

Context Dependency Management in Ontology Engineering: a Formal Approach*

Pieter De Leenheer, Aldo de Moor, and Robert Meersman

Semantics Technology and Applications Research Laboratory (STARLab)
Department of Computer Science
Vrije Universiteit Brussel
Pleinlaan 2, B-1050 BRUSSELS 5, Belgium
{pdeleenh, ademoor, meersman}@vub.ac.be

Abstract. A viable ontology engineering methodology requires supporting domain experts in gradually building and managing increasingly complex versions of ontological elements and their converging and diverging interrelationships. Contexts are necessary to formalise and reason about such a dynamic wealth of knowledge. However, context dependencies introduce many complexities. In this article, we introduce a formal framework for supporting context dependency management processes, based on the DOGMA framework and methodology for scalable ontology engineering. Key notions are a set of context dependency operators, which can be combined to manage complex context dependencies like articulation, application, specialisation, and revision dependencies. In turn, these dependencies can be used in context-driven ontology engineering processes tailored to the specific requirements of collaborative communities. This is illustrated by a real-world case of interorganisational competency ontology engineering.

Keywords: *context-driven ontology engineering, context dependency management, ontology evolution, ontology management, lexical disambiguation*

1 Introduction

Though a vast amount of research has been conducted on formalising and applying knowledge representation (KR) models (e.g., [1–5]), there is still a major problem with disambiguation of meaning during the *elicitation* and *application* of an ontology. The problem is principally caused by three facts: (i) no matter how expressive ontologies might be, they are all in fact lexical representations of concepts, relationships, and semantic constraints; (ii) linguistically, there is no bijective mapping between a concept and its lexical representation; and (iii) concepts can have different meaning in different contexts of use.

* We would like to thank our colleagues in Brussels, especially Stijn Christiaens and Ruben Verlinden for the valuable discussions about theory and case. We also would like to thank Tom Mens for the valuable discussions about semantic conflict merging. This research has been partially funded by the EU DIP EU-FP6 507483 project and the EU Leonardo da Vinci Project CODRIVE (BE/04/B/F/PP-144.339).

In collaborative applications, multiple stakeholders have multiple views on multiple ontologies. There, humans play an important role in the interpretation and negotiation of meaning during the elicitation and application of ontologies [6]. A viable ontology engineering methodology requires supporting domain experts in gradually building and managing increasingly complex versions of ontological elements and their converging and diverging interrelationships. Contexts are necessary to formalise and reason about the structure, interdependencies, and versioning of these ontologies, thus keeping their complexity manageable.

1.1 Context and Ontologies

Today in AI and linguistics, the word *context* has gained a (confusing) variety of meanings, which have led to diverse interpretations and purposes of context [7, 8]. Moreover, context is found in various AI application fields such as database integration [9], knowledge translation [10], reasoning [11–13], and lexical disambiguation [7, 14–16]. Furthermore, notions of context were adopted for scalable management of and reasoning on very large bases of formal artefacts using *micro-theories*, in particular for knowledge in Cyc [17, 18] and data models [19]. Here, we only review the key notions which we find useful for the purpose of this article. For a comprehensive survey of context in computer science we refer to [20].

On the Semantic Web [21], the primary role of context is to factor the differences, consequently remove ambiguity, between data sources when aggregating data from them. The Semantic Web is large-scaled and highly distributed in nature, hence, instead of adopting complex context mechanisms that introduce nested contexts and the ability to transcend contexts, rather stronger constraints on the computational complexity and ease of use of the context mechanism are placed [22].

Bouquet et al. [23] introduce an extension to the Web ontology language OWL, viz. Context OWL (C-OWL) for representing so-called contextual ontologies. They argue that not all knowledge should be integrated by an ontology, e.g., knowledge that is mutually inconsistent. In that case the ontology is *contextualised*, and for this reason considered a context. This means its contents are kept local, and are put in relation with the content of other contexts via explicit mappings. Introducing context in OWL required a revision to the OWL syntax and semantics. Giunchiglia [11] was especially motivated by the problem of reasoning on a subset of the global knowledge base. The notion of context is used for this “localisation”.

More recently is the so-called Pragmatic Web vision [24, 25]. This vision claims that it is not necessary (or even possible) to reach for context-independent ontological knowledge, as most ontologies used in practice assume a certain context and perspective of some community. Taking this in consideration, it is natural that ontologies co-evolve with their communities of use, and that human interpretation of context in the use and disambiguation of an ontology often plays an important role. More concretely, the aim is to augment human collaboration effectively by appropriate technologies, such as systems for negotiation during elicitation and application of ontologies for collaborative applications. In this view, the Pragmatic Web complements the Semantic Web by improving the quality and legitimacy of collaborative, goal-oriented discourses in communities.

Based on these viewpoints, we next define our notion of context-driven ontology engineering.

1.2 Context-driven Ontology Engineering

We define context-driven ontology engineering as a set of ontology engineering (OE) processes for which managing contexts (and their dependencies) effectively and efficiently is crucial for their success. The context of an entity is the set of circumstances surrounding it. Based on our literature study, we distinguish four key characteristics of context: (i) contexts package related knowledge: in that case a context defines part of the knowledge of a particular domain; (ii) context provides pointers for lexical disambiguation; (iii) lifting rules provide an alignment between assertions in disconnected knowledge bases; and (iv) statements about contexts are themselves in contexts; in other words, contexts can be embedded or linked [16]. In the next paragraphs we outline some important types of context-driven OE processes that address these issues. These are *macro-level* processes in that they provide the goals of the ontology engineering process.

lexical disambiguation At the start of the *elicitation* of an ontology (cfr. Fig. 1), its basic knowledge elements (such as concepts and relationships) are extracted from various resources such as a text corpus or an existing schema, or formulated by human domain experts. Many ontology approaches focus on the conceptual modelling task, hence the distinction between lexical level (term for a concept) and conceptual level (the concept itself) is often weak or ignored. In order to represent concepts and relationships lexically, they usually are given a uniquely identifying term (or label). However, the context of the resource the ontology element was extracted from is not unimportant, as the meaning of a concept behind a lexical term is influenced by this *elicitation context*. Phenomena such as synonyms and homonym are typical examples of this, and can result in frustrating misunderstanding and ambiguity when unifying information from multiple sources. An analysis of multiple contexts is generally needed to disambiguate successfully [16, 26].

multiple contextualisations Similarly for the *application* of an ontology: the interpretation of the knowledge elements (which are referred to by terms) of the ontology is ambiguous if the context of application, such as the purpose of the user, is not considered. Different domain experts might want to “contextualise” elements of an ontology differently for the purpose of their organisation, by e.g., selection, specialisation or refinement, leading to multiple diverging ontologies that are context-dependent on (read: contextualisations of) the same (part of an) ontology.

ontology integration An important class of OE processes concerns *ontology integration*. This process has been studied extensively in the literature (for a state-of-the-art survey, cf. [27, 28]). Although different groups vary in their exact definition, ontology integration is considered to consist of four key processes (adopting the terminology from [28]):

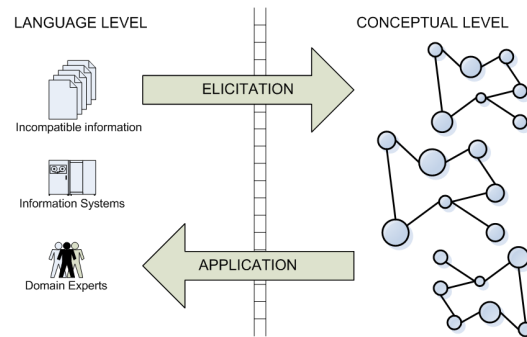


Fig. 1. Ontologies are elicited by extracting knowledge from various sources and are also applied in different contexts.

- 1,2 **mapping and alignment:** given a collection of multiple contextualisations, these often need to be put in context of each other, by means of mapping or aligning (overlapping) knowledge elements pairwise;
- 3 **schema articulation:** a collection of individual knowledge elements may need to be contextualised, by means of a consensual articulation schema of these (overlapping) elements;
- 4 **merging:** a collection of individual knowledge elements may need to be contextualised by means of a consensual merging of these (overlapping) elements¹.

ontology versioning Parts of an ontology might be revised, expanded, or contracted, resulting in branching of that ontology through time [29, 30]. This might possibly trigger a cascade of revisions to all ontologies that are context-dependent on the knowledge elements in the revised part.

1.3 Context Dependency Management

All ontology engineering methodologies use some combination of the above identified context-driven OE macro-processes. However, in their operational implementation of these processes, which we call context-driven OE *micro-processes*, methodologies differ widely. E.g., consider the plethora of Semantic Web and Conceptual Structures research on this matter. Our intention is not to add to these processes themselves, but to identify and *position* them, indicating how they can be used in the bigger picture of real-world ontology engineering processes, such as interorganisational ontology engineering [31]. The question is how to apply and (re)combine them to increase the quality of such processes.

As already mentioned, contexts are important building blocks in our decomposition and linking of ontology engineering processes. *Context dependencies* constrain the

¹ An ontology merging process requires an established articulation schema, which is the result of a successful articulation process. However, in this article we do not work out such relations between contextualisations.

possible relations between the entity and its context. Many different types of context dependencies exist, within and between ontological elements of various levels of granularity, ranging from individual concepts of definitions to full ontologies. One of the best studied dependencies are specialisation dependencies [31]. For instance, an organisational definition of a particular task (the entity) can have a specialisation dependency with a task template (its context). The constraint in this case is that each organisational definition must be a specialisation of the template.

In this article, we give a non-exhaustive analysis of context dependency types and meaning conflicts between diverging meanings as a natural consequence of interorganisational ontology engineering. We illustrate these dependencies by formally describing and decomposing the context-driven macro-processes (lexical disambiguation, contextualisation, alignment, and versioning) in terms of a non-exhaustive set of micro-process primitives for selecting, linking, and changing knowledge elements.

When managed consistently, tracing context dependencies by means of micro-process primitives, provides a better understanding of the whereabouts of knowledge elements in ontologies, and consequently makes negotiation and application less vulnerable to ambiguity, hence more practical. Therefore, we outline a context dependency management framework combining these macro-processes and micro-process primitives.

1.4 Towards a Formal Framework

To formalise the context dependency management framework we circumscribed in previous subsection, we adopt and extend the DOGMA² ontology engineering approach. This approach has some distinguishing characteristics such as its groundings in the linguistic representations of knowledge, and the explicit separation of conceptualisation and axiomatisation. The DOGMA approach is supported by DOGMA Server, an ontology library system [32], that already features context-driven disambiguation of lexical labels into concept definitions [16]. Provided this basis, it is convenient to extend the DOGMA framework with a layer for managing multiple context dependency types and operators, viz. a context dependency management framework.

For the formalisation of this layer, we reuse existing domain-independent frameworks for managing diverging and converging formal artefacts, and the different types of conflicts between knowledge elements that emerge from this.

