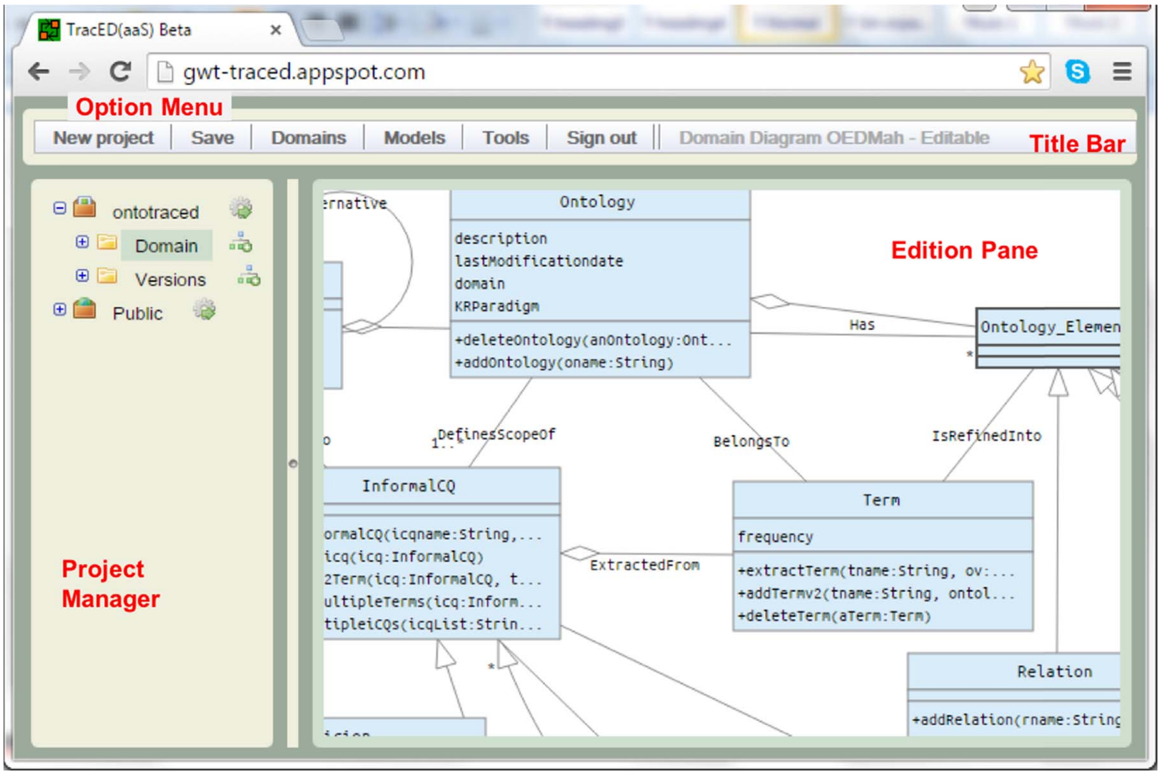
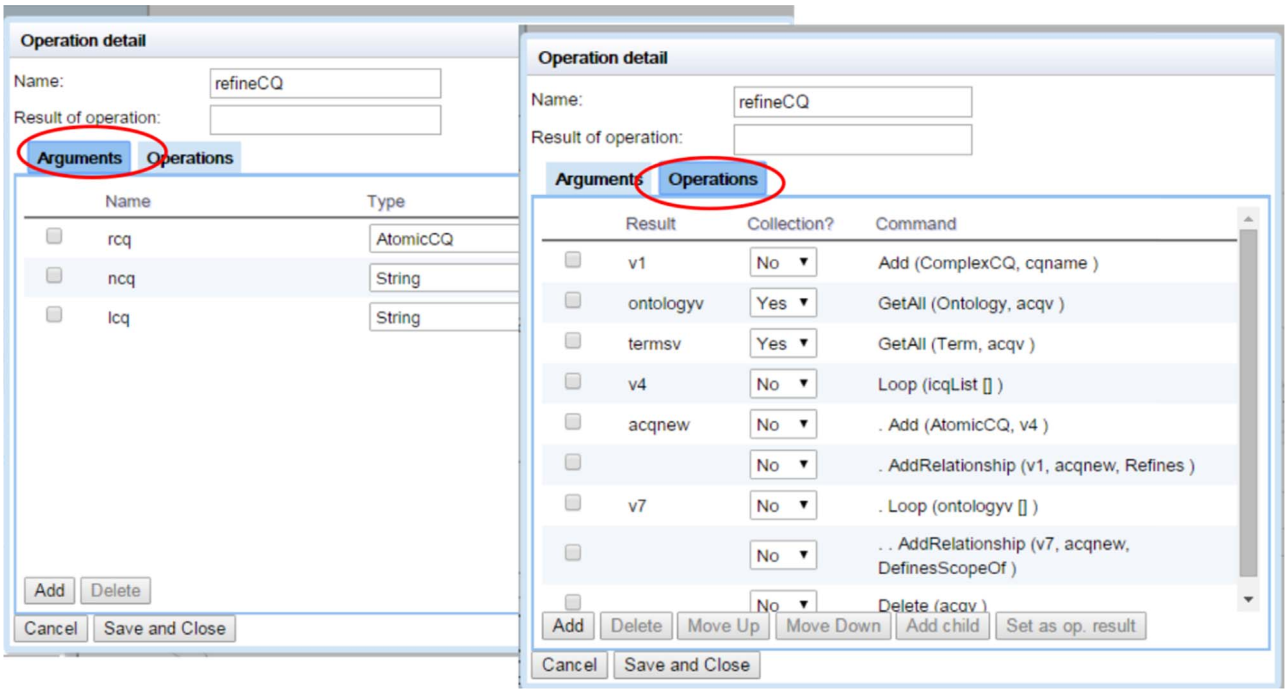


Fig. 6. TracED(aaS) main interface.



Operation detail

Name: refineCQ

Result of operation:

Arguments | Operations

	Name	Type
☐	rcq	AtomicCQ
☐	ncq	String
☐	lcq	String

Operation detail

Name: refineCQ

Result of operation:

Arguments | Operations

	Result	Collection?	Command
☐	v1	No	Add (ComplexCQ, cqname)
☐	ontologyv	Yes	GetAll (Ontology, acqv)
☐	termsv	Yes	GetAll (Term, acqv)
☐	v4	No	Loop (icqList [])
☐	acqnew	No	. Add (AtomicCQ, v4)
☐		No	. AddRelationship (v1, acqnew, Refines)
☐	v7	No	. Loop (ontologyv [])
☐		No	. . AddRelationship (v7, acqnew, DefinesScopeOf)
☐		No	Delete (acqv)

Fig. 7. Operations definition interface.

noticed by the ontology development team.

We have already considered integrating the OntoTracED framework with Stanford Protégé. Protégé is an extensible ontology editor and framework based on Java, which provides a plug-and-play environment that would make the integration possible. It has an open architecture that allows programmers to integrate i) plug-ins, which can appear as separate tabs, ii) specific user interface components (widgets), or iii) perform any other task on the current model. The *Domain Editor* and *Versions Manager* components of TracED(aaS) could be converted in separated Tab plugins in order to interoperate with the Protegé API. To make possible the integration, the following aspects are to be taken into account:

(i) When defining the OEDM domain, the set of concepts (knowledge-modeling structures) that Protégé implements (Class, Property, Individual, Constraint and Axiom), should be included as design object types. Additionally, suitable design operations are to be defined for each of them, and be associated with a specific Protégé user action (event). In this way, when a user selects an option from the tool menu, it would trigger the execution of the associated design operation, which is executed and captured in a background mode.

(ii) In order to capture any Protégé action a Listener component has to be included in the *Versions manager* plugin. This Listener could monitor User Interface (UI) actions, as well as changes in the OWL model. When any occurs, an event representing such a change will be triggered.

(iii) At the *Domain Editor* Tab, users could specify additional design object types that are not supported by Protégé(*CompetencyQuestion*, *Term*, *Decision*, *Argument* etc.), along with the design operations for managing them.

(iv) The *Versions Manager* Tab, would have a UI similar to the one provided by TracED(aaS), in which the obtained model versions are arranged in a hierarchical fashion. The UI of this Tab Window would offer a menu with the applicable domain operations, like *addTerm*, *refineCompetencyQuestion*, *term2Concept*, etc. This Tab window would present decisions, models and design reasoning knowledge. The *Versions Manager* Tab will provide a more complete view of the design ontology process by showing not only the set of classes organized in a subsumption hierarchy, but also other products generated during the ontology design process, along with the ontologists' design decisions that have originated them.

The TracED(aaS) tool was employed to develop the case study that is presented in the next section, proving to be a helpful environment.

5. Case Study. Development of an ontology to formalize and integrate the ISA-88 and ISA-95 standards

This section describes the use of TracED(aaS) in the development of a batch production planning domain ontology, a big and challenging project undertaken by our group (Vegetti and Henning, 2015) that was chosen to test TracED(aaS). Firstly, a brief description of project scope is introduced and then, some highlights of the ontology development process are presented.

ISA-88 and ISA-95 are two well-accepted standards in the industrial domain that provide a set of models considered as best engineering practices for industrial information systems in charge of manufacturing execution and business logistics. They are of outmost interest in the batch production planning domain. The main goal of ISA-88 is the specification of batch recipes and the control of batch processes, whereas the one of the ISA-95 standard is the development of an automated interface between the enterprise functions and the manufacturing execution systems. In consequence, both standards should interoperate. However, there are gaps and overlappings between their corresponding terminologies, as well as additional problems, such as semantic inconsistencies within each of the standards and the use of an informal graphical representation in one of the ISA-88 models. Therefore, there is a need for an ontology aimed at formalizing both standards, integrating them and overcoming the problems already pointed out.

