TooCoM : a Tool to Operationalize an Ontology with the Conceptual Graphs Model

Frédéric Fürst, Michel Leclère, Francky Trichet

Institut de Recherche en Informatique de Nantes
2 rue de la Houssinière - BP 92208
44322 Nantes France
`{furst,leclere,trichet}@irin.univ-nantes.fr`

Abstract. This article deals with the operational use of a domain ontology integrated into a Knowledge-Based System (KBS). It presents TooCoM, a tool dedicated to (1) the definition of ontologies with the Entity-Relationship paradigm and (2) the operationalization of ontologies in the context of the Conceptual Graphs model. TooCoM provides functionalities for specifying an operational scenario of use of the ontology which is under construction, for transcribing this ontology into the corresponding operational form and for using this operational form in an embedded inference engine.

Keywords: Ontology, Conceptual Graphs, Knowledge-Based Systems, Operationalization.

1 INTRODUCTION

Most of works which aims at developing tools for building an ontology focuses on the edition of the conceptual vocabulary, i.e. the terminological level. For instance, Protégé allows the knowledge engineer to build a hierarchy of concepts and to specify predefined properties of the concepts through the Frame model [11]. OntoEdit (renamed Kaon) is also based on the Frame paradigm. As Protégé, it focuses on the structuration of a set of concepts and on the specification of predefined properties of these concepts [20].

None of the tools listed within the OntoWeb project [6] aims at editing, in an intuitive and graphical way, the axioms of a domain. However, in our opinion, axioms are the main operational ressource of an ontology since they constrain the use of the conceptual vocabulary. Consequently, they are the only means to specify the semantics of a domain. For instance, in Protégé, the knowledge engineer must known the Protégé Axiom Language to specify the constraints and/or the rules of the domain. In OntoEdit, the specification of a non-predefined axiom must be done by using a logical formula.

TooCoM is a tool which adresses this problem. It allows the knowledge engineer (1) to specify the conceptual vocabulary of the domain by using the Entity-Relationship paradigm, (2) to specify the axioms of the domain in a graphical way and (3) to easily make these axioms operational in order to perform reasoning in the context of the Conceptual Graphs model[1]. For this last point, TooCoM can be considered as an innovative tool in the sense that it allows the knowledge engineer to follow reasoning processes in a graphical way. This aspect is very important because, in our opinion, this facilitates the appropriation and the control of the semantics which is associated to the ontology under construction. In other words, providing functionalities dedicated to a graphical appropriation of the implications of all the axioms (rules and constraints) of a domain makes the understanding (and therefore the refinement) of the semantics of a domain more easy.

As WebODE implements the METHONTOLOGY methodology to build an ontology [1], TooCoM implements original guidelines to specify axioms at the conceptual level and to specify the operational use of the ontology which determinates the operational form of the axioms.

From a technical point of view, TooCoM is based on CoGITaNT, a framework which offers capabilities to represent and manipulate Conceptual Graphs [10]. TooCoM has been tested in the context of the GINA project (Interactive and Natural Geometry) related to CAD (Computer-Aided Design) [13]. In this experiment, our tool has been used to build and to automatically operationalize an ontology of geometry [9].

The rest of this paper is structured as follows. Section 2 presents how building an ontology with TooCoM, in particular how specifying the conceptual vocabulary and the axioms. Section 3 first introduces the process we advocate to operationalize an ontology and then shows the application of this process in the context of the Conceptual Graphs model and its implementation in TooCoM. Finally, section 4 introduces a discussion about the innovative aspects of TooCoM in comparison with existing tools.

2 DEFINING AN ONTOLOGY WITH THE ENTITY-RELATIONSHIP PARADIGM

Defining an ontology with the Entity-Relationship (E/R) paradigm mainly consists in (1) specifying of the conceptual vocabulary of the domain which is considered and (2) specifying the semantics of the conceptual vocabulary through axioms.

2.1 The specification of the conceptual vocabulary

As implied by the Gruber's definition, (*« an ontology is a formal, explicit specification of a shared conceptualization »* [12]), the building of an ontology is based on a conceptualization, which is a conceptual description of the knowledge covered by the ontology. This description consists of a conceptual vocabulary which, in the context of the E/R paradigm, contains a set of concept types and a set of relation types which can both be structured by using subsomption links.

TooCoM allows the knowledge engineer to define such hierarchies, both for concept types and for relation types. Figure 1 shows an extract of the hierarchy of concept types

[1] Operationalizing knowledge consists in representing it with an operational language, according to an operational goal. An operational language is a formal language (i.e. a language having a syntax and formal semantics) which provides inference mechanisms allowing one to reason from its representations. An operational goal is specified by a *scenario of use* (cf. section 3.1).