As mentioned, we focus on the positioning, not on the implementation of context-driven OE processes. Moreover, we stress the importance of human understanding and interaction during the disambiguation and conflict resolution process. Worthwhile mentioning is that some of the surveyed conflict management techniques tackle this problem rather differently from classical ontology integration techniques. However, if positioned properly, they can contribute to the ontology engineering state-of-the art.

This article is structured as follows: in Sect. 2, we introduce the DOGMA OE framework, along with its extension to support context-driven term disambiguation. Next, in

² Acronym for Developing Ontology-Grounded Methods and Applications; a research initiative of VUB STARLab.

Sect. 3, we formalise a context dependency management framework and suggest possible approaches. Then, in Sect. 4, we illustrate our framework by considering inter-organisational context dependency management in a real-world case study. Finally we end the article with a discussion in Sect. 5, and a conclusion in Sect. 6.

2 DOGMA Ontology Engineering Framework

DOGMA is an ontology approach and framework that is not restricted to a particular representation language. An important characteristic that makes it different from traditional ontology approaches is that it separates the specification of the *conceptualisation* (i.e. lexical representation of concepts and their inter-relationships) from its *axiomatisation* (i.e. semantic constraints). The goal of this separation, referred to as the *double articulation* principle [33], is to enhance the potential for re-use and design scalability.

This principle corresponds to an orthodox *model-theoretic* approach to ontology representation and development [33]. Consequently, the DOGMA framework consists of two layers: the *Lexon Base* (conceptualisation) and the *Commitment Layer* (axiomatisation).

2.1 Lexon Base

The Lexon Base is an uninterpreted, extensive and reusable pool of elementary building blocks for constructing an ontology. These building blocks (called lexons³) are linguistic in nature, and intuitively represent *plausible binary fact-types* (e.g., Person drives/is_driven_by Car). The Lexon Base is stored in an on-line DOGMA server. For guiding the ontology engineer through this very large database, *contexts* impose a meaningful grouping of these *lexons* within the Lexon Base.

The context identifier of a lexon refers to the source it was extracted from. Sources could be terminological⁴ or human domain experts. We refer to Gómez-Pérez and Manzano-Macho [35] for a comprehensive survey on text mining methods and tools for creating ontologies. For mining DOGMA lexons in particular, we refer to Reinberger and Spyns [36]. A lexon is defined as:

Definition 1 (lexon). *A lexon is an ordered 5-tuple of the form $\langle \gamma, t_1, r_1, r_2, t_2 \rangle$ where $\gamma \in \Gamma$, $t_1 \in T$, $t_2 \in T$, $r_1 \in R$ and $r_2 \in R$. Γ is a set of identifiers, T and R are sets of strings; t_1 is called the head term of the lexon and t_2 is called the tail term of the lexon; r_1 is the role of the lexon, r_2 is the co-role; γ is the context in which the lexon holds.*

Given a lexon $l = \langle \gamma, t_1, r_1, r_2, t_2 \rangle$, we define accessors as: $\gamma(l) = \gamma$, $t_1(l) = t_1$, $r_1(l) = r_1$, $r_2(l) = r_2$, $t_2(l) = t_2$, and $\forall t_i \in T \cup R : t_i \in l \Leftrightarrow t_i = t_1(l) \vee t_i = r_1(l) \vee t_i = r_2(l) \vee t_i = t_2(l)$.

³ Lexons are DOGMA knowledge elements.

⁴ “A context refers to text, information in the text, to the thing the information is about, or the possible uses of the text, the information in it or the thing itself” [34, pp. 178].

Role and co-role indicate that a lexon can be read in two directions. A lexon $\langle \gamma, t_1, r_1, r_2, t_2 \rangle$ is a fact type that might hold in a domain, expressing that within the context γ , an object of type t_1 might plausibly play the role r_1 in relation to an object of type t_2 . On the other hand, the same lexon states that within the same context γ , an object of type t_2 might play the co-role r_2 in (the same) relation to an object of type t_1 .

Some role/co-role label pairs of lexons in the Lexon Base intuitively express an *ontological relationship* (such as taxonomy, meronymy), e.g. $\langle \gamma, manager, is\ a, subsumes, person \rangle$. However, as already mentioned above: the Lexon Base is uninterpreted, so the interpretation of a role/co-role label pair as being a part-of or specialisation relation, is delegated to the Commitment Layer, where the semantic axiomatisation takes place. A lexon could be approximately considered as a combination of an RDF/OWL triple and its inverse. Lexons and commitments are visualised in a NIAM⁵-like schema (cfr. Fig. 2).

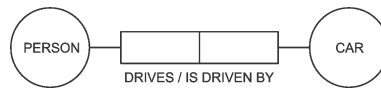


Fig. 2. Illustration of a lexon that is described in a hypothetical context γ .

2.2 Commitment Layer

Committing to the Lexon Base in the context of an application means selecting a meaningful set Σ of lexons from the Lexon Base that approximates well the intended⁶ conceptualisation, followed by the addition of a set of constraints, or rules, to this subset. We shall label these semantic constraints. The result (i.e., Σ plus a set of constraints), called an *ontological commitment*, is a logical theory of which the models are first-order interpretations that correspond to the intended meaning of the application (-domain). An ontological commitment constitutes an axiomatisation in terms of a network of lexons logically connected and provides a partial view of the Lexon Base. An important difference with the underlying Lexon Base is that commitments are internally unambiguous and semantically consistent⁷. Once elicited, ontological commitments (i.e. ontologies) are used by various applications such as information integration and mediation of heterogeneous sources. Though ontologies can differ in structure and semantics, they all are built on a shared Lexon Base.

A commitment is specified in a designated language, called Ω -RIDL [39]. It describes two aspects: (i) semantic constraints in terms of *paths*, covering all classical database constraints (cfr. ORM), and (ii) which role/co-role label pairs are interpreted

⁵ NIAM [37] is the predecessor of ORM [38].

⁶ With respect to the application domain.

⁷ Although it is outside the scope of this article, we find it valuable to note that in the research community it is debated that consistency is not necessarily a requirement for an ontology to be useful.

as which ontological relationship. Consequently, this impacts the semantics of the commitment. Commitments are also categorised and stored in a *commitment library* in the DOGMA server. Hence once applied in a commitment, a lexon declares either:

- i taxonomical relationship (*genus*): e.g., $\langle \gamma, \text{manager}, \text{is a}, \text{subsumes}, \text{person} \rangle$;
- ii non-taxonomical relationship (*differentia*):
e.g., $\langle \gamma, \text{manager}, \text{directs}, \text{directed by}, \text{company} \rangle$.

Note: a taxonomical relationship is *transitive*. This means that if an arbitrary pair $\langle \gamma, a, \text{is a}, \text{subsumes}, b \rangle, \langle \gamma, b, \text{is a}, \text{subsumes}, c \rangle \in \Sigma$, then we can assert (implicitly) $\langle \gamma, a, \text{is a}, \text{subsumes}, c \rangle \in \Sigma$. Furthermore, if $\langle \gamma, a, \text{is a}, \text{subsumes}, b \rangle, \langle \gamma, b, r_1, r_2, c \rangle \in \Sigma$ (for some arbitrary r_1, r_2), then (implicitly) $\langle \gamma, a, r_1, r_2, c \rangle \in \Sigma$.

A path differs from a lexon because it is directed, but it is trivially constructed from (a concatenation of) lexons. In the following two examples we illustrate two constraints.

Example 1. Suppose we have selected a subset Σ from the Lexon Base in order to conceptualise some domain of interest. Consider following lexons $l_i \in \Sigma$:

- $l_1 := \langle \text{EUVATDirective}, \text{company}, \text{publishes}, \text{published by}, \text{webpage} \rangle$;
- $l_2 := \langle \text{EUVATDirective}, \text{company}, \text{is referred by}, \text{refers to}, \text{name} \rangle$;
- $l_3 := \langle \text{EUVATDirective}, \text{company}, \text{is located in}, \text{locates}, \text{country} \rangle$.

Suppose we want to express that a company might publish *at most one* webpage. This is done by imposing the uniqueness constraint *UNIQ* on the path $p_1 = [\text{EUVATDirective}, \text{publishes}, \text{published by}, \text{company}]$: $UNIQ(p_1)$.

Example 2. In order to express that a company is identified by the combination of the *name it is referred by*, and the *country it is located in* we state another uniqueness constraint $UNIQ(p_2, p_3)$ in terms of two paths: $p_2 = [\text{EUVATDirective}, \text{name}, \text{refers to}, \text{is referred by}, \text{company}]$ and $p_3 = [\text{EUVATDirective}, \text{country}, \text{locates}, \text{is located in}, \text{company}]$. Another type of constraint we will illustrate is the mandatory constraint *MAND*. Suppose we want to express that a country locates *at least one* company, we state $MAND(p_3)$.

In our next definition of ontology we only consider, as proof of concept, the two constraint types *MAND* and *UNIQ* from Ex. 1 and 2, and the taxonomical and meronymical ontological relationships, resp. *isa* and *part_of*. We do this by defining a restricted ontological commitment as follows:

Definition 2 (ontology). A $(isa, partof, UNIQ, MAND)$ -restricted ontology or ontological commitment, is a tuple $\langle \Sigma, \mathcal{A} \rangle$, where Σ is a strict subset of the Lexon Base, and $\mathcal{A} = \{isa, partof, UNIQ, MAND\}$ is a particular subset or class of constraints or rules. Where $isa, partof \in R \times R$ are role/co-role label pairs that are interpreted as respectively taxonomical and meronymical ontological relationships. Furthermore, each constraint (*UNIQ*, *MAND*) is expressed as a collection of sets of paths in Σ .

Example 3. Reconsider the ontology from Ex. 1 and 2, viz. $O = \langle \Sigma, \mathcal{A} \rangle$ where p_1, p_2, p_3 are paths constructed from lexons $l_1, l_2, l_3 \in \Sigma$. Furthermore, $UNIQ, MAND \in \mathcal{A}$, where $UNIQ = \{\{p_1\}, \{p_2, p_3\}\}$ and $MAND = \{\{p_3\}\}$.

2.3 Contexts

A lexon is a lexical representation of a conceptual relationship between two concepts, however, there is no bijective mapping between a lexical representation and a concept. Consider for example phenomena such as synonyms and homonyms that can result in frustrating misunderstanding and ambiguity (see Def. 5). As we have seen, the meaning of a lexical term can vary depending on the context it was elicited from.