In order to deal with the aforementioned semantic challenges, the formalization of the ISA-88 and ISA-95 standards was performed by defining an ontology per each one, and then, an ontology to integrate them. The proposed approach avoids defining just one big ontology, which would be very complex and rigid, by sticking to a "divide and conquer" strategy. The development of each ontology has been carried out separately, starting with ISA-88, since it is the one that has the major semantic inconsistencies in its term definitions. Due to space limitations, this section just focuses on the development of one of the ISA-88 standard modules, which is called Procedure Control module. More details can be found in Appendix B.

For the development of the Procedure Control ontology, an ad-hoc methodology based on well-accepted principles has been adopted. It is based on the four phases described in Section 2: requirements specification, Conceptualization, Implementation and Evaluation. However, the order in which these phases were carried out was not truly sequential; indeed, any ontology development is an iterative and incremental process. If needs/weaknesses are detected during the execution of a given phase, it is possible to return to any of the previous ones to make modifications and/or refinements. In particular, two iterations in the Specification phase and three iterations in the Conceptualization one have been completed. Some highlights of the methodological tasks that have been executed in the design process are given in the remaining of this section.

Throughout the development process, the *Version Manager* provided support to the handling of the different ontology versions that were reached during the design process, as well as the design decisions that were made. Working with TracED(aaS)´s *Versions Manager*, the first specification model of the ontology, called *SpecificationModel1,* was created. The competency questions and the first identified terms were added to this model version (see Table B.1 of Appendix B) by means of the application of the *addCompetencyQuestion*, *extractTerm* and *linkCqToTerm* operations, available to be used in the OEDM$_{AH}$ domain model. Fig. 8 provides a snapshot of *SpecificationModel1*, showing instances of the *Ontology*, *AtomicCQ* and *Term* design object types, as well as instances of the *ExtractedFrom* and *BelongsTo* domain relationships. In this figure, it is possible to observe that the *product* and *master recipe* terms were identified (*ExtractedFrom* associations) from competency question 1 (see Table B.1 in Appendix B), labelled as *icq1* in the figure. In addition, both terms are part of *isa88ontology* (*BelongsTo links).*.

Once the competency questions were identified, a new iteration in the specification phase was carried out. During it, the refinement of the competency questions was performed. The supporting tool captured these refinements in the execution of a sequence of *refineCQ* operations. As a result of its application, a new version of the specification model was obtained, called *SpecificationModel2*.

The second main step in the development process comprised the identification and capture of the concepts and relationships proposed to satisfy the requirements that were specified as competency questions in the previous phase. Three model versions were generated during this step in TracED(aaS)´s *Version Manager*, one per each completed iteration: *ConceptualizationModel1*, *ConceptualizationModel2* and *ConceptualizationModel3*.

The *ConceptualizationModel1* model version was obtained by applying a sequence of operations to *SpecificationModel2*, including those for refining terms into ontology elements (concepts, relations, individuals and/or ontology design patterns, among others). The ontologist detected that the most frequent term in the competency questions was "master recipe"; therefore, decided to proceed in the first iteration of the conceptualization phase, by focusing on the modeling of the master recipe. Thus, the first conceptualization model was enriched by adding new concepts, relations, individuals and ontology design patterns that were discovered during the analysis of the ISA88 standard and the industrial domains in which it is employed. Among others, the main concepts that were identified are *RecipeEntiy*, and its components: *ProceduralStructure*, *Formula* and *EquipmentRequirements* (See Fig. B.2 of Appendix B).

The second iteration of the conceptualization phase was devoted to the refinement of the *ProceduralStructure* concept. The procedural structure of a recipe depicts the procedural logic for all levels of the recipe: recipe procedure, recipe unit procedure, and recipe operation.

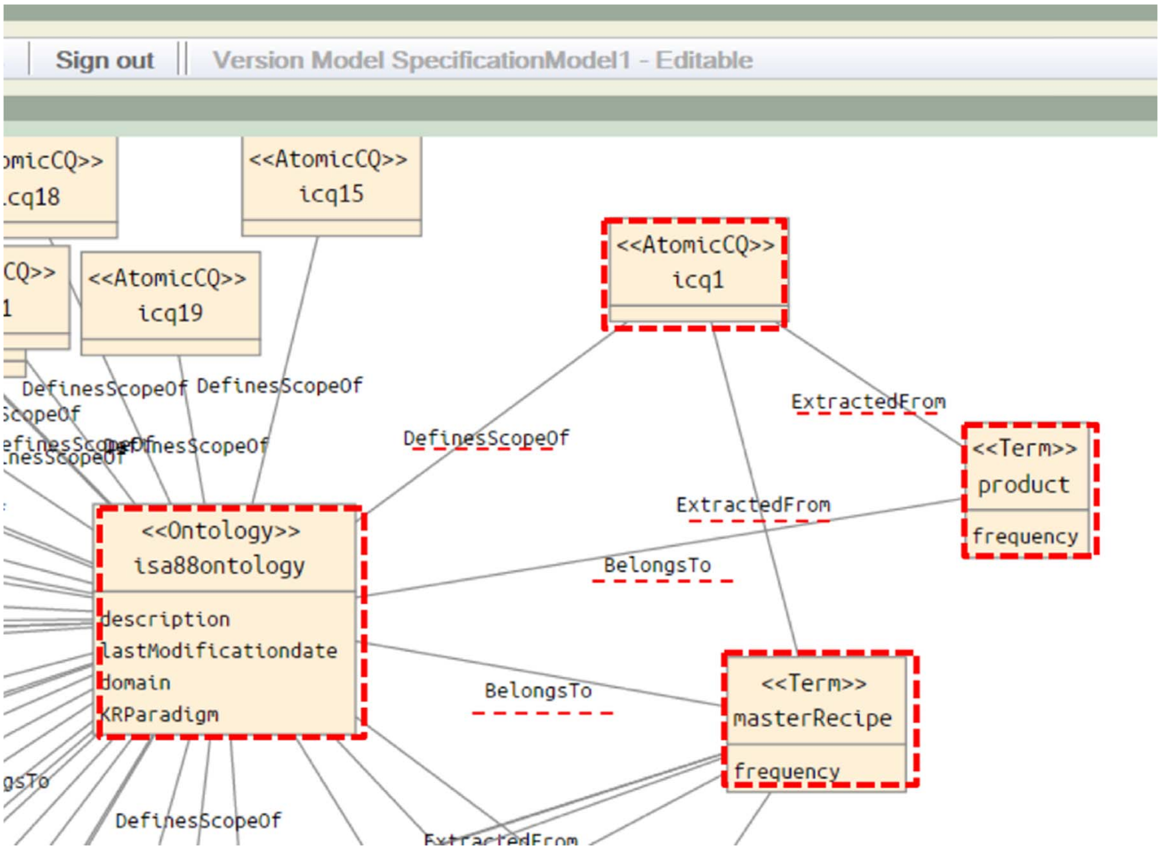


Fig. 8. A snapshot of the specificationModel1 model version in TracED(aaS).

ISA-88 suggests the use of Procedural Function Charts (PFC) to describe the procedural structure of a recipe. Therefore, the elements of the informal PFC diagrams were added into the ontology (see Fig. B.3 of Appendix B).

The *Formula* is a category of recipe information that includes process inputs, process parameters, and process outputs. The ontology engineer made a series of decisions that involved the refinement of the *Formula* concept. They were materialized in the execution of a sequence of operations that gave rise to the *ConceptualizationModel3* version, which was the result of the third iteration (see Fig. B.4 of Appendix B). Similarly, the fourth iteration focused on the decisions related to the refinement of the *EquipmentRequirement* concept.

At each iteration of the conceptualization phase, the members of the team that carried out the development process tried to answer a set of competency questions. For each of them, a position, involving a set of concepts, relations, individuals and/or design patterns, was stated to answer it. As an example, Fig. 9 illustrates the definition of a position, named *Icq2Position*, which was created to answer competency question 2 (*icq2*, see Table B.1 of Appendix B). The *Icq2Position* position, which is supported by the *RecursiveStructureofRecipe* argument, is linked by means of instances of the *Involves* relationship to each concept and relation proposed to define the ontology. In addition, this position is related to the *icq2* competency question by the *RespondTo* relationship. For the sake of simplicity, the associations that link each ontology element with the ontology are not shown in this figure..

Fig. 10 presents the conceptual interpretation of a fragment of the ontology design process corresponding to the case study. The project evolves from the *Root Model Version*, which was empty, to the *SpecificationModel1* model version by applying the sequence of operations named *CQsAndTermDefinition*. This sequence was captured by the tool from the operations performed by the ontologist during the first phase of the design process. Such sequence materialized the decisions related to the definition of competency questions and the derivation of concepts from them. Capturing these decisions let reconstruct the ontology design process. Fig. 10 also shows the evolution from the *SpecificationModel1* model version to the *SpecificationModel2* one, obtained by applying a new sequence of operations, named *CQRefinement*, which materialized the design decisions related to the refinement of competency questions.