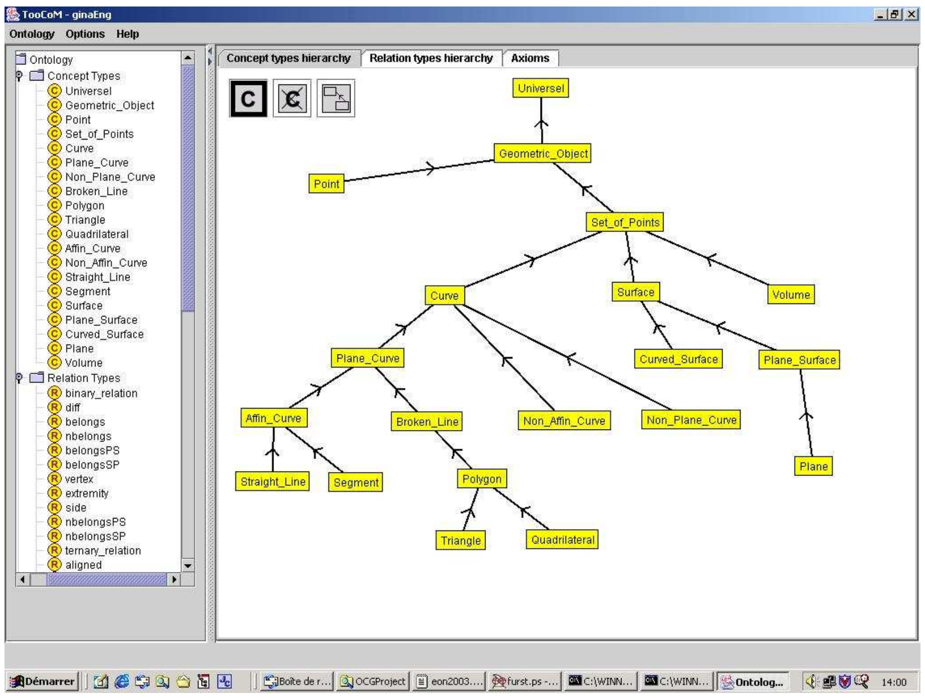


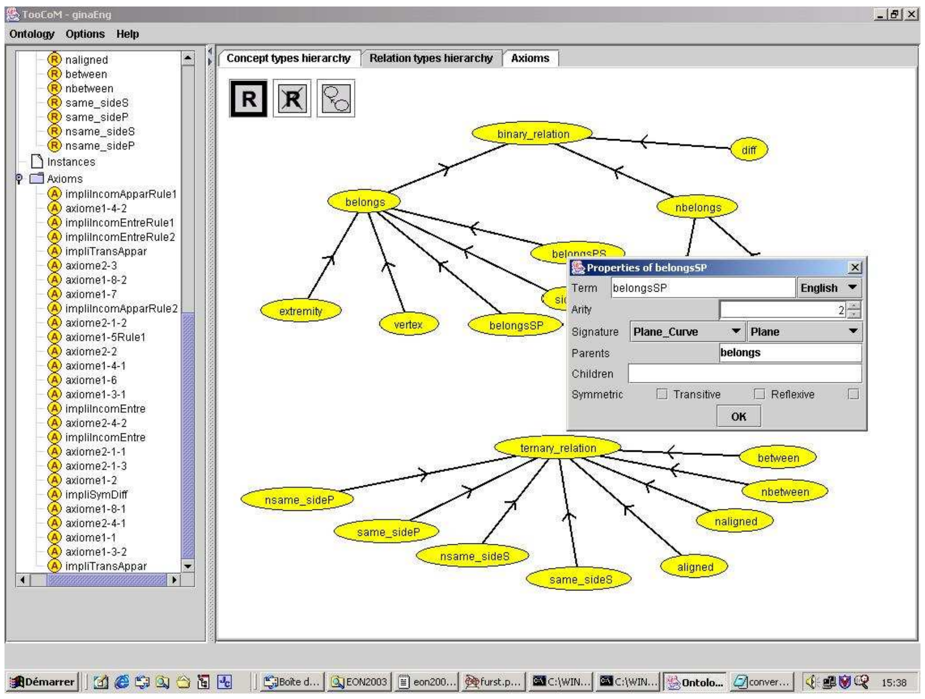
Fig. 1. A hierarchy of concept types in TooCoM. An arrow represents a subsomption link between a concept type and his parent concept type (« *a Triangle is-a Polygon* »).

which has been defined for the GINA project (i.e. an ontology of geometry defined according to Hilbert's book « *Grunlagen der Geometrie* »). Figure 2 shows the hierarchy of relation types.