In DOGMA, a context is used to group lexons that are related⁸ to each other in the conceptualisation of a domain. A context in DOGMA has one fundamental property: it is also a mapping function used to disambiguate terms by making them language-neutral. Based on Meersman [15], we can give the following definition for a context:

Definition 3 (context). *A context $\gamma \in \Gamma$ is a mapping $\gamma : T \cup R \rightarrow C$ from the set of terms and roles to the set of concept identifiers C in the domain. In a context, every term or role is intuitively mapped to at most one concept identifier. A context γ is also a reference to one or more documents and/or parts of a document. This reference is defined by the mapping $cd : \Gamma \rightarrow \mathcal{D}$.*

The intuition that a context provides here is: a context is an abstract identifier that refers to implicit and tacit assumptions in a domain, and that maps a term to its intended meaning (i.e. concept identifier) within these assumptions. Notice that a context in our approach is not explicit formal knowledge. In practice, we define a context by referring to a source (e.g., a set of documents, laws and regulations, informal description of best practice, etc.), which, by *human understanding*, is assumed to “contain” the necessary assumptions [40]. The formal account for context is manifested through the interpretation of lexons in commitments, and the context dependencies between them, which we will introduce later in Sect. 3.

A tuple $\langle \gamma, t \rangle$ ideally maps to only one concept identifier. However, during the initial stage of elicitation, when lack of agreement is not often occurs, it could map to a set of concepts that approximately frames the intended one. We elaborate more on this problem in Sect. 2.5.

With a concept we mean the thing itself to which we refer by means of a term (or role) in the Lexon Base. If we want to describe the set of concepts of our domain formally, we can do this, according to Meersman [15], by introducing the partial function $ct : \Gamma \times T \cup R \rightarrow C$ which associates a concept with a tuple consisting of a context and a term (or role). This partial function, which describes a form of *meaning articulation*, is defined as follows:

Definition 4 (meaning articulation). *Given the partial function $ct : \Gamma \times T \cup R \rightarrow C$, then*

$$ct(\gamma, t) = c \Leftrightarrow \gamma(t) = c.$$

An association $ct(\gamma, t) = c$ is called the “meaning articulation” or articulation⁹ of a term t (in a particular context γ) into a concept identifier c . ct is called a meaning articulation mapping.

⁸ Not necessarily in a logical way but more in an informal way. E.g., lexons are related because they were elicited from the same source, i.e. the elicitation context.

⁹ We adopt the term articulation from Mitra et al. ([41]) (see discussion).

Our definition above includes the most general case where roles are treated like terms, hence we provide the possibility to define meaning articulations for roles as well. In some cases such as task or process ontologies (e.g., the task templates we will consider in Sect.4), it might be useful to clearly disambiguate roles, and even define cross-contextual bridges between roles. However, we must note that in practice it is usually less straightforward or even infeasible to disambiguate roles as properly as terms. Example 4 illustrates the latter definition:

Example 4. Consider a term “capital”. If this term was elicited from a typewriter manual, it has a different meaning than when elicited from a book on marketing. Therefore, we have resp. two contexts: $\gamma_1 = \text{typewriter manual}$, and $\gamma_2 = \text{marketing book}$. To express that “capital” is associated with different meanings, we write $ct(\gamma_1, \text{capital}) = c_1$, and $ct(\gamma_2, \text{capital}) = c_2$.

Until now, the endpoint of the meaning articulation is a meaningless concept identifier $c_1, c_2 \in C$. However, in the next section we will introduce the Concept Definition Server. Each concept identifier itself will point to a particular concept definition. The terms (on the *language level*) that are articulated (using ct) are then mapped to a particular *explication* of a meaning, i.e. a concept definition of a term residing in the Concept Definition Server (on the *conceptual level*), instead of to a meaningless concept identifier. Before we continue, we present some useful terminology about synonyms and homonyms (polysemy), as defined by De Bo and Spyns [42]:

Definition 5 (synonyms and polysemous terms).

- Two terms $t_1 \in T$ and $t_2 \in T$ are synonyms within a context γ if and only if $(\gamma(t_1) = c \Leftrightarrow \gamma(t_2) = c)$.
- A term $t \in T$ is called polysemous if and only if $\exists \gamma_1, \gamma_2 \in \Gamma : \gamma_1(t) \neq \gamma_2(t)$.

These definitions also hold for roles $r \in R$.

2.4 Concept Definition Server

The idea for a Concept Definition Server (CDS) was first mentioned in [42], and is based on the structure of WordNet [43]. CDS is a database in which one can query with a term, and get a set of different meanings or *concept definitions* (called *senses* in Wordnet) for that term. A concept definition is unambiguously explicated by a gloss (i.e. a natural language (NL) description) and a set of synonymous terms. Consequently we identify each concept definition in the CDS with a concept identifier $c \in C$.

The following definition specifies the CDS:

Definition 6 (concept definition server). We define a Concept Definition Server \mathcal{Y} as a triple $\langle T_{\mathcal{Y}}, \mathcal{D}_{\mathcal{Y}}, \text{concept} \rangle$ where:

- $T_{\mathcal{Y}}$ is a non-empty finite set of strings (terms) ¹⁰;

¹⁰ Additionally, we could require $T \cup R \subseteq T_{\mathcal{Y}}$ (T and R from the Lexon Base). Doing so, we require each term and role in the Lexon Base to be a term in the synset of at least one concept definition.

- $\mathcal{D}_{\mathcal{T}}$ is a non-empty finite document corpus;
- $\text{concept} : C \mapsto \mathcal{D}_{\mathcal{T}} \times \wp(T_{\mathcal{T}})$ is an injective mapping between concept identifiers $c \in C$ and concept definitions.

Further, we define $\text{conceptdef}(t)$

$$= \{\text{concept}(c) \mid \text{concept}(c) = \langle g, sy \rangle \wedge t \in sy\},$$

where gloss $g \in \mathcal{D}_{\mathcal{T}}$ and synset $sy \subseteq T_{\mathcal{T}}$.

Going from the language level to the conceptual level corresponds to articulating lexons into meta-lexons:

Definition 7 (meta-lexon). Given a lexon $l := \langle \gamma, t_1, r_1, r_2, t_2 \rangle$, and an instance of an articulation mapping $ct : \Gamma \times T \cup R \rightarrow C$. A meta-lexon $m_{l,ct} := \langle ct(\gamma, t_1), ct(\gamma, r_1), ct(\gamma, r_2), ct(\gamma, t_2) \rangle$ (on the conceptual level) is the result of “articulating” lexon l via ct .

In Fig. 3 the articulation is illustrated by a *meaning ladder* going from the (lower) language level to the (higher) conceptual level and vice-versa. We refer to Fig. 1, where we introduced the levels and the ladder in the application–elicitation setting.

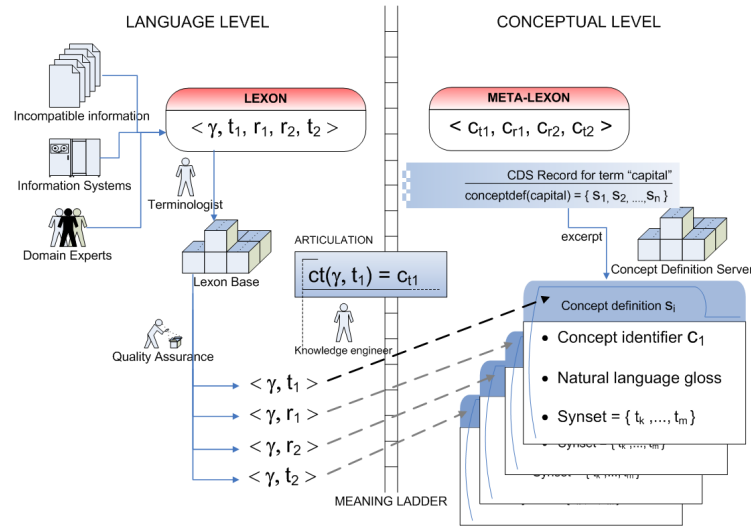


Fig. 3. Illustration of the two levels in DOGMA ontology: on the left – the lexical level, lexons are elicited from various contexts. On the right, there is the conceptual level consisting of a concept definition server. The meaning ladder in between illustrates the articulation of lexical terms into concept definitions.

Given a total articulation mapping ct , applying the articulation to the whole Lexon Base Ω would return a Meta-lexon Base $M_{\Omega,ct} = \{m_{l,ct} \mid l \in \Omega\}$.

Note: One might argue to define commitments in terms of the Meta-lexon Base, and get rid of the Lexon Base. This would translate in redefining paths in terms of meta-lexons. Doing so, however, results in a loss of the language level. The inverse articulation mapping of a meta-lexon could return more than one lexon. On the other hand by defining commitment on the Lexon Base, one can always translate to commitments on the Meta-lexon Base.

We end this section with an illustrative example:

Example 5. As an illustration of the defined concepts, consider Fig. 4. The term “capital” in two different contexts can be articulated to different concept definitions in the CDS. The terms are part of some lexons residing in the Lexon Base. The knowledge engineer first queries the CDS \mathcal{Y} for the various concept definitions of the term: $conceptdef(capital) = S_{capital} \subseteq \mathcal{D}_{\mathcal{Y}} \times \wp(T_{\mathcal{Y}})$. Next, he articulates each term to the concept identifier of the appropriate concept definition:

- Term “capital” was extracted from a typewriter manual, and is articulated to concept identifier c_1 that corresponds to concept definition (or meaning) $s_1 \in S_{capital}$ (as illustrated on the right of Fig. 4). A gloss and set of synonyms (synset) is specified for s_1 :

$$concept(ct(typewriter\ manual, capital)) = s_1.$$

- Term “capital” was extracted from a marketing book, due to the different context it was extracted from, it is articulated to another concept identifier c_2 that is associated with a concept definition $s_2 \in S$:

$$concept(ct(marketing\ book, capital)) = s_2.$$

On the other hand, suppose we have elicited a term “exercise” from the typewriter manual, and a term “example” from the marketing book. The engineers decide independently to articulate the resp. terms to the same concept definition with concept identifier c_3 with gloss: “a task performed or problem solved in order to develop skill or understanding”:

$$\begin{aligned} c_3 &= ct(typewriter\ manual, exercise) \\ &= ct(marketing\ book, example). \end{aligned}$$

This articulation defines a semantic bridge between two terms in two different ontological contexts.

2.5 Articulation and Application of Concepts in Practice

The DOGMA approach above assumes the ideal case where a tuple $\langle \gamma, t \rangle$ maps to exactly one concept identifier (hence concept definition) c . Once, after some iterations, this lexical disambiguation has been achieved, c is further ontologically organised and defined in terms of the binary relationships it has with other concepts, viz. meta-lexons.