Fig. 10 shows the *CQsAndTermDefinition* sequence of operations that includes the *addOntology* operation and the set of *addInformalCQ* and *deriveConcept* operations. To execute them, specific argument values were given; as the result, a set of ontology elements that are represented at both the Repository and Versions levels, has been obtained. In particular, the ontology is represented as $isa88Ontology_{VO}$ in the Repository and $isa88Ontology_{\upsilon 1}$ at the Versions level. Similarly, the competency questions and the first identified terms have been generated with a double representation. For simplicity reasons, the figure only shows those versionable objects corresponding to competency questions 1 and 8 ($icq1_{VO}$ and $icq8_{VO}$, respectively), and two terms ($product_{VO}$ and $masterRecipe_{VO}$), but it should be noted that their corresponding first versions have also been created at the Versions level ($icq1_{\upsilon 1}$, $icq8_{\upsilon 1}$, $product_{\upsilon 1}$ and $masterRecipe_{\upsilon 1}$, in Fig. 10). In turn, the second sequence of operations that was captured by TracED(aaS) comprises the set of *refinesCQ* operation executions, which have been applied to refine specific competency questions into simpler ones. In order to understand the effects of the execution of these operations on the *SpecificationModel2* model version, let us consider only the operations that were applied to refine competency question 8 (represented by the $icq8_{VO}$ and $icq8_{\upsilon 1}$ objects in Fig. 10):

(i) a version of *ComplexCQ* ($ccq8_{\upsilon 1}$) was added,
(ii) $icq8_{\upsilon 1}$ is deleted from the model,

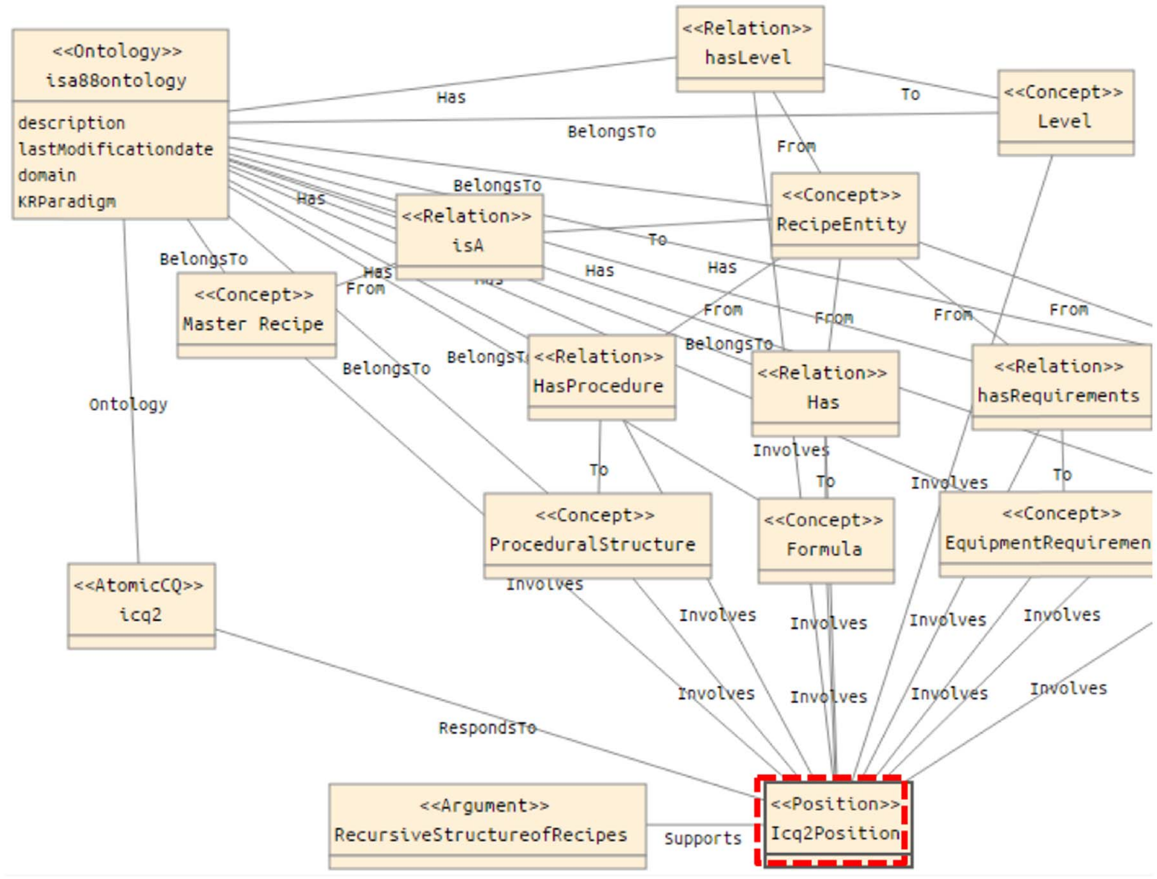


Fig. 9. Definition of the position associated with competency question 2 (*icq2*, see Table B.1 of Appendix B).

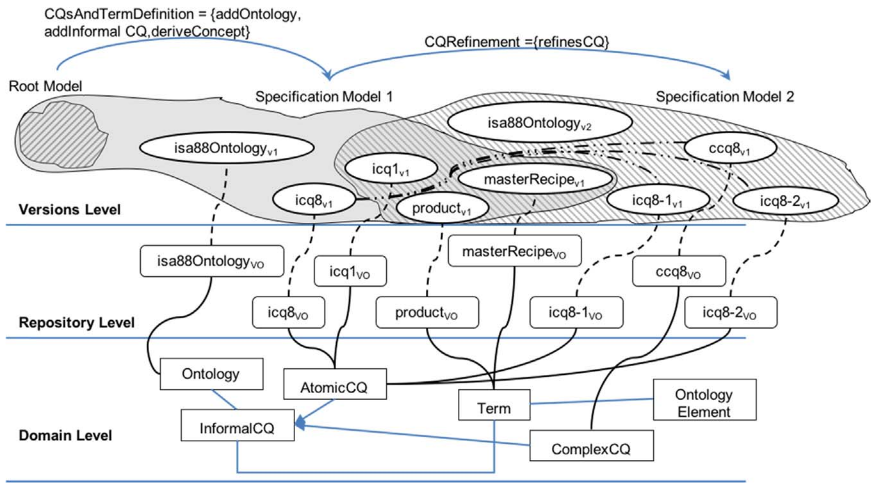
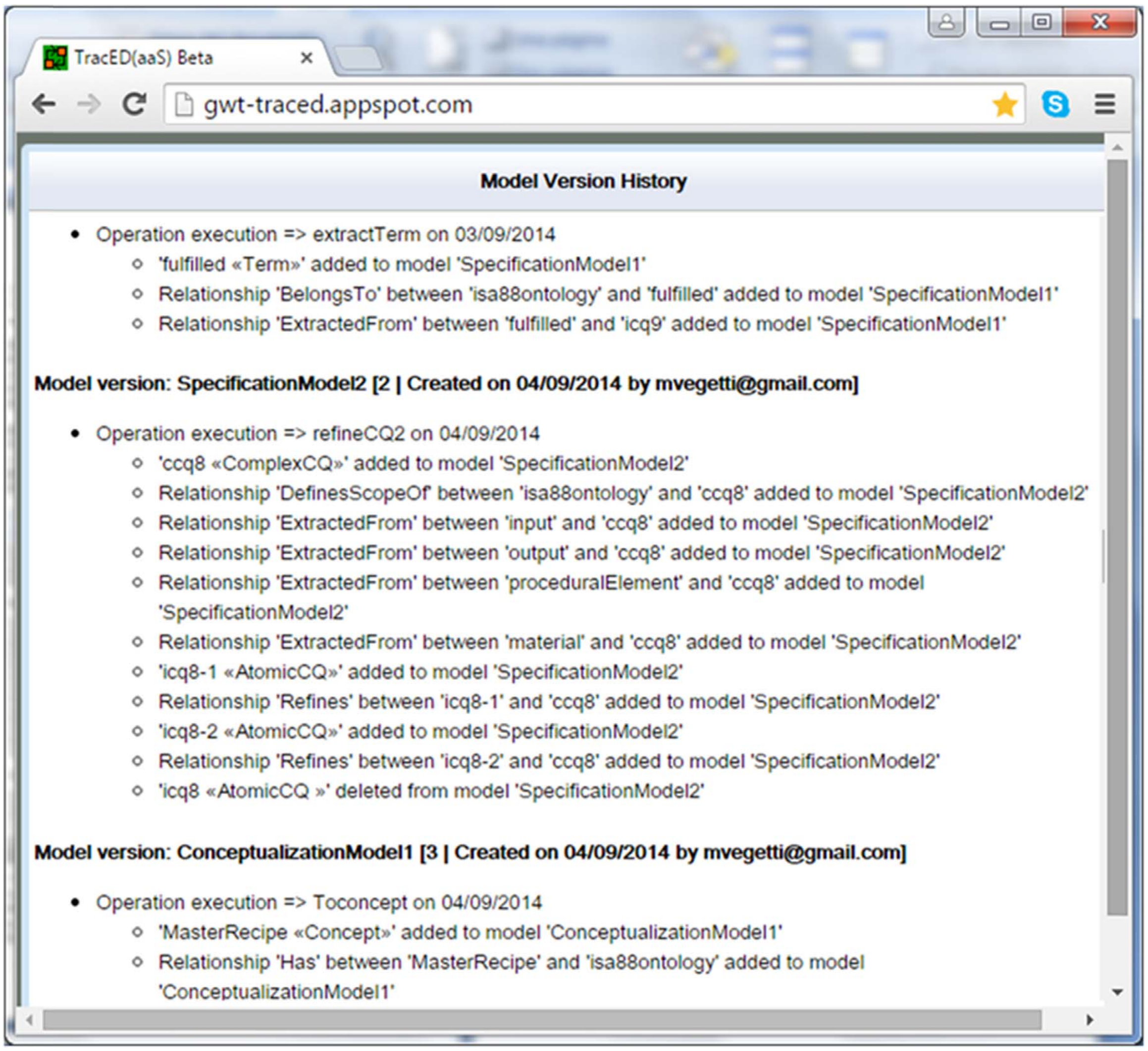


Fig. 10. Representation of a fragment of the ontology development process captured by TracED(aaS).