2.2 The specification of the axioms

Axioms represent the intension of concept types and relation types and, generally speaking, knowledge which is not strictly terminological [19]. Axioms are specific to ontologies and, in our opinion, allow us to distinguish an ontology from a thesaurus. Thesaurus are only based on terminological representations and can be compared to light weight ontologies, whereas heavy weight ontologies contain the whole semantics of a domain [18]. Axioms specify the way the terminological primitives must be manipulated. Two types of axioms can be distinguished :

– the axioms that represent common and well-defined properties of concept types or relation types ;
– the axioms that represent properties specific to the domain .



Properties of belongsSP

Term	belongsSP	English
Arity	2	
Signature	Plane_Curve	Plane
Parents	belongs	
Children		

Symmetric ☐ Transitive ☐ Reflexive ☐

OK

Fig. 2. A hierarchy of relation types in TooCoM. The property box of the *belongsSP* relation type is open. Such a box shows the signature, the parents, the children and the algebraic properties of a relation type. For instance the *belongsSP* relationship can only be stated between a *Plane_Curve* and a *Plane*, it has the *belongs* relation type as parent and no child and bears any algebraic property.

The common properties, that we call *axiom schemata*, can correspond to:

- algebraic properties such as symmetry, reflexivity, transitivity;
- the *is-a* link between two concept types or two relation types (subsomption property);
- the signature or the cardinalities of a relation type;
- the exclusivity or the incompatibility between two concept types or two relation types (the incompatibility between two primitives P_1 and P_2 is formalized by $\neg(P_1 \wedge P_2)$, the exclusivity is formalized by $\neg P_1 \Rightarrow P_2$).

Classical axiom schemata can be specified by simply indicated the property of the relation types in the tool box (cf. figure 2), i.e. without creating a new axiom by using the *Axioms* panel. If an additional property of relation type (symmetry, transitivity or reflexivity) is specified, the corresponding axiom is automatically created and added to the ontology.

However, an axiom does not necessarily correspond to a schema. For instance, figure 3 shows the axiom 1.2 of Hilbert's axiomatics. This axiom, which expresses a property of identity between a *Straight_line* and a couple of *Points* does not correspond to a classical axiom schema and must be build in the axiom panel.

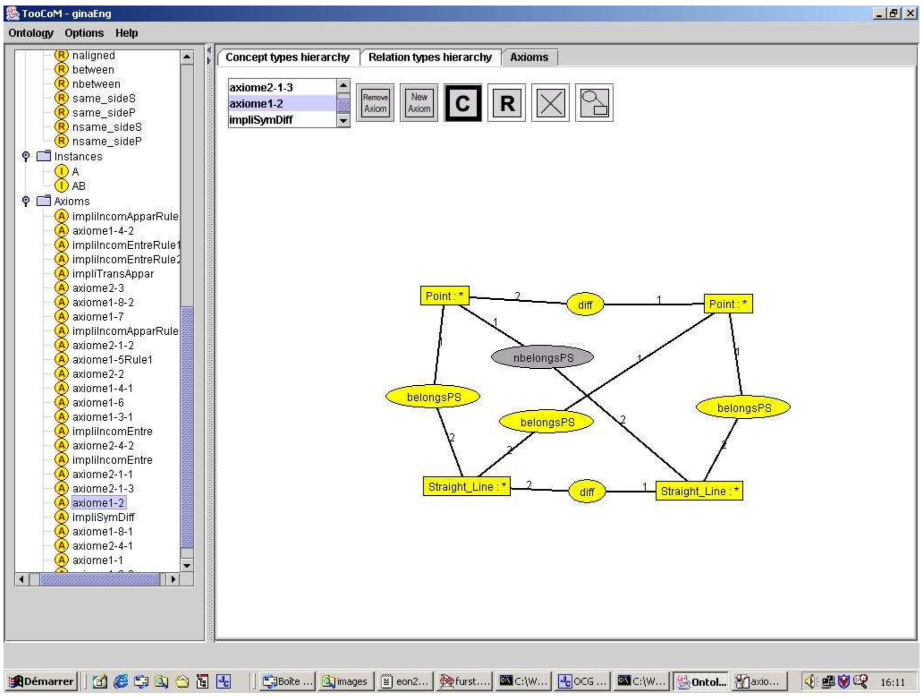


Fig. 3. Representation of an axiom in TooCoM. The yellow (bright) concepts and relationships represent the hypothesis part of the axiom and the gray (dark) concepts and relationships represent the conclusion part. Semantics of this axiom is as follows: *given two different points and two different straight lines, if one of these points belongs to the two lines, and if the other belongs to one the lines, it does not belong to the second line.*

In TooCoM, the subsomption links and the signatures of the relation types are the only properties that are embedded into the modeling paradigm underlying our tool, and they do not have to be expressed by axioms. All other properties of the conceptual primitives have to be specified as axioms via the definition of predefined axiom schemata in hierarchies of concept or relation types, or via the whole creation of an axiom in the *Axioms* panel.