The Meta-lexon Base consists of all *plausible* “uses” of a concept. Consequently, an application defines and constrains the genus and differentiae of each concept in its

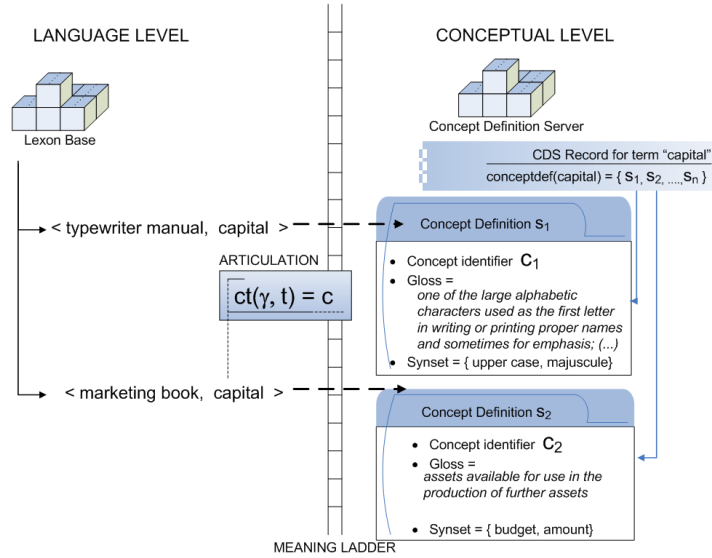


Fig. 4. Illustration of two terms (within their resp. contexts), being articulated (via the mapping ct) to their appropriate concept definition.

domain, particularly by selecting (read: committing to) a meaningful subset Σ of lexons (hence implicitly meta-lexons) that approximately fits the intended model of its concepts. Finally, a set of rules is added to constrain the possible interpretations, and hence increase the understandability and usability of the ontological commitment. Consequently, the Commitment Layer contains the (possibly empty) collections of all *committed* uses of all concepts.

The use of a concept is defined by, the axiomatised relationships it has with other concepts. Inspired by Putnam’s schematic clusters [44, pp. 33–69], we now define a query to retrieve from the Commitment Layer, the collection of all uses, given a concept:

Definition 8 (schematic cluster). Given a concept $c \in C$, the schematic cluster SC_c of c w.r.t. to a set of commitments \mathcal{O} is defined as the collection of non-empty lexon sets $\pi(O_i)$ with $O_i = \langle \Sigma_i, \mathcal{A}_i \rangle \in \mathcal{O}$, where $\pi(O_i)$ is defined as $\{l \in \Sigma_i \mid ct(\gamma(l), t_1(l)) = c \vee ct(\gamma(l), t_2(l)) = c\}$.

Concept definitions (stored in the CDS) present the essential meaning of a concept, while commitments represent the domain-specific applications of a particular concept. A similar distinction exists between respectively Aristotle and Wittgenstein: Aristotelian “type” definitions are obligatory conditions that state typical properties, while Wittgenstein considers the meaning of a concept to be the set of all its uses [45, pp. 128], the latter is analogue to Putnam’s schematic clusters above.

Brachman [46] also believes that a careful distinction must be made between essential and *incidental* properties of a concept. Only essential properties should be defined in the ontology as they are recognized as members of the type in every possible world.

From a DOGMA point of view, non-essential properties can always reside in the Metalexon Base, but whether they are committed to, is incidental to the application scenario. If a property is de facto always committed to, it actually becomes essential.

Based on Brachman's ideas, Bachimont [26] claims that organising the domain concepts in a taxonomy is a key component for building ontological commitments. In this process, he emphasizes the importance of a clear *normalisation* (or lexical disambiguation) of the meaning of concepts. Once terms are normalised properly, they can be *formalised* (hence selected and committed to) and *operationalised*. For the normalisation, a differential ontology is built which turns lexical labels into *notions*, based on differential semantics [47]. Practically, this means that notions are orthogonally described in terms of similarities and differences with their parents, children, and siblings. In the resulting taxonomy of notions, the meaning of a node is given by the gathering of all similarities and differences attached to the notions found on the way from the root notion (the more general) to this node.

Summarising, the disambiguation of terms extracted from verbal descriptions is clearly a methodological process of expressing the differences and similarities with other concept definitions using notions of context. In this process it is important to distinguish between necessary and incidental properties. In practice, in a typical ontology elicitation scenario where multiple stakeholders yield tacit and imperfect definitions, ontological definitions will continuously be subject to changing relevance and focus, and the distinction between essential or incidental will evolve as well in some cases, even within the same application domain. Agreeing on one unique concept definition to disambiguate a term, is an incremental process that should result in a right balance where essential properties are comprised in the concept definition, and incidental properties are put rather in commitments depending on the relevance for the respective stakeholders. This enforces our argument to extend our traditional ontology library system, in order to support these kind of OE processes in the management of context dependencies and conflicts between knowledge elements from different ontologies.

3 Context Dependency Management

The management of context dependencies is important for successful disambiguation in OE. Ding and Fensel [32] identify management as one of the three indispensable pillars for an ontology library system. They define management in terms of three features, viz. ontology identification, storage and versioning. We must note that in the collaborative application we envision, we cannot rely on pessimistic versioning control, where all ontologists work on the same library, and parallel contextualisation is prevented by locking. However, if we want ontologists to be able to work completely separately for some time on a personal copy of the ontology (optimistic versioning), we have to deal with the conflicts between parallel contextualisations.

3.1 Formal Framework

We identify the following features in our context dependency management framework:

1. a library of *ontologies*;

2. a sound and complete set of context dependency *operators* on ontologies;
3. a collection of *context dependency types*;
4. a library of *contextualisations*, i.e context dependency relations within between ontologies in the library.

A library of ontologies The Commitment Layer is a library of ontological commitments. DOGMA Server stores these in a RDBMS, and provides an API for unified access to all structures.

A set of context dependency operators An additional element of the framework is a sound and complete set of *context dependency operators*. The set should subsume every possible type of ontology access and manipulation (completeness issue), and in particular, the manipulation operators should only generate valid ontologies (soundness issue) [48]. In this article we only give a non-exhaustive set of operators. However, concerning the soundness issue, we make sure these operators are conditional, which means that their applicability depends on pre- and post-conditions.

The resemblance and differences between ontologies and data schema are widely discussed in literature such as [15, 33, 49]. In the schema and ontology evolution literature, much work focuses on devising taxonomies of elementary change operators that are sound and complete. Significant examples of data schema evolution include *transformation rules* (in terms of pre- and post-conditions) to effect change operators on data schemas and change propagation to the data [48], frameworks for managing multiple *versions of data schemas* coherently [50, 51] and models for different levels of granularity in change operators, viz. *compound change operators*¹¹ [52]. Furthermore, changes in one facet of an ontology might trigger a cascade of changes in other facets [53]. These triggers could be defined by the semantics of the dependencies.

Main results in ontology evolution include [54, 55, 29, 56], which base their work predominantly in the previous mentioned schema evolution techniques, next to addressing particular needs for evolution of ontologies.

We provide a non-exhaustive list of operators for supporting OE and characterising context dependencies. For now, we ignore axiom operators. We define the pre- and postconditions in terms of the ontology before the operation, denoted by $L = \langle \Sigma_L, \mathcal{A}_L \rangle$, and the ontology after the operation, denoted by $R = \langle \Sigma_R, \mathcal{A}_R \rangle$. Also $isa = \{r_{isa1}, r_{isa2}\} \in \mathcal{A}_{\{L,R\}}$ ¹².

1. *artConcept*($\langle \gamma, t \rangle, c$): articulate a term t in a particular context γ into a concept $c \in C$, provided $ct(\gamma, t)$ is not defined.
 $\forall t \in T; \forall \gamma \in \Gamma; \forall c \in C :$
 - *PreCond* = $\{\langle \gamma, t \rangle \in \Sigma_L; ct(\gamma, t) \text{ undefined}\}$
 - *PostCond* = $\{\langle \gamma, t \rangle \in \Sigma_R; ct(\gamma, t) = c\}$
2. *defineGenus*($\langle \gamma, t \rangle, \langle \gamma, g \rangle$): set $ct(\gamma, g)$ as genus of a concept $ct(\gamma, t)$.
 $\forall t, g \in T; \forall \gamma \in \Gamma:$

¹¹ e.g., moving an attribute x from a class A to a class B , means (more than) successively deleting x in A and adding x in B .

¹² cf. Definition 2.

- $PreCond = \{\langle \gamma, t \rangle \in \Sigma_L, \langle \gamma, g \rangle \in \Sigma_L, \langle \gamma, g, r_{isa1}, r_{isa2}, t \rangle \notin \Sigma_L\}$
 - $PostCond = \{\langle \gamma, t, r_{isa1}, r_{isa2}, g \rangle \in \Sigma_R\}$
3. *defineDiff*($\langle \gamma, t \rangle, \bigcup_i d_i$): add a set of differentiae $\bigcup_i d_i$, where $d_i = \langle \gamma, t, r_1^{d_i}, r_2^{d_i}, t_2^{d_i} \rangle$, to an already defined concept $ct(\gamma, t)$.
 $\forall t, t_2^{d_i} \in T; \forall r_1^{d_i}, r_2^{d_i} \in R; \forall \gamma \in \Gamma$:
 - $PreCond = \{\langle \gamma, t \rangle \in \Sigma_L; \forall_i \langle \gamma, t_2^{d_i} \rangle \in \Sigma_L; \forall_i d_i \notin \Sigma_L\}$
 - $PostCond = \{\langle \gamma, t \rangle \in \Sigma_R; \forall_i \langle \gamma, t_2^{d_i} \rangle \in \Sigma_R; \forall_i d_i \in \Sigma_R\}$
 4. *specialiseDiff*($\langle \gamma, t \rangle, \langle \gamma, s \rangle, d$): replace any occurrence of $\langle \gamma, t \rangle$ with one of its children $\langle \gamma, s \rangle$ in an existing differentia $d = \langle \gamma, t, r_1, r_2, t_2 \rangle$.
 $\forall t, s, t_2 \in T; \forall r_1, r_2 \in R; \forall \gamma \in \Gamma$:
 - $PreCond = \{\langle \gamma, t, r_1, r_2, t_2 \rangle, \langle \gamma, s, r_{isa1}, r_{isa2}, t \rangle \in \Sigma_L; \langle \gamma, s, r_1, r_2, t_2 \rangle \notin \Sigma_L\}$
 - $PostCond = \{\langle \gamma, s, r_1, r_2, t_2 \rangle, \langle \gamma, s, r_{isa1}, r_{isa2}, t \rangle \in \Sigma_R; \langle \gamma, t, r_1, r_2, t_2 \rangle \notin \Sigma_R\}$
 5. *generaliseDiff*($\langle \gamma, t \rangle, \langle \gamma, p \rangle, d$): replace any occurrence of $\langle \gamma, t \rangle$ with a parent $\langle \gamma, p \rangle$ in an existing differentia $d = \langle \gamma, t, r_1, r_2, t_2 \rangle$.
 $\forall t, p, t_2 \in T; \forall r_1, r_2 \in R; \forall \gamma \in \Gamma$:
 - $PreCond = \{\langle \gamma, t, r_1, r_2, t_2 \rangle, \langle \gamma, t, r_{isa1}, r_{isa2}, p \rangle \in \Sigma_L; \langle \gamma, p, r_1, r_2, t_2 \rangle \notin \Sigma_L\}$
 - $PostCond = \{\langle \gamma, p, r_1, r_2, t_2 \rangle, \langle \gamma, t, r_{isa1}, r_{isa2}, p \rangle \in \Sigma_R; \langle \gamma, t, r_1, r_2, t_2 \rangle \notin \Sigma_R\}$
 6. *pullUp*($\langle \gamma, t \rangle, \langle \gamma, s \rangle$): pull up an already defined concept (including its children) higher in the taxonomy as a child of $\langle \gamma, s \rangle$.
 $\forall t, t_1, s \in T; \forall \gamma \in \Gamma$:
 - $PreCond = \{\langle \gamma, t, r_{isa1}, r_{isa2}, t_1 \rangle \in \Sigma_L; \langle \gamma, t, r_{isa1}, r_{isa2}, s \rangle \notin \Sigma_L\}$
 - $PostCond = \{\langle \gamma, t, r_{isa1}, r_{isa2}, t_1 \rangle \notin \Sigma_R; \langle \gamma, t, r_{isa1}, r_{isa2}, s \rangle \in \Sigma_R\}$
 7. *pullDown*($\langle \gamma, t \rangle, \langle \gamma, s \rangle$): pull down an already defined concept (including its children) lower in the taxonomy as a child of $\langle \gamma, s \rangle$.
 $\forall t, t_1, s \in T; \forall \gamma \in \Gamma$:
 - $PreCond = \{\langle \gamma, t, r_{isa1}, r_{isa2}, t_1 \rangle \in \Sigma_L; \langle \gamma, t, r_{isa1}, r_{isa2}, s \rangle \notin \Sigma_L\}$
 - $PostCond = \{\langle \gamma, t, r_{isa1}, r_{isa2}, t_1 \rangle \notin \Sigma_R; \langle \gamma, t, r_{isa1}, r_{isa2}, s \rangle \in \Sigma_R\}$