(iii) two new atomic competency questions (*AtomicCQs*, *icq8-1*$_{\upsilon 1}$ and *icq8-2*$_{\upsilon 1}$) were added,

(iv) associations between the complex competency question (*ccq8*$_{\upsilon 1}$) and the ones that refine it (*icq8-1*$_{\upsilon 1}$ and *icq8-2*$_{\upsilon 1}$), as well as links between the new competency questions and the *isa88Ontology* ontology, were included.

Each time a sequence of operations was performed, the *Version Manager* created a model history link to maintain the association between the predecessor and the successor model versions. This allows tracking the design operations and the decisions behind them. Moreover, a version history link was created for each executed operation in the sequence, in order to save the trace among the



Model Version History

- Operation execution => extractTerm on 03/09/2014
  - 'fulfilled «Term»' added to model 'SpecificationModel1'
  - Relationship 'BelongsTo' between 'isa88ontology' and 'fulfilled' added to model 'SpecificationModel1'
  - Relationship 'ExtractedFrom' between 'fulfilled' and 'icq9' added to model 'SpecificationModel1'

Model version: SpecificationModel2 [2 | Created on 04/09/2014 by mvegetti@gmail.com]

- Operation execution => refineCQ2 on 04/09/2014
  - 'ccq8 «ComplexCQ»' added to model 'SpecificationModel2'
  - Relationship 'DefinesScopeOf' between 'isa88ontology' and 'ccq8' added to model 'SpecificationModel2'
  - Relationship 'ExtractedFrom' between 'input' and 'ccq8' added to model 'SpecificationModel2'
  - Relationship 'ExtractedFrom' between 'output' and 'ccq8' added to model 'SpecificationModel2'
  - Relationship 'ExtractedFrom' between 'proceduralElement' and 'ccq8' added to model 'SpecificationModel2'
  - Relationship 'ExtractedFrom' between 'material' and 'ccq8' added to model 'SpecificationModel2'
  - 'icq8-1 «AtomicCQ»' added to model 'SpecificationModel2'
  - Relationship 'Refines' between 'icq8-1' and 'ccq8' added to model 'SpecificationModel2'
  - 'icq8-2 «AtomicCQ»' added to model 'SpecificationModel2'
  - Relationship 'Refines' between 'icq8-2' and 'ccq8' added to model 'SpecificationModel2'
  - 'icq8 «AtomicCQ »' deleted from model 'SpecificationModel2'

Model version: ConceptualizationModel1 [3 | Created on 04/09/2014 by mvegetti@gmail.com]

- Operation execution => Toconcept on 04/09/2014
  - 'MasterRecipe «Concept»' added to model 'ConceptualizationModel1'
  - Relationship 'Has' between 'MasterRecipe' and 'isa88ontology' added to model 'ConceptualizationModel1'

Fig. 11. Version manager history windows.

performed operation, its argument values, and its resulting object versions. It is possible to see an example of these links in Fig. 10. For clarity reasons this figure only shows the version history links that relate the *icq8*$_{v1}$ object versions in *Specification*Model1 with the *ccq8*$_{v1}$, *icq8-1*$_{v1}$, and *icq8-2*$_{v1}$, object versions in *Specification*Model2.

By means of the history links it is possible to reconstruct the history of a given model version starting from the root one. The *Version Manager* presents such information in the so-called *History Window,* shown in Fig. 11.

It can be seen that TracED(aaS) allows keeping knowledge about the development evolution of *isa88ontology*, from which it is possible to identify:

(i) the predecessor and successor model versions of *SpecificationModel2*;
(ii) the applied sequences of operations that originated each model version;
(iii) the object versions that were generated (such as c*cq8, icq8-1 and icq8-2*), or that were deleted *(icq8*) as a result of a given operation execution (*refinesCQ* in *SpecificationModel2* model version).

In addition, the *Version Manager History Window* (Fig. 11) presents detailed data about each applied operation, such as the time point at which a given operation was executed and the resulting object versions. In this example, the history window shows that a *refineCQ* operation was executed on *SpecificationModel2* on date 04/09/2014. The execution of this operation also implied the addition of an instance of the *ComplexCQ* design object type (*ccq8*), two instances of *AtomicCQ* (*icq8-1* and *icq8-2*) and some associations (one instance of *DefinesScopeOf*, four instances of *ExtractedFrom* and two instances of *Refines*).

6. Conclusions and future work

This contribution presented ONTOTracED, which is a framework aimed at capturing and tracing ontology development processes. The framework is based on a conceptual model of generic engineering design projects, an Ontological Engineering Domain Model (OEDM), which specifies design objects and operations according to a methodology that guides ontology development processes. It relies on a computational environment, named TracED(aaS), which implements such model. The capabilities of TracED(aaS) have been presented and afterwards illustrated by means of a case study that addresses the development of an ontology of industrial interest. The example shows that it is possible to keep track of the development process along with its associated products and to store its history, by allowing the future retrieval of knowledge and experience.

Although the OEDM represents design objects and operations of the adopted ad-hoc methodology, the proposed framework is flexible enough to be used in the development of ontologies that rely on other development methodologies or on approaches that address particular fields. In such a case, it would only be necessary to define a new Ontological Engineering Domain Model representing the design objects and particular operations of the required methodology. If needed, the TracED(aaS) domain editor can be used to extend the proposed Ontological Engineering Domain Model or to create a new one.

This contribution shows that ONTOTracED is a comprehensive framework. It has been extensively validated and verified in actual applications, though only a partial view of such testing experience is presented in this contribution. Since the whole ontology development process, its history and rationale, as well as all the intermediate products can be apprehended in an integrated fashion, it can be concluded that ONTOTracED makes an important contribution to the ontological engineering field.

Regarding future work, it is important to remark that TracED(aaS) needs to be extended with features like a query processor in order to facilitate the easy retrieval of more history information. As examples of relevant queries, please consider the following ones:

(i) Given a concept, which requirement originated its addition to the ontology?
(ii) Which argument supported the inclusion of a given concept into the ontology?
(iii) Which argument rejected the inclusion of a given concept into the ontology?
(iv) Given a competency question, which were the different positions that have been proposed to answer it? Which one has been selected and why?
(v) Given a competency question, which is the position that was selected to answer it? And which is the argument that supports this decision?

Currently, the answers to all these questions can be found in an ad-hoc fashion by tracing the history information that is described in the history windows. Therefore, the development of new TracED(aaS) capabilities to automate knowledge retrieval from the captured information would be very useful and will be tackled in the future.

Finally, another task that will be addressed in the future is related to accomplishing the final steps of the integration of TracED(aaS) with Protégé according to the guidelines provided in Section 4.

Acknowledgments

The authors acknowledge the financial support received from CONICET (PIP 11220110100906 and PIP 11220110101145), ANCYPT (PICT-2013 1310), UNL (CAI+D 2011 50120110100473LI) and UTN (UTI3810TC, UTI3804TC and UTI3803TC).

Appendix A. Functional specifications of operations in OEDM$_{ah}$

In OEDM$_{ah,}$ the proposed operations are classified into four categories: Basic, Refinement, ODP Application, and Design Rationale. Operations in the first group are related to the addition or removal of certain object versions from a model version. Operations in the second and third group are more complex and are associated with the refinement of design activities and the application of ontology design patterns, respectively. The last group considers operations related to the capture of design rationale. Figs. A.1–A.3 show the functional specification of operations in each of these groups. Functional specifications provide an outline on how operations would be defined, using a computational tool that implements the proposed model. Fig. A.1 presents functional specifications of three basic operations (*addTerm*, *addCompetencyQuestion* and *addRelation* ones). The body of each operation is defined in terms of primitive operations, such as *add(nt, Term,* l_{props}*)* in the *addTerm* operation, and auxiliary functions, such as *addAssociation(o, r, BelongsTo)* in the *addRelation* operation. The *addAssociation* is a predefined auxiliary function included in the operation model to establish associations between versionable objects at the repository level. In addition, the operation's body can contain other non-primitive operations already defined in the proposed OEDM….

The signature of an operation indicates its name and its required parameters (name and types enclosed in brackets). For example (see Fig. A.1.), the parameters of the *addTerm* operation are *o* (the ontology in which the term is going to be added), *nt* (the name of the term to be included), l_{cq} (a list of competency questions in which the term appears), and l_{Props} (a collection having the values of the *properties* of the new term), which could be empty. The operation aims at incorporating a new term *nt* to the *o* ontology. To achieve this, firstly, a version of the *t* term is added (*t:=add(nt, Term,* l_{props}*)).* After that,

```
addTerm(o: Ontology, nt: String, $l_{cq}$: Collection [InformalCQ],  $l_{props}$: Collection[PrimitiveDataType])
    t:= add(nt, Term, $l_{props}$)
    for each cq in $l_{cq}$
        addAssociation(t, cq, ExtractedFrom)
    end foreach
    addAssociation(o, t, BelongsTo)
end
addCompetencyQuestion(o: Ontology, ncq:String, $l_{exp}$: Collection[String])
    icq:= add(ncq, AtomicCQ,$l_{exp}$)
    addAssociation(o, icq, DefinesScopeOf)
end
addRelation(o:Ontology, nr:String, fc:Concept, tc:Concept, $l_{props}$: Collection[PrimitiveDataType])
    r:add(nr,Relation, $l_{props}$)
    addAssociation(o, r,  BelongsTo)
    addAssociation(r, fc, From)
    addAssociation(r, tc, To)
end
```