An axiom is composed of an hypothesis part and a conclusion part, respectively represented by a *conceptual graph*[2]. A conceptual graph is a bipartite graph composed of concept vertices (representing objects of the domain) and relationship vertices (describing relationships between objects). Each vertex of a conceptual graph is labeled. A

[2] The Conceptual Graphs model, first introduced by Sowa [17], is a knowledge representation model which belongs to the semantic networks. An extension of this model, the SG family [2], presented in section 3.2, extends the model with reasoning primitives, rules and constraints.

concept vertex is labeled with the concept type from which the represented object is an instance. To identify the represented object, one can possibly add an individual marker. In that case, the vertex is called an individual concept. In other case, one adds to the concept type a star which denotes the generic marker (i.e. the identity of this concept is not defined). Such a vertex is called a generic concept. A relation vertex is simply labeled by a relation type specifying the nature of the link between the neighbouring concepts.

But this representation of axioms does not specify their operational semantics, in the sense that it does not specify the way the axioms will be used in an operational application. Because this operational semantics depends on the operational goal of the KBS, it can not be included in an ontology, which must be independent from any operational goal. Thus specifying this semantics conducts to an *operational ontology*, through an *operationalization* process.

3 OPERATIONALIZING AN ONTOLOGY WITH TooCoM

An ontology is only a conceptual representation of a domain, independently of any operational applications. To use an ontology in a KBS, it is necessary to transcribe the conceptual representation into a form in accordance with the way the KBS will be used. This form must be an operational form, in the sense that the knowledge representation model must offer operational mechanisms, such as inference mechanisms, in order to allow the manipulations to which the KBS is dedicated. For instance, to perform automatic reasoning, the operational formalism must allow the representation of derivation rules and the effective application of these rules on a set of facts. Thus, the use of an ontology in a KBS requires an *operationalization* process, that consists in transcribing the ontology in an operational formalism, in accordance with the operational use of the KBS.

3.1 The scenarii of use and the operationalization of axioms

The operationalization of an ontology is only conceivable for a well defined operational use, characterized by a precise *scenario of use* [5]. A scenario of use is the description of the purposes for which knowledge will be manipulated in the system. Defining a scenario of use mainly consists in describing the way the axioms will be used in the system, because the operational representation of terminological knowledge does not depend on the different contexts of application. Indeed the representation of a concept or a relation type is the same in the case of a system dedicated to knowledge validation or in the case of a system built to produce new facts from a knowledge base. Only the operational representations of the axioms are specific to the goal of the application.

We consider that an axiom can be used to validate knowledge in relation to the ontology or to produce new facts from a base. For instance, the axiom 1.6 of Hilbert *« If two points A and B of a straight line d belong to a plane α, then all the points of d belong to α »* can be used either to deduct the membership of points to a plane, or to indicate that a situation is not in accordance with the semantics of geometry, such as

« there are two points that belong to both a straight line and a plane and a point of the straight line which does not belong to the plane ».

Moreover, an axiom can be used when the user of the system asks for it, or it can be applied automatically by the system everywhere it is possible. The first application is called *explicit*, the second *implicit*. For instance, the axiom 1.3.1 of Hilbert *« On a straight line, there are at least two points »* can be implicitly used if the user is not supposed to apply this axiom before considering points on a straight line or, on the contrary, can be explicitly used if he is supposed to resort to the axiom for considering such points, for instance for educational purposes.

So, operationalizing an ontology requires, for each axiom, the choice of a *context of use* which specifies the purpose for which the axiom will be used and how it will be applied in the system. The different contexts of use we have identified are:

– The **inferential and explicit** context of use: the user applies the axiom by himself on a fact base to produce new facts;

– The **inferential and implicit** context of use: the axiom is applied by the system on a fact base to produce new facts;

– The **validation and explicit** context of use: the user applies the axiom by himself to check that a fact base is in accordance with the semantics of a domain;

– The **validation and implicit** context of use: the axiom is applied by the system to verify that a fact base is in accordance with the semantics of a domain.