The latter two in fact produce resolution points as there are alternative strategies to resolve the inheritance of differentiae when pulling up or down a concept. As this discussion is outside the scope of this article we take the most straightforward resolution and omit further elaboration on this.

A set of context dependency types How a new ontology was obtained from the original is determined by a sequence of applied operators. By constraining the possible combinations of operators, we can characterise various types of dependency between the new and the old. We refer to these dependency types by inferring a class of dependency types.

An ontology that is context-dependent on another ontology is called a contextualisation. The contextualisation of ontological definitions might be constrained in different

ways. One particular example in the sense of conceptual graph theory [45] would be a specialisation dependency for which the dependency constraint is equivalent to the conditions for CG-specialisation [45, pp. 97]. A specialisation dependency corresponds to a monotone specialisation.

We can identify and express different context dependency types between sub-contexts within one ontology (*intra-ontological*) and between different ontologies (*inter-ontological*): (i) articulation; (ii) application; (iii) specialisation; and (iv) revision. As suggested by discussions, there are many more context dependency types. However, we will only formalise the above four in this article, and characterise them in terms of applicable operators.

articulation dependency Usually, lexical labels are used to represent concepts and relations, though there is no bijective mapping between them. Phenomena such as synonyms and homonyms exemplify this, and can result in frustrating misunderstanding and ambiguity when unifying information from multiple sources. The meaning of a concept behind a lexical term is *dependent* on the *context of elicitation*. This ambiguity might be partly resolved by taxonomically categorizing it by setting its genus, however this should be complemented by explicating the *articulation* of a lexical term in a particular context into the intended concept. Hence this dependency type, denoted by *ART*, is characterised by *artConcept* and *defineGenus*.

application dependency Once a term has been articulated into the intended concept and its genus has been set, it is used or applied in multiple ways. Within a particular context of application, a concept is defined in terms of differentiae and axioms, resulting in an ontological definition. An application dependency corresponds to a monotone refinement, consequently this dependency type, denoted by *APP*, is characterised by *defineDiff*.

specialisation dependency The specialisation of ontological definitions is typically constrained to refining differentiae, including specialising concepts. Hence this dependency type, denoted by *SPE*, is characterised by *specialiseDiff*.

revision dependency When considering versioning, evolution steps are stored for later reference, therefore we devise a revision dependency. A revision might be *non-monotone*, i.e. it might correspond to well-formed sequence of expansions and contractions of definitions, or even reclassification in taxonomies. Hence this dependency type's particular characteristics depend on the operators used. Notice that a revision possibly triggers a cascade of revisions to all ontologies that are context-dependent on the knowledge elements in the revised part. This dependency is denoted by *REV*.

Example 6. As an illustration of defining a context dependency type in terms of constraints on possible operators, the necessary and sufficient condition to assert that O_2 is application-dependent of concept c in O_1 , is that O_2 is a monotone refinement of a subset of O_1 in terms of a sequence of exclusively *defineDiff* operations. In that case O_2 is a “concept-application” contextualisation of O_1 . We denote this as $O_1 \dashrightarrow_{APP} O_2$.

A library of contextualisations Context dependency types are instantiated in terms of a source and a target ontology from the library. These instantiations are referred to as contextualisations.

In this subsection, we defined a context dependency framework that is mainly defined by a set of context dependency operators and types. Next, we look at related approaches that might support our framework in the management of these context dependencies, and above all resolution of related meaning conflicts.

3.2 Approaches for Context Dependency Management

In this subsection we suggest some existing approaches for the management of diverging and converging formal artefacts, and the related conflicts, for supporting collaboration. Mens [57] gives an excellent survey of such techniques in the context of software merging. Among these we particularly focus on domain-independent formalisms, so they can be reused for our purposes, viz. the management of context dependencies and emerging conflicts between knowledge elements in ontologies. He considers four types of conflicts that can occur in formal artefacts: textual, syntactic, structural and semantic conflicts.

As mentioned in the introduction, we focus on the positioning, not on the implementation of context-driven OE processes. Moreover, we stress the importance of human interaction during the conflict resolution process. Worthwhile mentioning is that some of the surveyed conflict management techniques provide such facilities for resolving conflicts, which are rather different than classical ontology integration techniques [28]. However, if positioned properly, they can contribute to the ontology engineering state-of-the-art.

collaborative application frameworks These frameworks support multiple distributed users working on temporal replicas of a shared artefact, and assistance in the following merging process. Timewarp [58] uses divergent and convergent timelines to support collaboration, and provides facilities for conflict detection resolution comparable to analysis in Ex. 7. The framework is extended with multi-level or inter-leaved timelines, and support for capturing causal relationships in timelines via a nested transaction mechanism. In GINA [59], merging of command histories is achieved by selectively using a redo mechanism to apply one developer's changes to the other's version. The user is assisted by the system when merging diverged replicas. Approaches in this area concentrate especially on the user interface aspects. Therefore, they are in fact complementary to the formal foundations that conceptual graph rewriting provides.

conditional graph rewriting Mens [60] defined a domain-independent formalism for detecting and resolving merge conflicts during parallel evolutions of formal artefacts in general, and software in particular. Therefore he adopts principles of graph rewriting (GR) theory [61]. The advantage of GR is that it provides a domain-independent and stable formal foundation for both *descriptive* as *operational* aspects.

In this formalism, we can describe a possibly infinite collection of graphs in a finite way: by stating a set of initial graphs and a set of graph rewriting rules. Through

repeated application of these rules, starting from one of the initial graphs, new graphs can be generated. An asset here is that the ontology engineer does not have to explicitly specify this sequence of rules: he only needs to specify how the new ontology should look like. Another advantage of formalising ontology transformations in a conditional GR system is that we can provide a precise and unambiguous definition of context dependency notions, and make it visually attractive.

Next to the descriptive aspect, the graph rewriting rules simplify the composition of context dependency operators, hence its semantics. Furthermore GR, reduces the complexity of the manipulation and analysis during the OE processes we identified above. E.g., it provides theoretical insights into the parallel and sequential independence of ontology transformations, the latter which are described in terms of sequences of operators. This means, translated to context-driven OE, support for the detection of possible meaning conflicts between multiple parallel contextualisations of the same original ontology (see Ex. 7).

Example 7. Consider an ontology O , which is contextualised by two ontologists concurrently. The first ontologist adds a differentia to an already existing concept $\langle \gamma, t \rangle$ using $defineDiff(\langle \gamma, t \rangle, \langle \gamma, t, r_1, r_2, t_2 \rangle)$. The second ontologist concurrently defines a genus for $\langle \gamma, t \rangle$, viz. $\langle \gamma, g \rangle$, using $defineGenus(\langle \gamma, t \rangle, \langle \gamma, g \rangle)$. Now consider $\langle \gamma, g, r_1, r_2, t_2 \rangle \in \Sigma_O$. For both operations the pre-conditions w.r.t. to the O are satisfied. However, when merging both contextualisations a conflict would have merged as each operation introduces an element that is forbidden to exist in the pre-condition of the other.

The caveat for reusing Mens' formalism is the definition of a typegraph, a labelled typed (nested) graph representation of the meta-schema for the kind of formal artefact that is to be manipulated. The fact-oriented character of DOGMA ontologies and meta-schema, makes GR particularly suitable for our purposes.

4 Inter-organisational Context Dependency Management

In this section we illustrate context-driven ontology elicitation in a realistic case study of the European CODRIVE¹³ project.

4.1 Competencies and Employment

Competencies describe the skills and knowledge individuals should have in order to be fit for particular jobs. Especially in the domain of vocational education, having a central shared and commonly used competency model is becoming crucial in order to achieve the necessary level of interoperability and exchange of information, and in order to integrate and align the existing information systems of competency stakeholders like schools or public employment agencies. None of these organisations however, have successfully implemented a company-wide “competency initiative”, let alone a strategy for inter-organisational exchange of competency related information.

¹³ CODRIVE is an EU Leonardo da Vinci Project (BE/04/B/F/PP-144.339).

The CODRIVE project aims at contributing to a competency-driven vocational education by using state-of-the-art ontology methodology and infrastructure in order to develop a conceptual, shared and formal KR of competence domains. Domain partners include educational institutes and public employment organisations from various European countries. The resulting shared “Vocational Competency Ontology” will be used by all partners to build interoperable competency models.

The example concerns collaborative OE between two participating stakeholders *EI* and *PE*, each containing organisations being resp. *educational institute* and *public employment agency*. Core shared and individual organisational ontologies have already been defined for both *EI* and *PE*. Fig. 5 illustrates the initial versions, say *V0*, of the different contexts¹⁴:

- *CO* is the common ontology, containing two sub-contexts, viz. *CO_TMPL* and *CO_TH*. The latter is *CO*'s type hierarchy (detailed in Fig. 6). It has among its concepts *Task*, with as subtypes *EducationalTask* and *JobTask*. The concepts are in fact terms, but within the context *CO* they refer to at most one concept definition. *CO_TMPL* is a set of templates that define applications of concepts in *CO_TH* in terms of differentiae, consequently *CO_TMPL* is application-dependent on *CO_TH*.
- *OO_EI* and *OO_PE* are the individual organisational ontologies for resp. educational institutes and public employment agencies. Organisational ontologies also contain two sub-contexts: e.g., *OO_EI_TH* is *EI*'s type hierarchy whose concepts are articulation-dependent on *CO_TH*. *OO_EI_DEFS* contains *EI*'s ontological definitions refining the templates in *CO_TMPL*, hence *OO_EI_DEFS* is application-dependent on *CO_TMPL*.