Fig. A.1. Functional specification of some basic operations.

```
refineCQ(rcq:AtomicCQ, ncq:String, $l_{cq}$: Collection [String])
    o:=get(rcq, Ontology)
    $l_{terms}$:= get (rcq,Term)
    ncqv := add(ncq, ComplexCQ)
    addAssociation(o, ncqv, DefinesScopeOf)

    for each t in $l_{terms}$
        addAssociation(t, ncqv, ExtractedFrom)
    end foreach

    for each acq in $l_{cq}$
        acqv := add(acq, AtomicCQ, null)
        addAssociation(o, acqv, DefinesScopeOf)
        addAssociation(acqv, ncqv, Refines)
    end foreach
    delete(o, rcq)
end

term2Concept(t: Term)
    o:= get(t, Ontology)
    tn= get(t, Name)
    $l_{props}$:=get(t, Props)
    nc:= add(tn, Concept, $l_{props}$:)
    addAssociation(o, nc, BelongsTo)
    addAssociation(t, nc, IsRefinedInto)
end

term2Individual(t:Term, c:Concept)
    o:get(t, Ontology)
    tn= get(t, Name)
    $l_{props}$:= get(t, Props)
    i:= add(tn, Individual, lprops)
    addAssociation(o, i, BelongsTo)
    addAssociation(t, i, IsRefinedInto)
    addAssociation(i, c, InstanceOf)
end

formalizeCQ(infcq:InformalCQ, fcq:String, fexp:Collection[String])
    o:=get(infcq, Ontology)
    ncq:= add(fcq, FormalCQ, fexp)
    addAssociation(o, ncq, BelongsTo)
    addAssociation(infcq, ncq, Formalizes)
end

applyValuePartitionPattern(vpp:String, cp:Concept, $l_{parts}$:
Collection[PrimitiveDataType])
    odp:=add(vpp, ValuePartitionODP, null)
    o:=get(cp, Ontology)
    addAssociation(o, odp, BelongsTo)
    addAssociation(odp, cp, Contains)
    uof:= addConcept(o, "UnionOf", null)
    addAssociation(uof, cp, "EquivalentConcept")
    for each p in $l_{parts}$
        np:=add(p, Concept, null)
        addAssociation(np, odp, Contains)
        r: =addRelation(o, "unionOf", uof, np, null)
        addAssociation(r, odp, Contains)
        r: =addRelation(o, "SubClassOf", uof, cp, null)
        addAssociation(r, odp, Contains)
    end foreach
    $l_p$= get(cp, SublclassOf)
    addDisjoiness(np, $l_p$)
end
```

Fig. A.2. Functional specification of some complex operations.

associations between *t* and each competency question version in l_{cq}, are inserted, by employing *loop* functions for iterating on the collection elements. To instantiate *loop* commands in the functional specification, the special syntactic element "for each … in …" is used. Finally, the added term version *t* is linked to the *o* ontology to which it belongs (*addAssociation(o, t, BelongsTo)*).

Fig. A.1 also illustrates the *addCompetencyQuestion*, which includes an informal competency question version (*add(ncq, InformalCQ)*) and links it to the ontology that is indicated as its scope (*addAssociation(o, icq, DefinesScopeOf)*). The *AddRelation* operation (Fig. A.1) inserts a new version of a relation between two concepts, which are passed as parameters to the operation. In order to do that, a *Relation* version is added (*add(nr, Relation, lprops)*); then, it is linked to the ontology to which it belongs (*addAssociation(o, r, BelongsTo)*) and to the concepts that correspond to the extremes of the inserted relation (*addAssociation(r, fc, From) and addAssociation(r, tc, To)*).

As it was previously mentioned, the second group is related to refinement operations. This group contains operations to split a competency question into simpler ones, to transform a given *term* into a *concept*, *relation* or *individual*, and to formalize an *informal competency question*, *constraint* or *assumption*. Fig. A.2 presents the functional specification of some operations that are included in this group: *refineCQ*, *term2Concept*, *term2Individual*, and *formalizeCQ*. These operations consist of the refinement of certain design objects existing in the predecessor model version. Some relations in which the original ontology element took part, such as the ontology to which it

```
addPosition(pname:String, icqv:InformalCQ,
aname:String,elist:Collection[OntologyElement])
  pv := add(pname, Position, null)
  addAssociation(icqv, pv, RespondsTo)
  argv:= add(aname, Argument, null)
  addAssociation(pv, argv, Supports)
  for each e in elist
        addAssociation(pv, e, Involves)
  end foreach
end
addAlternative(pv1:Position, pv2:Position)
  addAssociation(pv1, pv2, Alternative)
end
addRejectingArgument(aname:String, pv: Position)
  av:add(aname, Argument, null)
  addAssociation(av, pv, ObjectsTo)
end
addSupportingArgument(aname:String, pv: Position)
  av:add(aname, Argument, null)
  addAssociation(av, pv, Supports)
end
rejectPosition(dname:String, pv:Position, argv:Argument)
  dv:=add(dname, Decision, null)
  addAssociation(pv, dv, Rejects)
  addAssociation(dv, argv, Uses)
end
selectPosition (dname:String, pv:Position, argv:Argument)
  dv:=add(dname, Decision, null)
  addAssociation(pv, dv, Select)
  addAssociation(dv, argv, Uses)
end
```

Fig. A.3. Functional specification of some design rationale related operations.

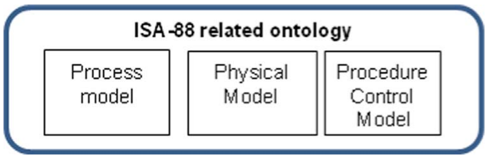


Fig. B.1. Modules of the Isa88ontology.

belonged, or the terms extracted from a competency question, should be maintained by the resulting design objects. For this reason, the operations shown in Fig. A.2 use get functions to retrieve such individuals from the previous model version and then, new associations between the added design object version and the retrieved ones are defined in the new model version by the corresponding operations.

As an example, consider the *refineCQ* operation, which aims at splitting off a competency question into more specific ones. This operation transforms a version of an atomic informal competency question into a complex one and creates new competency questions that are the refinement of the original one. The competency question to be refined (*rcq*) and the names/descriptions for the new ones (l_{cq}) are passed as parameters. The *RefineCQ* operation starts by obtaining the ontology *o*, whose scope is defined by the competency question to be refined (*o:=get(rcq, Ontology)).* Then, the next action is to establish a new relation between *o* and the refined competency question (*add(o, ncq, DefinesScopeOf)*), and to get those terms that are linked to *rcq* by the *ExtractedFrom* associations. Then, the operation sets new links between each term and the new competency question (*addAssociation(t, ncq, ExtractedFrom,* in the first *for each* statement). Moreover, specific competency questions are added (*add(acq, AtomicCQ, null)*) and linked to the refined one (*addAssociation(acqv, ncqv, Refines)*) and to the ontology (*addAssociation(o, acqv,*

Table B.2
Refined competency questions and identified terms.

Competency question	Identified terms
Which are the material inputs associated with a given procedural element?	Material, inputs, procedural element
Which are the material outputs associated with a given procedural element?	Material, outputs, procedural element

Table B.1
Some competency questions and identified terms.

Competency question	Identified terms
Given a product P, which is the Master Recipe (MR) of a batch of such product?	Product, batch master recipe
Which are the candidate resources for a given MR?	Resource, master recipe
Given a Master Recipe, which are the Control Recipes (CR) that have been created from it during a specific period?	Master recipe, control recipe, created from
Given a Control Recipe, which is the MR from which it has been generated?	Control recipe, master recipe, generated from
Which are the materials and their corresponding quantities associated with the specification of a given batch at the Master Recipe level?	Material, quantity, batch, master recipe, level
Which are the candidate equipment items that can be assigned to each of the Unit Procedures belonging to a given MR?	Equipment, assigned to, unit procedure, master recipe
Which are the procedural elements associated with a given MR?	Procedural element, master recipe
Which are the material inputs and outputs associated with a given procedural element?	Material, input, output, procedural element
Which are the conditions that should be fulfilled to start a given Unit Procedure?	Condition, fulfilled, start, unit procedure
Which are the conditions that should be fulfilled during the execution of a given Unit Procedure?	Condition, fulfilled, execution, unit procedure
Which are the Operations comprised in a given Unit Procedure?	Operation, comprised in, unit procedure
Given an operation O, which other operations have to be finished before O can start?	Operation, be finished, start
Given an operation O, which operations require the end of O to begin their execution?	Operation, end, begin, execution
Which operations are executed in parallel with a given operation O?	Operation, execution, parallel
Which equipment units are capable of executing a given operation?	Equipment unit, capable of, execution, operation
Which are the conditions that should be fulfilled to start a given operation?	Condition, fulfilled, start, operation
Which are the phases comprised in a given operation?	Phase, comprised in, operation
Given a set of master recipes associated with various product batches, which are the materials that they have in common?	Master recipe, material, in common
Given a set of master recipes associated with various product batches, which are the required equipment units that they have in common?	Master recipe, requirement, equipment, in common
In which unit of measure is expressed a given parameter?	Unit of measure, parameter
Given a Master Recipe which is its current version?	Master recipe, version
…	

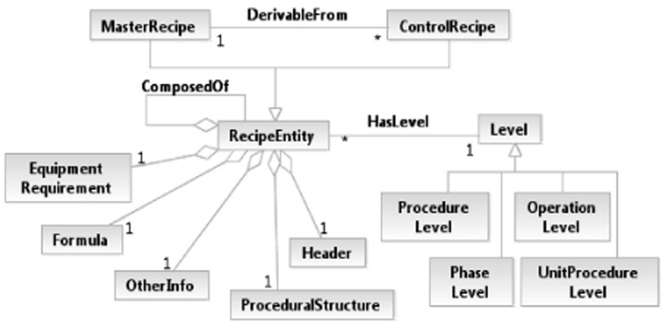


Fig. B.2. Recipe entity definition.