A *scenario of use* consists in a set of contexts of use choosen for each axiom of the ontology. Generally speaking, the operational form of an ontology includes inferential mechanisms and validation mechanisms. These mechanisms are required for the automatic (or semi-automatic) manipulation of knowledge. For instance, a scenario dedicated to a computer-aided teaching application allows the user to apply knowledge to deduce new facts or to check his work. Such a scenario comprises automatic inferences and validation processes, in accordance with the level of the user.

Figure 4 presents the general inference cycle through which the axioms are applied in a KBS. First the user can add facts to the fact base, then he can apply an axiom choosen between the inferential and explicit ones. Then the system applies all the inferential implicit axioms in order to sature the fact base with implicit knowledge. Finally, a validation step, which can be partially leaded by the user, permits to detect « semantical inconsistencies » in the fact base.

Two particular scenarii can be distinguished: the pure validation scenario, where the operational ontology is used to check a fact base according to the semantics of a domain (all axioms are operationalized in a validation context of use), and the inferential and implicit scenario, where the operational ontology is used to automatically produce new knowledge (all axioms are operationalized in an implicit context of use). To define the scenario of use of an ontology, the context of use of each axiom must be specified. This context constrains the operational form of the axiom. But, of course, the choice of the operational knowledge representation language also constrains this form.

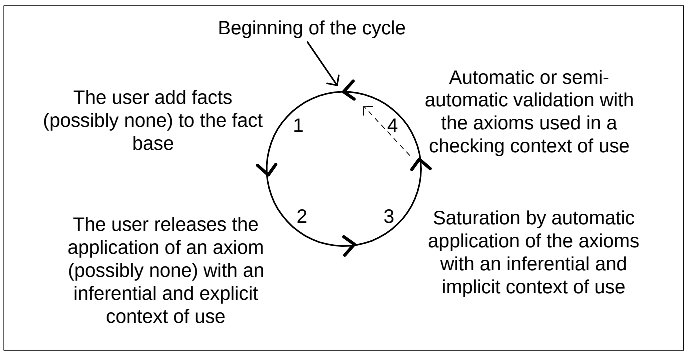

Beginning of the cycle
The user add facts (possibly none) to the fact base
1
Automatic or semi-automatic validation with the axioms used in a checking context of use
4
2
3
The user releases the application of an axiom (possibly none) with an inferential and explicit context of use
Saturation by automatic application of the axioms with an inferential and implicit context of use

Fig. 4. The inference cycle dedicated to the use of an operational ontology.

3.2 The operationalization of the axioms with the Conceptual Graphs model in TooCoM

TooCoM is based on an extension of the Conceptual Graphs model (CGs). The CGs model is an operational knowledge representation language which provides conceptual primitive representations through concepts and relationships between these concepts [17]. The subsomption property and the signature of relationships are integrated in the model. The other axioms, that express the way the primitives must be manipulated, can be represented with three types of reasoning primitives, that have been added as an extension of the model, the SG family [2]:

- The **positive constraints**, with an hypothesis part and a conclusion part, of which the semantics is: if the hypothesis part is present, then the conclusion part must be present (otherwise the constraint is broken);
- The **negative constraints**, with an hypothesis part and a conclusion part, of which the semantics is: if the hypothesis part is present, then the conclusion part must be absent (otherwise the constraint is broken);
- The **rules**, with an hypothesis part and a conclusion part, of which the semantics is: if the hypothesis part is present, then the conclusion part can be produced.

A rule can be implicitly used by the system (i.e. applied everywhere the hypothesis of the rule is present) or explicitly applied by the user (on a given fact in the knowledge base). A negative or positive constraint can be automatically used by the system (i.e. checked everywhere in the knowledge base) or explicitly applied by the user.

In order to allow the automatic operationalization of ontologies in TooCoM, we have defined operationalization mechanisms for each form of axiom. For instance, an axiom can have the following form:

$$\forall x_1, ..., x_n \; H \Rightarrow \exists y_1, ..., y_m \; r_1(..) \wedge .. \wedge r_p(..) \qquad (1)$$

where r_i are relationships between the x_i and/or y_j variables and H a conjonction of predicats which express concepts or relations.