The different contextualisations above, illustrate the *grounding* of the individual organisational definitions in the common ontology *CO*.

In the following subsections we consider a scenario where we show that in order to effectively and efficiently define shared relevant ontological meanings, context dependencies are indispensable. In building the shared ontology, the individual organisational ontologies of the various stakeholders need to be aligned *insofar necessary*. It is important to realise that costly alignment efforts only should be made when necessary for the shared collaboration purpose.

The scenario illustrates the following processes by different stakeholders, necessary in inter-organisational ontology engineering: (i) term disambiguation in *OO_EI* and *OO_PE*; (ii) template creation in *CO*; (iii) template specialisation *OO_EI*; and finally (iv) template revision, triggering a cascade of revisions to context-dependent sub-contexts. These processes are triggered by business rules tailored to the specific requirements of a collaborative community. Even with a few simple operators, already many context dependencies are introduced.

¹⁴ In this case study, each context corresponds to exactly one ontology and vice-versa. However, an ontology engineer might select lexons from various contexts for modelling his ontology.

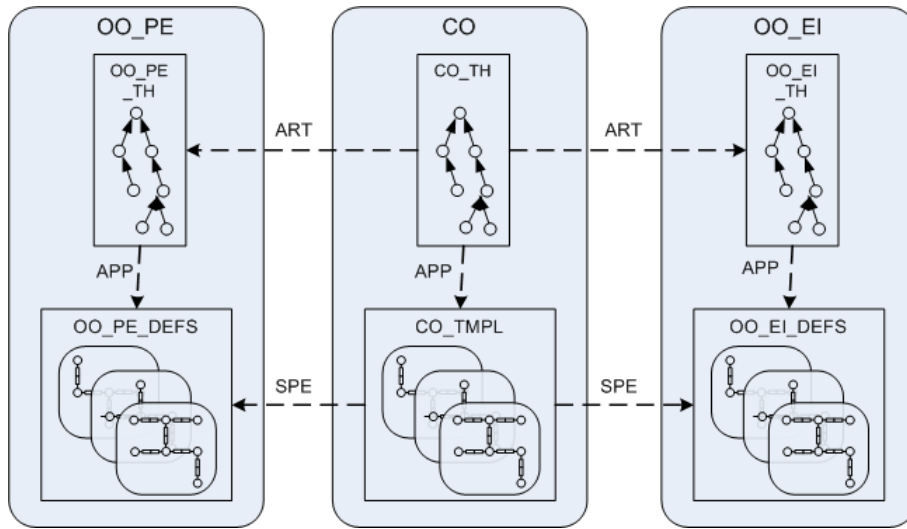


Fig. 5. The scenario illustrates a snapshot of contexts and context dependencies emerging from different OE processes by multiple stakeholders in inter-organisational ontology engineering: (i) term articulation (*ART*); (ii) template creation (*APP*); and (iii) template specialisation (*SPE*).

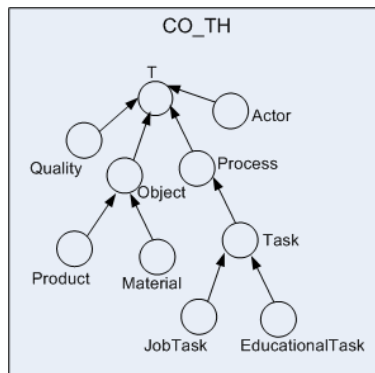


Fig. 6. Excerpt from a sub-context *CO.TH* in *CO*. T denotes the root concept.

4.2 Term Disambiguation in *OO_EI*

EI is a national bakery institute, responsible for defining curriculum standards for bakery courses. As such, much of its organisational terminology should be grounded in the common ontology *CO*. It now wants to add a new term “panning” to its *OO_EI* ontology. It defines this informally as the act of “depositing moulded dough pieces into baking pans with their seam facing down”.

First the ontology maintainer defines the term’s genus (*defineGenus*) and articulation (*artConcept*) in a sub-context *OO_EI_TH* of *OO_EI*. *OO_EI_TH* is supposed to be articulation-dependent on *CO_TH*, so *EI* is allowed to define and articulate the new term in its ontology. However, in this *EI* case, the precondition is that the genus is predefined in *CO_TH* (Fig. 6). This is inferred from the production *defineGenus* defined in Sect. 3 strengthened by the following business rule¹⁵:

R1: The CODRIVE ontology server (COS) asks *EI* to classify the shared concept to which the term belongs.

Hence, The *EI* representative classifies *Panning* as an *EducationalTask* in taxonomy *OO_EI_TH*.

Next, *EI* decides to articulate *Panning* as follows:

$$\text{concept}(ct(OO_EI_TH, \text{panning})) = \langle g, sy \rangle$$

where gloss *g* corresponds to “depositing moulded dough pieces into baking pans with their seam facing down”. The synset *sy* is ignored here. The above operations are applications of resp. *defineGenus* and *artConcept* which results in following articulation-contextualisation:

$$CO_TH \dashrightarrow_{ART} OO_EI_TH.$$

4.3 Template Creation in *CO*

One of the business rules in the CoDrive ontology server demands:

R2: IF a NewTask is an EducationalTask, and the IndividualOntologyOwner is an EducationalInstitute THEN a FullSemanticAnalysis of the NewTask needs to be added to the IndividualOntology of the IndividualOntologyOwner;
another meta-rule fires as an immediate consequence:

R3: IF a FullSemanticAnalysis needs to be made of a Concept in an IndividualOntology or SharedOntology THEN the ConceptTemplate needs to be filled out in that Ontology. Furthermore, for each Term and Role of these definitions, a MeaningArticulation needs to be defined.

The template was created in a separate process by the common ontology maintainer. Creating a template corresponds to applying a sequence of *defineDiff* operations in order to define differentiae of a given concept. This results in an application dependency (*APP* arrows in Fig. 5):

$$CO_TH \dashrightarrow_{APP} CO_TMPL. \tag{1}$$

¹⁵ The concepts underlined in the rules below are modelled but not shown.

Hence, it resides in a sub-context of *CO*, viz. *CO_TMPL*. The template is shown in Fig. 7. In this case the template states it is necessary to know who is the performer

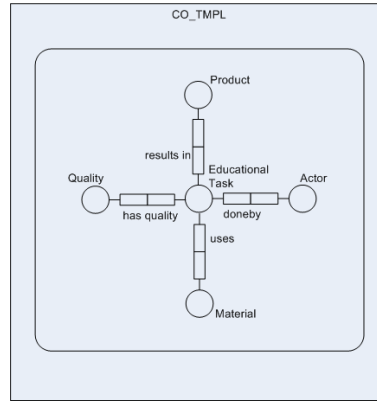


Fig. 7. Template for *Educational Task*, defined in *CO_TMPL*, sub-context of *CO*.

of the task (e.g. *Student*), what inputs and materials are necessary for the task (e.g. *Baking_Pan*, *Dough*), what is the result of the task (*Greased Pan*), and so on.

4.4 Template Specialisation in *OO_EI*

A template is an ontological definition that needs to be specialised using the specialisation dependency (*SPE* arrows in Fig. 5).

In *OO_EI_DEFS* (a sub-context of *OO_EI*), the new task *Panning* is specialised, which boils down to specialising (and in a way disambiguating) the differentiae in the template (illustrated by Fig. 7). Each new term or role to be used in the specialisation must first be defined and articulated in *OO_EI_TH*. Similar to before, this produces new operations that extend our articulation contextualisation:

$$CO_TH \dashrightarrow_{ART} OO_EI_TH. \quad (2)$$

The specialisation of the template is done by iteratively applying the *specialiseDiff* production which results in following specialisation-contextualisation:

$$CO_TMPL \dashrightarrow_{SPE} OO_EI_DEFS. \quad (3)$$

4.5 Term Disambiguation in *OO_PE*

Concurrently with **R3**, following business rule is triggered:

R4: IF an EducationalTask is added to an IndividualOntology THEN a corresponding JobTask needs to be defined in all instances of IndividualOntology of all PublicEmploymentAgencies

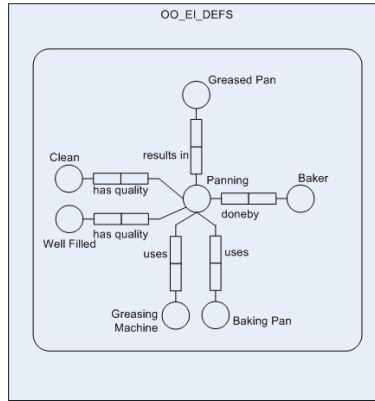


Fig. 8. A specialisation-contextualisation $CO_TMPL \dashrightarrow_{SPE} OO_EI_DEFS$.

The rationale for this rule is that public employment agencies need to be aware of changes to the curricula of educational institutes, so that they are better able to match job seekers with industry demands. However, unlike the definitions of educational tasks, the job task definitions in public employment agency ontologies only require a short informal description of the concept itself, not an extended template definition:

R5: IF a JobTask is added to an IndividualOntology THEN a Gloss needs to be defined for that Concept.

Of course, public employment agencies also could have the need for template definitions, but those would refer to the job matching processes in which the tasks play a role (*why* is panning needed), not to *how* the tasks themselves are to be performed.

Hence, **R4** requires the PE representative to classify *Panning* as an *JobTask* in *OO_PE_TH*.

Next, he decides to articulate *Panning* as follows:

$$concept(ct(OO_PE_TH, panning)) = \langle g, sy \rangle$$

where gloss *g* corresponds to “an essential skill in baking”. The synset *sy* is ignored here.

Again, the above operations are applications of resp. *defineGenus* and *artConcept* which results in following articulation-contextualisation:

$$CO_TH \dashrightarrow_{ART} OO_PE_TH.$$

4.6 Template Revision in *CO_TMPL*

Figure 5 shows the version 0 snapshot of a library of ontologies and their context dependencies, the latter denoted by arrows in different directions. When considering the revision dependency type, we must introduce a timeline that is orthogonal to all arrows so far. A revision to a knowledge element in some ontology should result in a new version of that ontology. As a consequence, a cascade of revision requests is triggered to

all ontologies that are context-dependent on the revised element. This in order to keep the context dependencies intact, and hence anticipate on conflicts. In the following example, we will show that, as an additional problem, in most cases there are multiple alternative ways to resolve the conflict. Our framework identifies conflicts, hence ambiguity, and delegates these decision options to the human ontologist.