DefinesScopeOf)). Finally, the original atomic competency question (*rcq*) is deleted from the current model version. Refinement operations involve refining the terms that are identified during the specification phase, into concepts, individuals or relations, in the next phase of the ontology development. Two of these operations are illustrated in Fig. A.2. The *term2Concept* and *term2Individual* operations allow designers to refine a given term *t* into a concept or individual, respectively. Both operations start by obtaining the name that identifies the term, the ontology to which it belongs and its properties. Then, a concept/individual is added to the ontology and linked to it and to the refined term by means of the *BelongsTo*/*isRefinedInto* association, respectively.

Formalization operations also belong to the second group of operations. Their objective is to capture expressions that formalize competency questions, assumptions or constraints. An example of this type of operations is shown in Fig. A.2. The *FormalizeCQ* operation adds a version of *FormalCQ* that represents the formal expression of the informal competency question that is passed as a parameter, and links the formal competency question to the informal one and to the ontology.

The third group of operations is related to the application of ontology design patterns. As an example, Fig. A.2 presents the *applyValuePartitionPattern* operation, which allows ontologists to apply the Value Partition ontology design pattern (*ValuePartitionODP*). This pattern describes how to model a partition; i.e., a given concept that is divided into several disjoint concepts. The mail parameters of this operation are: (a) the concept to be partitioned (*cp*), and (b) the names of the partitions (l_{parts}). The operation adds the new pattern to the model and links it to the ontology. Then, it creates a new concept, which is also associated with the *cp* concept, representing the union of the partitions, and associates it with the partitioned concept by means of the *EquivalentConcept* relation. The operation also creates a new concept for each element of l_{parts} (*np:=add(p, Concept, null)*) and links each of them to the *unionOf* concept (*addRelation(o, "UnionOf", uof, np, null)*), as well as to the design pattern version (*addAssociation(np, odp, Contains*), which the operation also creates (*odp:=add(vpp, ValuePartitionODP, null)*). The relations that are created are also associated with the design pattern version (*addAssociation(r, odp, Contains*). Finally, all the new added concepts are linked, as subclasses, to the partitioned concept and all of them are associated among themselves by a disjoint relation by means of the operation *addDisjoiness(np, lp),* whose implementation is not shown in this article due to space limitations.

Additionally, it would be important to rely on operations that can be applied to design rationale concepts. These operations embody those ontologist's decisions which have a fundamental impact on the ontology conceptual model, and which are to be documented in order to enable the future evolution of the ontology. Some examples of these operations are shown in Fig. A.3. The operation *addPosition* is intended to link a competency question with a set of ontology elements. A position is added as an object version (*add(pname, Position, null)*) and it is related to the competency question to which the new position responds (*addAssociation(icqv, pv, RespondsTo)*) and to the ontology elements that are added to the ontology to answer such question (*addAssociation(pv, e, Involves)*). In addition, the position is supported by an argument (see operations *add(aname, Argument, null)* and *addAssociation(pv, argv, Supports)* in Fig. A.3).

The *addSupportingArgument* and *addRejectingArgument* operations are intended to make explicit the reason that supports or rejects a position, by adding it as an object version (*add(aname, Argument, null)*). In both operations, an instance of a relationship concept indicates the argument that is supporting (*addAssociation(av, pv, Supports*)) or rejecting (*addAssociation(av, pv, ObjectsTo))* the position. In addition, the *selectPosition* and *rejectPosition* operations enable capturing the ontology decision about the best answer to a competency question among all the alternative possible positions. This operation also captures the argument that supports the decision.

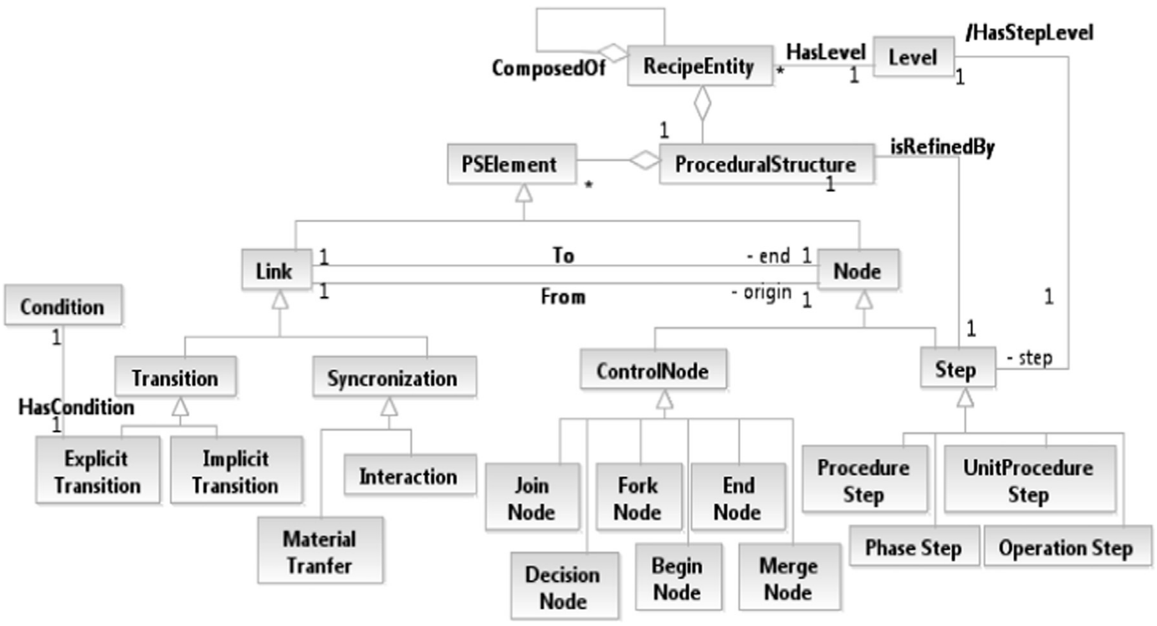


Fig. B.3. Procedural structure definition.

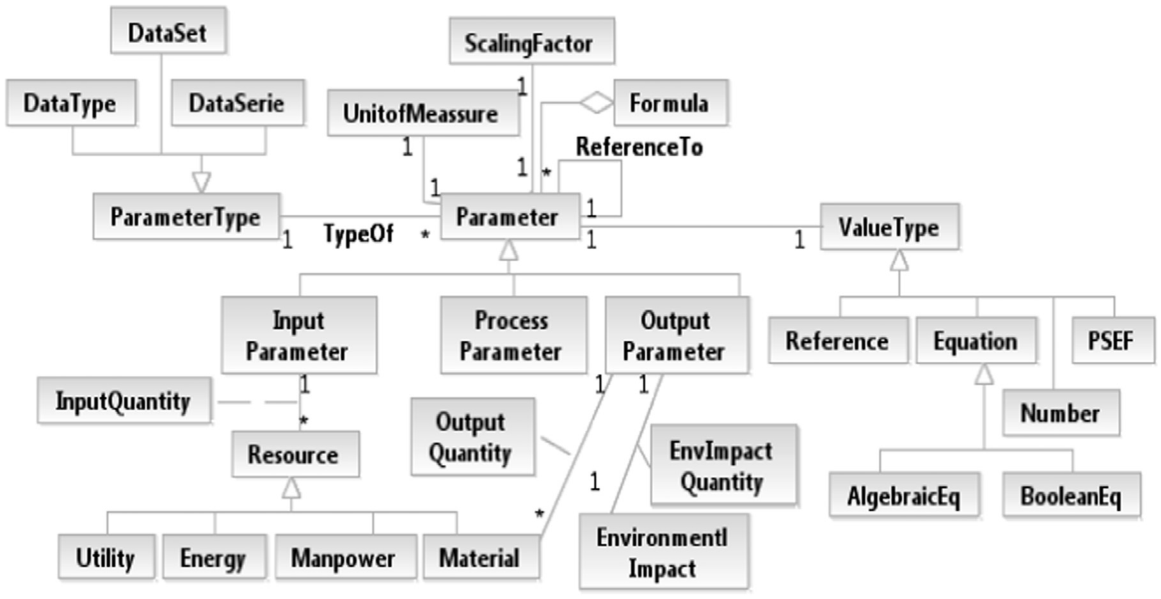


Fig. B.4. Formula Definition.

Appendix B. ISA-88 procedure control model ontology

This appendix aims at describing the ontology that formalizes the control model of the ISA-88 standard. The development of this ontology module was supported by the framework presented in this article. Firstly, a brief description of the ISA-88 ontology is provided. Then, an overview of the ontology is introduced.

The ISA-88 standard originally addressed batch process control issues, and was later extended to tackle discrete manufacturing and continuous processes. Its original purpose was to solve problems which were widely acknowledged by the batch control community: (i) no universal model for batch control, (ii) user difficulties to communicate requirements, (iii) problems to integrate informatic applications coming from different vendors, (iv) troubles to configure control systems. To overcome these problems, the developers of the standard set the goal of separating product knowledge, which is kept in recipes, from the equipment capabilities. In addition, recipes are defined at different levels of abstraction in order to augment portability. Due to these features, recipes can be created and easily modified, without requiring changes in the control system. Therefore, recipes turn to be more flexible, reusable, and maintainable, their validation is simplified and can be moved from one manufacturing system to another. The ISA-88 standard organizes knowledge along three different perspectives: the physical model, the process model, and the procedural control model. These representations, which are hierarchical and are related among themselves, are employed to specify recipes.