The different operational forms of such an axiom, depending on the context of use, are:

– Inferential and implicit context of use: the axiom is operationalized by an implicit rule which corresponds to the the logical formula $\forall x_1, ..., x_n\ H \Rightarrow \exists y_1, ..., y_m\ r_1(..) \wedge .. \wedge r_p(..)$;
– Inferential and explicit context of use: the axiom is operationalized by an explicit rule which corresponds to the logical formula $\forall x_1, ..., x_n\ H \Rightarrow \exists y_1, ..., y_m\ r_1(..) \wedge .. \wedge r_p(..)$ and p negative constraints which correspond to the statement $\forall x_1, ..., x_n\ H\ (\bigwedge r_i(..))_{i=1..p, i \neq j}$, it can not exist $r'_j(..)$, $j = 1..p$, where r'_j is exclusive with r_j in the ontology[3]. If any relationship exclusive with r_j exists in the ontology, the corresponding constraint is replaced by q negative constraints which correspond to the statement $\forall x_1, ..., x_n\ H\ (\bigwedge r_i(..))_{i=1..p, i \neq j}$, it can not exist $r'_{j_k}(..)$, $k = 1..q$, where r'_{j_k} are all incompatibles with r_j ;
– Validation and implicit (respectively explicit) context of use: the axiom is operationalized by p negative and implicit (respectively explicit) constraints $\forall x_1, ..., x_n\ H\ (\bigwedge r_i(..))_{i=1..p, i \neq j} \Rightarrow r'_j(..)$, $j = 1..p$, where r'_j is exclusive with r_j in the ontology. If any relationship exclusive with r_j exists in the ontology, the corresponding constraint is replaced by q negative constraints which correspond to the statement $\forall x_1, ..., x_n\ H\ (\bigwedge r_i(..))_{i=1..p, i \neq j}$, it can not exist $r'_{j_k}(..)$, $k = 1..q$, where r'_{j_k} are all incompatibles with r_j .

In TooCoM, the user can build an operational ontology by specifying the context of use of each axiom of the ontology. According to this context, each axiom is automatically transcribed into an appropriate form (i.e. a rule, a constraint, a rule and a set of constraints or a set of constraints). Then, the operational ontology, which includes the conceptual primitives and the axioms in an operational form, can be exploited by the TooCoM inference engine which implements the reasoning cycle presented in figure 4.

3.3 The use of an operational ontology in TooCoM

TooCoM provides an inference engine based on the manipulation of conceptual graphs. This inference engine uses the CoGITaNT framework which allows to compare graphs and to apply CG rules through a graph projection operator [10]. By using this inference engine, the knowledge engineer can test the ontology under construction by applying the operational ontology to different situations. For instance he can state a fact represented by a graph and runs the engine over this fact. During the explicit inferential phase, he can choose the axiom he wants to apply and where he wants to apply it. The result of the reasoning process is displayed in real-time in the interface and the user can check if the resulting fact is correct in relation to the result which is intended. Again, as shown in figures 5 and 6, we argue in favor of a graphical semantics. These figures present the running panel of the inference engine.

If the user has a set of competency questions, he can check it with the inference engine. Moreover, the system can indicate exactly what axiom creates an inconsistency or

[3] The incompatibility between two primitives P_1 and P_2 is formalized by $\neg(P_1 \wedge P_2)$, the exclusivity is formalized by $\neg P_1 \Rightarrow P_2$.