Suppose that the template we created (cf. Fig. 7) is expanded with a new differentia. First a new concept *LectureRoom* is defined and then articulated in *CO_TH*:

$$\begin{aligned} & \text{defineGenus}(\langle CO, \text{LectureRoom} \rangle, \langle CO, \text{Object} \rangle) \\ & \text{artConcept}(\langle CO, \text{LectureRoom} \rangle, c) \end{aligned}$$

for some $c \in C$. This results in the following revision-contextualisation:

$$CO_TH \dashrightarrow_{REV} CO_TH_V1.$$

Note the version-id that is appended, which brings *CO_TH_V1* to a time snapshot different from all other ontologies so far. Next, we define a new differentia in *CO_TMPL* using the new concept in *CO_TH_V1*:

$$\begin{aligned} & \text{defineDiff}(\langle CO, \text{EducationalTask} \rangle, \\ & \quad \langle CO, \text{EducationalTask}, \text{has, of}, \text{LectureRoom} \rangle). \end{aligned}$$

This results in another revision-contextualisation:

$$CO_TMPL \dashrightarrow_{REV} CO_TMPL_V1,$$

constrained by following application dependency:

$$CO_TH_V1 \dashrightarrow_{APP} CO_TMPL_V1.$$

This application dependency itself can be considered as an extension of the previously created dependency in (1).

The latter revision, forces us to evolve all ontologies that are context-dependent on *CO_TMPL*. As an illustration, take the specialisation-contextualisation

$$CO_TMPL \dashrightarrow_{SPE} OO_EI_DEFS$$

in Fig. 8. Because the template has been expanded, this template-specialisation has now become underspecified: first, it tells nothing about some *LectureRoom* room in which *Panning* is being instructed; and secondly, it doesn't specify in which particular type of *LectureRoom* that *Panning* is being instructed.

Hence, rule **R3** is triggered again. This rule required that templates should be fully specified. The process that follows is analogue to the template specialisation that we described earlier, so we skip the details. In short, in order to specialise the newly introduced differentia in the template, *LectureRoom* is specialised by introducing a new subconcept *PreparationRoom* in *OO_EI_TH*. This results in following revision-contextualisations:

$$OO_EI_TH \dashrightarrow_{REV} OO_EI_TH_V1,$$

$$OO_EI_DEFS \dashrightarrow_{REV} OO_EI_DEFS_V1,$$

again constrained by following articulation and specialisation dependencies respectively:

$$CO_TH_V1 \dashrightarrow_{ART} OO_EI_TH_V1,$$

$$CO_TMPL_V1 \dashrightarrow_{SPE} OO_EI_DEFS_V1,$$

being extensions of previously created dependencies in (2) and (3) respectively.

5 Discussion

The previous scenario, although using only a few simple operators, already demonstrated the complex dependencies emerging in a typical case of context-driven ontology engineering. When introducing additional context dependency operators, complexity only grows. Furthermore, we have only considered relatively straightforward expansion operators. Contraction even makes context dependency management more difficult. Especially then there is special need for conflict pair analysis (cf. Ex. 7), which we did not touch upon in the scenario.

Some further observations:

1. Context-driven ontology elicitation and disambiguation avoids wasting valuable modelling time and enormous cost. This is illustrated by the density of lexon elicitation in the *EI_DEFS* ontology compared to the sparsely populated *PE_DEFS* for term “panning”. The density reflects the minimal level of modelling details needed. For other terms this might be vice-versa.
2. Context dependencies can be defined between sub-contexts within one ontology and between different ontologies. Their types are characterised by applicable operators. This is illustrated by the arrows between the various nested boxes.
3. The context dependency arrows define a lattice. Management and access is facilitated by properly navigating through its branches. This suggests the need for an appropriate query language.
4. Some contextualisations require the pre-existence of other types of contextualisation. Reconsider the specialisation of *CO_TMPL* which required a number of applications of articulations first.

It is clear that proper management of these dependency complexities is essential for context-driven ontology engineering to be successful. In this article, we have presented the foundation of an extensible context dependency management framework. It currently contains a number of important context dependency types and operators. In future research, we will produce a more detailed typology.

Next some more reflections on related work and future research.

5.1 Lexical Disambiguation

Shamsfard et al. [62] provide a comprehensive survey of methods and tools for (semi) automatic ontology elicitation. However, in this article our focus is not on automation.

Work which is strongly related with what we need is e.g., Mitra et al. [41], who indirectly adopt some of those features we concluded with in our synthesis earlier. They illustrate a semi-automatic tool for creating a mapping between two ontologies (in fact contexts). Their motivation is that two different terms can have the same meaning and the same term can have different meanings, which exactly defines the lexical disambiguation problem. This mapping is manifested by an *articulation ontology*, which is automatically generated from a set of *articulation rules* (i.e. semantic relationships) between concepts in each context resp. Finally, we refer to the MikroKosmos project [63], which is concentrated in the domain of ontology-based disambiguation for natural-language processing in a multi-cultural context.

In our framework, the CDS provides a basic means for relevant alignment of heterogeneous ontologies. The concept definitions (gloss, synset) in the CDS support the meaning articulation of language level terms. As was illustrated in Ex. 5, the articulation of terms from different contexts to a shared CDS, results in cross-context equivalence relations, i.e. synonyms.

The meta-rules we made in step 3 and 5, were not formalised explicitly in this article. However, we could devise a syntax, e.g.:

$$\begin{aligned} ct(EI, panning) &\preceq ct(SHARED, educational\ task) \\ ct(PE, panning) &\preceq ct(SHARED, job\ task) \end{aligned}$$

The rules above extend the cross-context equivalence: they allow us to specify an alignment between a term in one context to a possibly more general term in another context. In the future we will extend this feature and provide a formal semantics of meta-rules. This can be very powerful in context-driven ontology elicitation and application such as meta-reasoning on context and ontology alignment processes, and meaning negotiation processes between stakeholders. Currently we are exploring reasoning for commitment analysis and conceptual graph tools for ontology elicitation and analysis.

Initially, the lexon base and CDS are empty, but the CDS can be easily populated by importing similar information from publically available electronic lexical databases, such as WordNet [43] or Cyc [64]. The Lexon Base is populated during the first step of an ontology elicitation process by various (not necessarily human) agents. See Reinberger & Spyns [36] for unsupervised text mining of lexons. The second step in the elicitation process is to articulate the terms in the “learned” lexons.

5.2 Context Dependency Management

For supporting context-driven OE, we consider two points of view to manage (storage, identification, versioning [32]) the disambiguation of meaning – ontologies in particular – in our framework:

1. The context-dependency-driven way is to store only the initial ontologies and the restricted derivation sequences. Access to the contextualisations is done by navigating through the lattice composed by the context dependencies. Once arrived, the contextualisation is generated automatically by completing the pushout of the derivation sequence, the latter being gradually built from the path followed in the lattices.

2. The meta-data-driven way would be to store all the ontologies, but none of the context dependencies between them. Access to these contextualisations is by identifying them with domain-specific meta-data tags (cf. Cyc [65]). This might include authorship and space-time allocation. Other dimensions are correlations with other contexts, in our case, context dependencies. We have to investigate whether it is possible, given any pair of ontologies, to induce this information.

The context dependency types and operators we introduced in this article are not exhaustive. Currently, we have only defined expanding and specialising operators, which did not turn out to be very conflictive. However, to support complete revision capability, we must add contracting and more generalising operators of which we could expect following troublesome scenarios:

- Revisions will trigger cascades of revisions to existing context dependencies, usually implying alternative resolutions in which the human has to make choices.
- A revision resulting from a sequence of concept generalisations and specialisations could force the concept to be unnaturally reclassified in a different branch in the taxonomy.
- Introducing operators for axioms, implying new “axiomatisation” dependency types, could return unexpected behaviour. In this article we did not consider the semantic constraints completely. The only visible constraint is the interpretation of ontological relationships. We aim to extend our framework with typical ORM [38] constraints such as uniqueness, mandatory, and exclusion. We can expect conflicts between axiom restriction and relaxation operators.

Existing work might help in the future extension to a broader set of context dependencies and operators. E.g., in [66], algorithms are described for detecting unsatisfiability of ORM schema. There, conflict patterns provide insights for the specification of application conditions for productions on axioms. Furthermore, by founding our framework in terms of graph rewriting (Sect. 3.2), we aim to extend the use of conflict pair analysis we introduced in Ex. 7. Successful application of this exist in software evolution [60] and collaborative applications [59, 58], but could have a whole new application domain in context-driven ontology engineering.

6 Conclusion

Contexts play a very important role in real-world, gradual ontology elicitation and application efforts. In context-driven ontology engineering processes, such as ontology disambiguation, integration, and versioning, managing contexts effectively and efficiently is crucial for their success. However, contexts and especially their dependencies are still little understood.

In this article, we introduced a formal framework for supporting context dependency management processes, based on the DOGMA framework and methodology for scalable ontology engineering. Key notions are a set of context dependency operators, that can be combined to manage complex context dependencies like articulation, application, specialisation, and revision dependencies. In turn, these dependencies can be used

in context-driven ontology engineering processes tailored to the specific requirements of collaborative communities. This was illustrated by a case of interorganisational competency ontology engineering.

With the foundation of the framework firmly established, research can shift to developing more extensive descriptions of context dependency operators and types, and their systematic integration in community workflows through business rules. By including these context modules in the existing DOGMA methodologies and tools in the near future, experiments can be done to see which combinations of dependency management processes are required in practice. It is our conviction that this will demonstrate the true power of systematic context dependency management. Instead of being frustrated by out-of-control change processes, proper context dependency management support will allow human experts to focus on the much more interesting meaning interpretation and negotiation processes. This, in turn should make ontologies much more useful in practice.

References

1. Gruber, T.: Cyc: a translation approach to portable ontologies. *Knowledge Acquisition* **5(2)** (1993) 199–220
2. Guarino, N.: Formal ontology and information systems. In: Proc. of the 1st Int'l Conf. on Formal Ontologies in Information Systems (FOIS98) (Trento, Italy), IOS Press (1998) 3–15
3. Meersman, R.: The use of lexicons and other computer-linguistic tools in semantics, design and cooperation of database systems. In: Proc. of the Conf. on Cooperative Database Systems (CODAS99), Springer Verlag (1999) 1–14
4. Ushold, M., Gruninger, M.: Ontologies: Principles, methods and applications. *The Knowledge Engineering Review* **11(2)** (1996) 93–136
5. Farquhar, A., Fikes, R., Rice, J.: The ontolingua server: a tool for collaborative ontology construction. *Int'l Journal of Human-computer Studies* **46(6)** (1997) 707–727
6. de Moor, A.: Ontology-guided meaning negotiation in communities of practice. In Mambrey, P., Gräther, W., eds.: Proc. of the Workshop on the Design for Large-Scale Digital Communities at the 2nd International Conference on Communities and Technologies (C&T 2005) (Milano, Italy). (2005)
7. McCarthy, J.: Notes on formalizing context. In: Proc. of the 15th Int'l Joint Conf. Artificial Intelligence (IJCAI93) (Chambéry, France), Morgan Kaufmann (1993) 555–560
8. Sowa, J.: Peircean foundations for a theory of context. In: *Conceptual Structures: Fulfilling Peirce's Dream*, Springer-Verlag (1997) 41–64
9. Farquhar, A., Dappert, A., Fikes, R., Pratt, W.: Integrating information sources using context logic. In Knoblock, C., Levy, A., eds.: *Information Gathering from Heterogeneous, Distributed Environments*, Stanford University, Stanford, California (1995)
10. Buvač, S., Fikes, R.: A declarative formalization of knowledge translation. In: Proc. of 4th Int'l Conf. on Information and Knowledge Management (ACM CIKM 95). (1995)
11. Giunchiglia, F.: Contextual reasoning. special issue on I Linguaggi e le Macchine **XVI** (1993) 345–364
12. Nayak, P.: Representing multiple theories. In: Proc. of the 12th Nat'l Conf. on Artificial Intelligence (AAAI 94)(Seattle, Washington), AAAI Press (1994)
13. McCarthy, J., Buvač, S.: Formalizing context (expanded notes). Technical Report STAN-CS-TN-94-13, Stanford University (1994)