The *physical model* organizes the enterprise into sites, areas, process cells, and units, as well as equipment and control modules. The *process model* provides a high level representation of a batch process, and it is the basis for defining equipment independent recipe procedures, i.e. the so-called *General* and *Site Recipes*. The ISA-88 *process model* prescribes that a batch process is comprised by an ordered set of process stages, which in turn are comprised by process operations, and, finally, these ones by process actions. The *procedural control model* is a hierarchical model that depicts the orchestration of procedural elements to carry out process-oriented tasks associated with *Master* and *Control Recipes*. This model describes recipe procedures, each of which consists of an ordered set of unit procedures, which in turn are comprised by an ordered set of operations, each of which consisting of an ordered set of phases. In the proposed ontology, these models are organized in different modules, as shown in Fig. B.1..

The remaining of this appendix provides some additional highlights of the ad-hoc development process that was proposed and adhered to during the construction of the ontology. It was captured with the aid of TracED(aaS), as it is presented in Section 5. These highlights just cover two iterations of the requirements specification phase, as well as three iterations of the conceptualization one.

Procedure control module – requirements specification phase

Iteration one (Table B.1).
Iteration two (Table B.2).

Procedure control module - conceptualization phase

Iteration one

The *RecipeEntity* is the combination of a procedural element with associated recipe information. A *Recipe* is a *RecipeEntity*, which is defined at the highest level of the procedural control model. Each *Recipe* is built up of lower-level recipe entities. These levels are hierarchically organized according to the *procedural control model* (*Procedure*, *UnitProcedure*, *Operation* and *Phase*). A *MasterRecipe* is a *RecipeEntity* at the *ProcedureLevel,* which is the highest one. Similarly, a *ControlRecipe* is also a *RecipeEntity* defined at the highest level of the procedure control model and it is derivable from a *MasterRecipe*. Fig. B.2 also shows the relation of a *RecipeEntity* with its components: *Header*, *EquipmentRequirement*, *ProceduralStructure* and *Formula*..

Iteration two

The ISA-88 standard suggests the use of Procedural Function Charts (PFC) to describe the procedural structure of a recipe. The ontology proposed in this contribution formalizes the elements included in this informal diagram. A *ProceduralStructure* is a set of procedural elements (*PSElement* in Fig. B.3) that depicts the procedural logic for all recipe levels: recipe procedure, recipe unit procedure, and recipe operation..

A Procedural Element may be a *Link* or a *Node*. A *Node* is a procedural element that represents an action (procedure, unit procedure, operation or phase) or a symbol that controls the transition between steps. A *Link* defines a relation between two nodes.

Different types of links and nodes are defined in order to formalize all PFC symbols. The links that were included in the ontology are:

- *Transition*: a direct link between nodes.
- *ImplicitTransition*: transition having a single condition that states that the directly preceding step has to finish its execution.
- *ExplicitTransition*: transition having a condition that has to evaluate to true in order to activate the step that follows the transition.
- *Synchronization*: link that relates recipe elements among which there is a certain form of synchronization.
- *MaterialTransfer*: link that represents material transfer from a step to another.
- *Interaction*: kind of synchronization that does not involve material movement.

The proposed ontology defines the following concepts to represent the different types of *Nodes* in PFC:

- *Step*: a node that represents a recipe procedural element: procedure recipe entity, unit procedure recipe entity, operation recipe entity or phase recipe entity.
- *ProcedureStep:* step defined at the highest level of the Procedure Control Model, called Procedure.
- *UnitProcedureStep:* step belonging to the Unit Procedure level of the Procedure Control Model.
- *OperationStep*: step that is defined at the Operation level of the Procedure Control Model.
- *PhaseStep*: step belonging to the lowest Phase level of the Procedure Control Model.
- *ControlNode*: node that controls the intended thread of execution of the recipe procedural elements.
- *BeginNode*: identifies the start of each procedural structure and/or each subordinate structure.
- *EndNode*: indicates the conclusion of a procedural structure and/or a subordinate structure.
- *ForkNode*: defines the start of independent threads of execution of certain recipe elements, which are executed in parallel.
- *JoinNode*: indicates the end of independent threads of execution.
- *DecisionNode:* specifies the beginning of alternative threads of execution.
- *MergeNode*: shows the joining of alternative threads of execution.

Valid diagrams have to follow consistent rules for the execution threads. Therefore, the formalization of the procedural structure allows defining constraints that help building valid procedural structures. For example, the specialization of the *Step* concept in order to represent steps at different levels of the *Procedural Control Model* facilitates the definition of constraints that avoid the construction of a PFC in which *UnitProcedures*, *Operations* or *Phases* could be mingled in the same diagram.

Iteration three

As it is shown in Fig. B.4, a *Formula* is modeled as a set of *Parameters*, which may be categorized as process inputs, process outputs, or process parameters. Parameter values may be simple values, expressions, or references to parameters that are defined at the same level or at higher levels in the procedural hierarchy. Values that are expressions may include references to other parameters (*ReferenceTo* association in Fig. B.4)..

An input parameter represents the identification and quantity of a resource required to make a batch of product. These resources may be raw materials, energy, manpower or utilities. Moreover, the *Resource* class may be specialized to take into account other supplies pertaining to specific industrial domains. Similarly, an output parameter specifies a certain material and the quantity that is expected to result from the execution of a certain recipe. A process parameter details information such as temperature, pressure, or time that is pertinent to the manufacture of a batch of product, but which does not fall into the input or output categories.

Each parameter is associated with a value type, a unit of measure, a scaling factor and a reference type. The corresponding definitions are introduced in the following paragraphs:

- *Value type*: specifies how the parameter value is interpreted. It includes: basic data types, data sets that define material transactions (transfer, consumption, generation of material); or data series (e.g., a temperature profile that needs to be tracked).
- *UnitofMeasure*: identifies the engineering units of measure for the *Value* (e.g., kg , pounds).
- *ReferenceType*: specifies the way parameters are related if the associated parameter has references to others. The types of relations may be: Algebraic or Boolean equations, Product specific entry forms that work on one or more parameters, Deferral of parameters to different recipe entities (at the same or another level), among others.
- *Scaling Factor*: defines the scaling rule, which indicates how the parameter should be scaled with the batch size.

References

Baroni, P., Romano, M., Toni, F., Aurisicchio, M., Bertanza, G., 2015. Automatic evaluation of design alternatives with quantitative argumentation. Argum. Comput. 6, 24–49.

Bernaras, A., Laresgoiti, I., Corera, J., 1996. Building and reusing ontologies for electrical network applications. In: Proceedings of the European Conference on Artificial Intelligence (ECAI'96), Budapest, Hungary, pp. 298-302.

Beydoun, G., Low, G., García-Sánchez, F., Valencia-García, R., Martínez-Béjar, R., 2014. Identification of ontologies to support information systems development. Inf. Syst. 46, 1–16.

Blomqvist, E., 2009. Semi-automatic Ontology Construction Based on Patterns (Ph.D. Thesis). Linköpings Universitet, Linköping.

Blomqvist, E., Sandkuhl, K., 2005. Patterns in ontology engineering: classification of ontology patterns. In: Proceedings of the 7th International Conference on Enterprise Information Systems, Miami, USA, pp. 413-416.

Borgo, S., Leitão, P., 2007. Foundations for a core ontology of manufacturing. Integr. Ser. Inf. Syst. 14, 1–40.

Brickley, D., Guha, R. V., 2014. RDF Schema 1.1. W3C Recommendation, Available on-line: ⟨http://www.w3.org/TR/2014/REC-rdf-schema-20140225/⟩. (Last accessed: 14.04.15).

Buschmann, F., Meunier, R., Rohnert, H. , Sommenrlad, P. , Stal M., 1996. Pattern-oriented software architecture. A system of patterns, vol. 1, John Wiley & Sons, England.

Cristani, M., Cuel, R., 2005. A survey on ontology creation methodologies. Int. J. Semant. Web Inf. Syst. 1, 49–69.

Darmoul, S., Pierreval, H., Hajri-Gabouj, S., 2013. Handling disruptions in manufacturing systems: an immune perspective. Eng. Appl. Artif. Intell. 26, 110–121.

De Nicola, A., Missikoff, M., Navigli, R., 2009. A software engineering approach to ontology building. Inf. Syst. 34, 258–275.

Debruyne, C., De Leenheer, P., 2014. Using a method and tool for hybrid ontology engineering: an evaluation in the Flemish research information space. J. Theor. Appl. Electron. Commer. Res. 9, 48–63.

Dellschaft, K., Engelbrech, H., Barreto, J.M., Rutenbeck, S., Staab, S., 2008. Cicero: tracking design rationale in collaborative ontology engineering. Lect. Notes Comput. Sci. 5021, 782–786.

Doumbouya, M.B., Kamsu-Foguem, B., Kenfack, H., Foguem, C., 2015. Argumentative reasoning and taxonomic analysis for the identification of medical errors. Eng. Appl. Artif. Intell. 46, 166–179.

Efthymiou, K., Sipsas, K., Mourtzis, D., Chryssolouris, G., 2015. On knowledge reuse for manufacturing systems design and planning: a semantic technology approach. CIRP J. Manuf. Sci. Technol. 8, 1–11.

Elhdad, R., Chilamkurti, N., Torabi, T., 2013. An ontology-based framework for process monitoring and maintenance in petroleum plant. J. Loss Prev. Process Ind. 26, 104–116.

Fernández-López, M., Gómez-Pérez, A., Sierra, J.P., Sierra, A.P., 1999. Building a chemical ontology using methontology and the ontology design environment. Environ. Intell. Syst. 14, 37–46.

Gandon, F., Schreiber, G., 2014. RDF 1.1 XML Syntax. W3C Recommendation. Available on-line: ⟨http://www.w3.org/TR/REC-rdf-syntax/⟩. (Last accessed: 14.04.15)

Gangemi, A., 2005. Ontology design patterns for semantic web content. In: Proceedings of the 4th International Semantic WebConference (ISWC 2005), Galway, Ireland, pp. 262– 276.