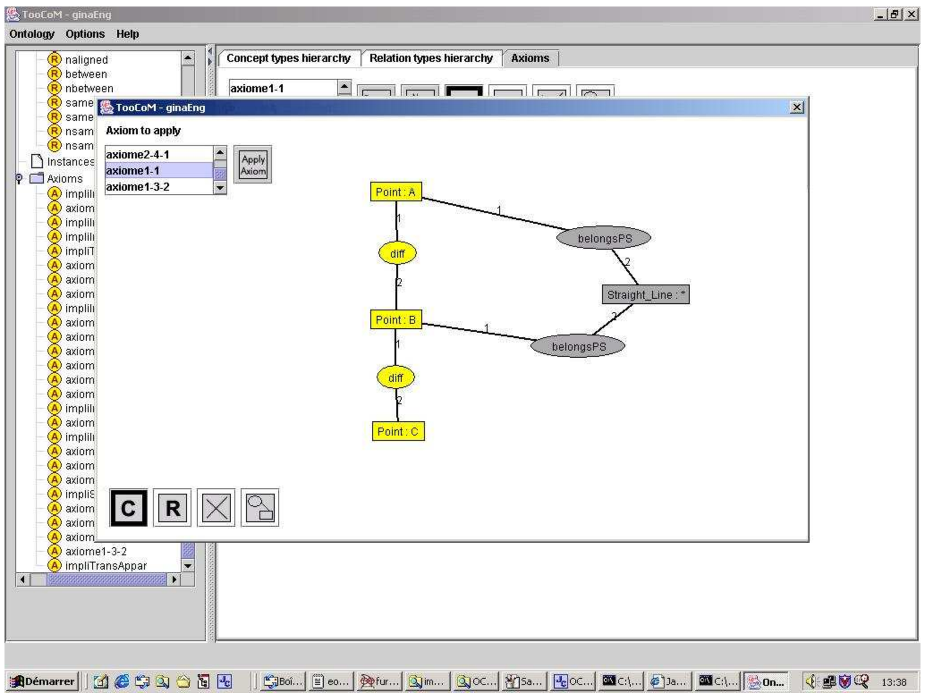


Fig. 5. A step of an inference cycle. The user has build a graph with three points A, B and C where A is different from B and B different from C (the graph appears in bright color). He selects the axiom 1.1 (*given two different points, it exists a straight line to which belong these two points*) which can be applied on the points A and B, or B and C (the conclusion part of the axiom appears in dark color). The system suggests to the user different projections on which the axiom can be applied. By using the keyboard arrows, the user can examine the different projections and apply the explicit axiom where he wants. In this example, the user applies the axiom on the points A and B (cf. figure 6 for the next step).

what axiom is lacking to answer the question. For instance, in the domain of geometry, we have use TooCoM to produce an operational form of the ontology appropriated to the automatic theorem proof checking [9]. In this case, all the Hilbert's axioms have been operationalized through an explicit and inferential context of use and the other axioms (e.g. the exclusivity between relation types) have been operationalized through an implicit and inferential context of use. By testing the proof of some theorems, we have discover some missings, which correspond to implicit knowledge not stated by Hilbert in his book, but really used in the proofs [9].

The building of different kinds of KBS is possible as far as the system can use the general reasoning cycle. In the context of geometry, we can adapt the scenario of use to automatically generate a module of an Intelligent Tutoring System which will use some axioms to validate the student's assertions and others to complete these assertions, whereas the student will use the explicit axioms to prove a theorem or to build a geometric figure.

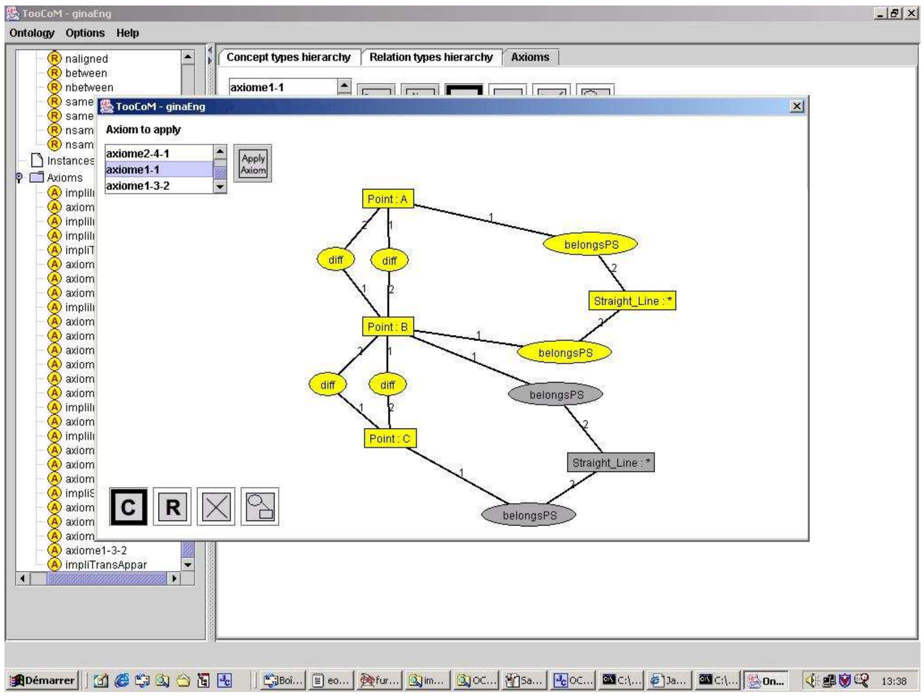


Fig. 6. After applying the axiom 1.1, the system automatically applies the implicit rules, and deduces the difference between the points from the symmetry property of the *diff* (difference) relation type. The user can then apply another axiom, for instance the axiom 1.1.

4 RELATED WORK

The first aspect that differentiates TooCoM from its related tools is that it is based on the Entity-Relation paradigm to structure an ontology, whereas most of other tools dedicated to the building of ontology, like OILEd [3], Protégé [11] or OntoEdit [20], are based on the Frame paradigm. Indeed, TooCoM is based on the Conceptual Graphs model which provides both a conceptual paradigm used to structure the terminological level of the ontology and reasoning mechanisms based on graph homomorphism in keeping with the first order logic.