14. Buvač, S.: Resolving lexical ambiguity using a formal theory of context. In Van Deemter, K., Peters, S., eds.: *Semantic Ambiguity and Underspecification*, CSLI Publications (1996)
15. Meersman, R.: Reusing certain database design principles, methods and techniques for ontology theory, construction and methodology. Technical report, VUB STAR Lab, Brussel (2001)
16. De Leenheer, P., de Moor, A.: Context-driven disambiguation in ontology elicitation. In Shvaiko, P., Euzenat, J., eds.: *Context and Ontologies: Theory, Practice, and Applications*. Proc. of the 1st Context and Ontologies Workshop, AAAI/IAAI 2005, Pittsburgh, USA, July 9, 2005. (2005) 17–24
17. Guha, R., D., L.: Cyc: a midterm report. *AI Magazine* **11**(3) (1990) 32–59
18. Guha, R.: Contexts: a formalization and some applications. Technical Report STAN-CS-91-1399, Stanford Computer Science Department, Stanford, California (1991)
19. Theodorakis, M.: Contextualization: an Abstraction Mechanism for Information Modeling. PhD thesis, University of Crete, Greece (1999)
20. Guha, R., McCarthy, J.: Varieties of contexts. In: *CONTEXT 2003*. (2003) 164–177
21. Berners-Lee, T.: *Weaving the Web*. Harper (1999)
22. Guha, R., McCool, R., Fikes, R.: Contexts for the semantic web. In McIlraith, S.A., Plexousakis, D., van Harmelen, F., eds.: *Proceedings of the International Semantic Web Conference*. Volume 3298 of *Lecture Notes in Computer Science*., Springer Verlag (2004)
23. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: C-owl: Contextualizing ontologies. In: *Proc. of the 2nd Int'l Semantic Web Conference (ISWC 2003) (Sanibel Island, Florida)*, LNCS 2870, Springer Verlag (2003) 164–179
24. Singh, M.: The pragmatic web: Preliminary thoughts. In: *Proc. of the NSF-OntoWeb Workshop on Database and Information Systems Research for Semantic Web and Enterprises*. (2002) 82–90
25. Schoop, M., de Moor, A., Dietz, J.: The pragmatic web: A manifesto. *Communications of the ACM* **49**(5) (2006)
26. Bachimont, B., Troncy, R., Isaac, A.: Semantic commitment for designing ontologies: a proposal. In Gómez-Pérez, A., Richard Benjamins, V., eds.: *Proc. of the 13th Int'l Conf. on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web (EKAW 2002) (Sigüenza, Spain)*, Springer Verlag (2002) 114–121
27. Euzenat, J., Le Bach, T., Barrasa, J., et al.: State of the art on ontology alignment. Knowledge web deliverable KWEB/2004/d2.2.3/v1.2 (2004)
28. Kalfoglou, Y., Schorlemmer, M.: Ontology mapping: The state of the art. In: *Proc. of the Dagstuhl Seminar on Semantic Interoperability and Integration (Dagstuhl, Germany)*. (2005)
29. Klein, M., Fensel, D., Kiryakov, A., Ognyanov, D.: Ontology versioning and change detection on the web. In: *Proc. of the 13th European Conf. on Knowledge Engineering and Knowledge Management (EKAW02) (Sigüenza, Spain)*. (2002) 197–212
30. De Leenheer, P., Kopecky, J., Sharf, E., de Moor, A.: A versioning tool for ontologies (2006) EU IP DIP (FP6-507483) Deliverable D2.4.
31. de Moor, A., De Leenheer, P., Meersman, R.: DOGMA-MESS: A meaning evolution support system for interorganizational ontology engineering. In: *In Proc. of the 14th Int'l Conference on Conceptual Structures (ICCS 2006) (Aalborg, Denmark)*. LNAI 4068, Springer Verlag (2006) 189–203
32. Ding, Y., Fensel, D.: Ontology library systems: the key to succesful ontology re-use. In: *Proc. of the 1st Semantic Web Symposium (SWWS01) (Stanford, California)*. (2001)
33. Spyns, P., Meersman, R., Jarrar, M.: Data modelling versus ontology engineering. *SIGMOD Record* **31**(4) (2002) 12–17
34. Sowa, J.: *Knowledge Representation - Logical, Philosophical and Computational Foundations*. Brooks/Cole Publishing Co. (2000)

35. Gómez-Pérez, A., Manzano-Macho, D.: A survey of ontology learning methods and techniques. *OntoWeb Deliverable D1.5* (2003)
36. Reinberger, M.L., Spyns, P.: Unsupervised text mining for the learning of DOGMA-inspired ontologies. In: Buitelaar P., Handschuh S., and Magnini B.,(eds.), *Ontology Learning and Population*, IOS Press (2005) in press
37. Verheijen, G., Van Bekkum, J.: NIAM, an information analysis method. In: *Proc. of the IFIP TC-8 Conference on Comparative Review of Information System Methodologies (CRIS 82)*, North-Holland (1982)
38. Halpin, T.: *Information Modeling and Relational Databases (From Conceptual Analysis to Logical Design)*. Morgan Kauffman (2001)
39. Verheyden, P., De Bo, J., Meersman, R.: Semantically unlocking database content through ontology-based mediation. In: *Proc. of the 2nd Workshop on Semantic Web and Databases, VLDB Workshops (SWDB 2004)* (Toronto, Canada), Springer-Verlag (2004) 109–126
40. Jarrar, M., Demey, J., Meersman, R.: On reusing conceptual data modeling for ontology engineering. *Journal on Data Semantics* **1**(1) (2003) 185–207
41. Mitra, P., Wiederhold, G., Kersten, M.: A graph-oriented model for articulation of ontology interdependencies. In: *EDBT '00: Proceedings of the 7th International Conference on Extending Database Technology*, London, UK, Springer-Verlag (2000) 86–100
42. De Bo, J., Spyns, P., Meersman, R.: Assisting ontology integration with existing thesauri. In: *Proc. of On the Move to Meaningful Internet Systems (OTM2004)* (Ayia Napa, Cyprus), Springer Verlag (2004) 801–818
43. Fellbaum, C., ed.: *Wordnet, an Electronic Lexical Database*. MIT Press (1998)
44. Putnam, H.: *Mind, Language, and Reality*. Cambridge University Press, Cambridge (1962)
45. Sowa, J.: *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley (1984)
46. Brachman, R., McGuinness, D., Patel-Schneider, P., Resnik, L., Borgida, A.: Living with classic: When and how to use a KL-ONE-like language. In Sowa, J., ed.: *Principles of Semantic Networks*, Morgan Kaufmann (1991) 401–456
47. Rastier, F., Cavazza, M., Abeillé, A.: *Sémantique pour L'analyse*. Masson, Paris (1994)
48. Banerjee, J., Kim, W.: Semantics and implementation of schema evolution in object-oriented databases. In: *ACM SIGMOD Conf., SIGMOD Record*. (1987) 311–322
49. Noy, N., Klein, M.: Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems* **6**(4) (2004) 428–440
50. Kim, W., Chou, H.: Versions of schema for object-oriented databases. In: *Proc. of the 14th Int'l Conf. on Very Large Data Bases (VLDB88)* (L.A., CA.), Morgan Kaufmann (1988) 148–159
51. Roddick, J.: A survey of schema versioning issues for database systems. *Information and Software Technology* **37**(7) (1995) 383–393
52. Lerner, B.: A model for compound type changes encountered in schema evolution. *ACM Transactions on Database Systems (TODS)* **25**(1) (2000) 83–127
53. De Leenheer, P.: Revising and managing multiple ontology versions in a possible worlds setting. In: *Proc. of On the Move to Meaningful Internet Systems Workshops (OTM2004)* (Ayia Napa, Cyprus), LNCS 3292, Springer Verlag (2004) 798–818
54. Oliver, D., Shahr, Y., Musen, M., Shortliffe, E.: Representation of change in controlled medical terminologies. *AI in Medicine* **15**(1) (1999) 53–76
55. Heflin, J.: *Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment*. PhD thesis, University of Maryland, College Park, MD, USA (2001)
56. Stojanovic, L., Maedche, A., Motik, B., Stojanovic, N.: User-driven ontology evolution management. In: *Proc. of the 13th European Conf. on Knowledge Engineering and Knowledge Management (EKAW02)* (Sigüenza, Spain). (2002) 285–300

57. Mens, T.: A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering* **28**(5) (2002) 449–462
58. Edwards, W., Igarashi, T., LaMarca, A., Mynatt, E.: A temporal model for multi-level undo and redo. In Press, A., ed.: *Proc. of the 13th annual ACM symposium on User interface software and technology* (San Diego, CA). (2000) 31–40
59. Berlage, T., Genau, A.: A framework for shared applications with a replicated architecture. In Press, A., ed.: *Proc. of the 6th annual ACM symposium on User interface software and technology* (Atlanta, GA). (1993) 249–257
60. Mens, T.: Conditional graph rewriting as a domain-independent formalism for software evolution. In: *Proc. of the Int'l Conf. on Applications of Graph Transformations with Industrial Relevance* (Agtive 1999). Volume 1779., Springer-Verlag (2000) 127–143
61. Löwe, M.: Algebraic approach to single-pushout graph transformations. *Theoretical Computer Science* **109** (1993) 181–224
62. Shamsfard, M., Barforoush, A.: The state of the art in ontology learning: a framework for comparison. *the Knowledge Engineering Review* **18**(4) (2003) 293–316
63. Beale, S., Nirenburg, S., Mahesh, K.: Semantic analysis in the MikroKosmos machine translation project. In: *In Proc. of the 2nd Symposium on Natural Language Processing* (Bangkok, Thailand). (1995) 297–307
64. (OpenCyc) <http://www.opencyc.org>.
65. Lenat, D.: The dimensions of context-space. Cycorp technical report (1998)
66. Jarrar, M., Heymans, S.: Unsatisfiability reasoning in orm conceptual schemes. In: *Proc. of Int'l Conf. on Semantics of a Networked World* (Munich, Germany), Springer Verlag (2006) *Lecture Notes in Artificial Intelligence*, in press.