Gangemi, A., Lehmann, J., Presutti, V., Nissim, M., Catenacci, C., 2007. C-ODO: an OWL meta-model for collaborative ontology design. In: Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge at the 16th International World Wide Web Conference, Banff, Canada, pp. 1–9

Giménez, D.M., Vegetti, M., Leone, H.P., Henning, G.P., 2008. PRoduct ONTOlogy: defining product-related concepts for logistics planning activities. Comput. Ind. 59,

231–241.
Gómez-Pérez, A., Fernández-López, M., Corcho, O., 2004. Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web 2nd ed.. Springer, London.
Gonnet, S., Henning, G.P., Leone, H., 2007b. A model for capturing and representing the engineering design process. Expert Syst. Appl. 33, 881–902.
Gonnet, S., Vegetti., M., Leone, H., Henning, G.P., 2007a. SCOntology: a formal approach towards a unified and integrated view of the supply chain. In: Adaptive Technologies and Business Integration: Social, Managerial and Organizational Dimension, Idea Group Inc., pp. 137-156.
Gruber, T.R., 1993. A translation approach to portable ontology specification. Knowl. Acquis. 5, 199–220.
Grüninger, M., Fox, M., 1995. Methodology for the design and evaluation of ontologies. In: Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing (IJCAI 95), Montreal, Canada, pp. 258–269.
Guizzardi, G., 2005. Ontological Foundations for Structural Conceptual Models (Ph.D. Thesis) with Cum Laude. Telematica Instituut Fundamental Research Series, 015, Enschede, The Netherlands.
Hai, R., Theißen, M., Marquardt, W., 2011. An ontology based approach for operational process modeling. Adv. Eng. Inform. 25, 748–759.
Hartmann, J., Paslaru Bontas, E., Palma, R., Gómez-Pérez, A., 2006. DEMO – design environment for metadata ontology. Lect. Notes Comput. Sci. 4011, 429–441.
Hartmann, J., Palma, R., Sure, Y., Haase, P., Suárez-Figueroa, M.C., 2005. OMV – ontology metadata vocabulary. In: Proceedings of the workshop Ontology Patterns for the Semantic Web (ISWC 2005), Galway, Ireland, pp. 1–9.
Hauagge Dall Agnol, J.M., Tacla, C.A., 2013. A method for collaborative argumentation in merging individual ontologies. J. Univers. Comput. Sci. 19, 1808–1833.
Heras, S., Botti, V., Julián, V., 2014. Modelling dialogues in agent societies. Eng. Appl. Artif. Intell. 34, 208–226.
ISO, 2009. ISO 15836: The Dublin Core Metadata Element Set, Version 1.1. Available on-line: ⟨http://dublincore.org/documents/dces/⟩. (Last accessed: 14.04.015).
Kifer, M., Lausen, G., Wu, J., 1995. Logical foundations of object-oriented and frame-based languages. J. ACM 42, 741–843.
Konstantinos, K., Vouros, G.A., 2006. Human-centered ontology engineering: the HCOME methodology. Knowl. Inf. Syst. 10, 109–131.
Krötzsch, M., Vrandečić, D., Völkel, M., 2006. Semantic mediawiki. Un: Proceedings of the 5th The International Semantic WebConference (ISWC 2006), Athens, USA, pp. 935–942.
Kunz, W., Rittel, H.W.J., 1970. Issues as elements of information systems. Working Paper no. 131. Institute of Urban and Regional Development, University of California.
Lee, J., 1997. Design rationale systems: understanding the issues. IEEE Expert, 78–85.
Liao, Y., Lezoche, M., Panetto, H., Boudjlida, N., 2016. Semantic annotations for semantic interoperability in a product lifecycle management context. Int. J. Prod. Res. 54, 5534–5553.
Marquardt, W., Morbach, J., Wiesner, A., Yang, A., 2010. OntoCAPE: a Re-Usable Ontology for ChemicalProcess Engineering. Springer, Berlin.
Morbach, J., Yang, A., Marquardt, W., 2007. OntoCAPE – a large scale ontology for chemical process engineering. Eng. Appl. Artif. Intell. 20, 147–161.
Morbach, J., Wiesner, A., Marquardt, W., 2009. OntoCAPE 2.0 – a (re)usable ontology for computer-aided process engineering. Comput. Chem. Eng. 33, 1546–1556.
Natarajan, S., Ghosh, K., Srinivasan, R., 2012. An ontology for distributed process supervision of large-scale chemical plants. Comput. Chem. Eng. 46, 124–140.
NCITS, 1998. Draft proposed American National standard for Knowledge interchange format. National Committee for Information Technology Standards, Technical Committee T2 Information Interchange and Interpretation. Available on-line: ⟨http://logic.stanford.edu/kif/dpans.html⟩. (Last accessed 14.04.2015).
Neuhaus, F., Vizedom, A., Baclawski, K., Bennett, M., Dean, M., Denny, M., Grüninger, M., Hashemi, A., Longstreth, T., Obrst, L., Ray, S., Sriram, R., Schneider, T., Vegetti, M., West, M., Yim, P., 2013. Towards ontology evaluation across the life cycle: the Ontology Summit 2013 Communiqué. Appl. Ontol. 8 (3), 179–194.
ODPS. Ontology Design Patterns (ODPS) Public Catalog. Available on-line: ⟨http://www.gong.manchester.ac.uk/odp/html/index.html⟩. (Last accessed: 14.04.15.
ODPsOrg. Semantic Web Portal Dedicated to Ontology Design Patterns (ODPs). Available on-line: ⟨OntologyDesignPatterns.org⟩. (Last accessed: 14.04.15).
OMG (Object Management Group)., 2014. Ontology definition metamodel, version 1.1. Available on-line: ⟨http://www.omg.org/spec/ODM/1.1⟩. (Last accessed: 19.05.16)
OMG (Object Management Group)., 2015. Unified modeling language (UML) V2.5. Available on-line: ⟨http://www.omg.org/spec/UML/2.5/⟩. (Last accessed: 19.05.16)
Palma, R., Corcho, O., Gómez-Pérez, A., Haase, P., 2011. A holistic approach to collaborative ontology development based on change management. Web Semantics: Science, Services and Agents on the World Wide Web, vol. 9, pp. 299–314.
Panetto, H., Dassisti, M., Tursi, A., 2012. ONTO-PDM: product-driven ONTOlogy for product data management interoperability within manufacturing process environment. Adv. Eng. Inform. 26, 334–348.
Pinto, H.S., Tempich, C., Staab, S., 2009. Ontology engineering and evolution in a distributed world using DILIGENT. In: Handbook on Ontologies, Springer, pp. 153–176.
Potts, C., Bruns, G., 1988. Recording the reasons for design decisions. In: Proceedings of the 10th International Conference on Software Engineering (ICSE ‘1988), Singapore, pp. 418–427.
Roldán, M.L., Gonnet, S., Leone, H., 2010. TracED: a Tool for capturing and tracing engineering design processes. Adv. Eng. Softw. 41, 1087–1109.
Shadbolt, N., Hall, W., Berness-Lee, T., 2006. The Semantic Web revisited. IEEE Intell. Syst. 21 (3), 96–101.
Suárez-Figueroa, M.C., Gómez-Pérez, A., Fernández-López, M., 2015. The NeOn methodology framework: a scenario-based methodology for ontology development. Appl. Ontol. 10, 107–145.
Suárez-Figueroa, M.C., Gómez-Pérez, A., Motta, E., Gangeni, A., 2012. Ontology Engineering in a Networked World. Springer, Berlin.
Sure, Y., Staab, S., Studer, R. 2009. Ontology Engineering Methodology. In: Handbook on Ontologies. Second Edition, 135-152. Springer-Verlang, Berlin Heidelberg
Swartout, B., Ramesh, P., Knight, K., Russ, T., 1997. Toward distributed use of large scale ontologies. In: Proceedings of the Symposium on Ontological Engineering of AAAI, California, pp. 138–148.
Uschold, M., Gruninger, M., 1996. Ontologies: principles, methods and applications. Knowl. Eng. Rev. 11, 93–155.
Uschold, M., King, M., Moralee, S., Zorgios, Y., 1998. The enterprise ontology. Knowl. Eng. Rev. 13, 31–89.
Usman, Z., Young, R.I.M., Chungoora, N., Palmer, C., Case, K., Harding, J.A., 2013. Towards a formal manufacturing reference ontology. Int. J. Prod. Res. 51, 6553–6572.
Vegetti, M., Henning, G.P., 2015. An ontological approach to integration of planning and scheduling activities in batch process industries. Comput.-Aid Chem. Eng. 37, 995–1000.
Vegetti, M., Leone, H., Henning, G.P., 2011. PRONTO: an ontology for comprehensive and consistent representation of product information. Eng. Appl. Artif. Intell. 24, 1305–1327.
Vegetti, M., Roldán, L., Gonnet, S., Henning, G.P., Leone, H., 2012. ONTOTracED: A framework to capture and trace ontology development processes. In: Proceedings of the International Conference on Knowledge Engineering and Ontology Development, pp. 419–422
W3C OWL Working, W3C-OWLwg Group, 2012. OWL 2 Web Ontology Language document overview. Technical Report. Available on-line: ⟨http://www.w3.org/TR/owl2-overview/⟩. (Last accessed: 14.04.15)
Wu, C.G., Xu, X., Zhang, B.K., Na, Y.L., 2013. Domain ontology for scenario-based hazard evaluation. Saf. Sci. 60, 21–34.
Zhang, Y., Luo, X., Li, J., Buis, J., 2013. A semantic representation model for design rationale of products. Adv. Eng. Inform. 27, 13–26.