Then, most of existing tools provides a textual mode to specify conceptual vocabulary and axioms. For instance, in OntoEdit, the specification of a non-predefined type of axiom requires the use of the F-Logic syntax [20]. But some of them allows the knowledge engineer to build ontologies in a graphical way: WebOnto provides a graphical interface for the edition of the conceptual vocabulary but not for the edition of axioms [7]. The graph based paradigm used in TooCoM is more intuitive than a textual one and it allows the knowledge engineer to specify both the terminological knowledge and all kind of axioms in a graphical interface, without knowing a textual axiom language.

We think that a graphic visualization of the inferences carried out is a significant factor which, on the one hand, facilitates the appropriation of a formal system and, on the other hand, allows the expert to validate the adopted model on its own (without reinterpretation of the implemented reasonings by a logician)[4]. The use of a graphical langage to build an ontology, which is a knowledge model, is coherent with the use of graphical languages, as UML, to build modelization in the programming domain.

The second, and most important, innovative aspect of TooCoM, is to allow the representation and the operationalization of all kinds of axioms. As fast as the use of ontology is growing, it becomes necessary to represent more and more complex properties of the concepts. For instance, the specification of OWL [16] includes new properties, as intersection of concept classes or algebraic properties, that do not appear in the RDFS specification. In our opinion, a complete ontology representation language must allow to represent any axiom, and not only predefined axioms. This allows the knowledge engineer to define properties that are not included in the language. For instance, in the domain of geometry, a lot of properties expressed through mathematical axioms can not be related to well defined properties, like algebraic properties.

An other advantage of the operational representation of axioms is the possibility to use ontologies for reasoning. This aspect becomes more and more important for the applications of the Semantic Web [8]: the Web services will use ontologies to reason and this requires the representation of axioms and not only the representation of terminological primitives organized in hierarchies. For instance, the RuleML language [4] is dedicated to the representation of rules and constraints in order to allow deduction, rewriting, and further inferential-transformational tasks. But the operational representation of axioms is conditioned by their operational uses. So, building operational representation of axioms requires an operationalization process through which these representations are produced according to contexts of use.

The representation of all kinds of axioms and their use in an inference engine through an original operationalization process allows to perform the original goal of Protégé, that is the interactive building of a KBS [11]. Moreover, in TooCoM, it is possible to automatically make the ontology operational and to manipulate it at a conceptual level. The context of use of each axiom can be specified and the KBS appropriated to the application which is intended can be automatically generated. As in many tools, this mechanism permits a constraint checking of the ontology. But it also allows the knowledge engineer to easily check the completeness of the ontology, by submitting competency questions to the inference engine.

At this moment, the ontologies can be stored in the BCGCT format [15], which is peculiar to the CoGITaNT framework, or in the CGXML format. These formats allow to represent the terminological primitives of a domain, the subsomption links between these primitives, the instances of concepts types, and axioms in rule form. We plan to add a module in order to allow the storage and the loading of ontologies in other common ontology languages like RDFS or OWL, as far as the expressivity of these langages allows us to represent all axioms.

[4] The validation can then be considered as a simple study of graphical explanations of the reasonings that have been performed by the system.

5 CONCLUSION

TooCoM allows a knowledge engineer to build ontologies within the Entity-Relationship paradigm, and to specify both the terminological knowledge of a domain and the semantics of this domain through axioms. The main characteristics of TooCoM is the possibility to define all kinds of axioms and to generate different operational ontologies from the specification of scenarii of use. So, thanks to a graphical semantics, TooCoM facilitates the appropriation of a global understanding of the semantics of the domain mainly defined by the axioms.

The operationalization mechanism provided by TooCoM permits, via the definition of an operational scenario, to produce operational ontologies. These operational ontologies can be used to validate the ontology itself, by submiting competency questions to the inference engine. This corresponds to a knowledge level prototyping approach [14]. For instance, the experiment we have done in the domain of geometry has lead us to modify our ontology after that the proof of a theorem failed.

The operationalization guideline implemented in TooCoM must be extended to other formalisms than the CGs model. In particular, the use of a combination of OWL, to represent the terminological knowledge, and RuleML, to represent axioms, is planned. It will permits to build operational ontologies that can be used on the Web.

References

1. J. Arpirez, O. Corcho, M. Fernandez-Lopez, and A. Gomez-Perez. Weboe: a workbench for ontological engineering. In *Proceedings of the first International Conference on Knowledge Capture (K-CAP'2001), Victoria, Canada*, 2001.
2. J.F. Baget and M.L. Mugnier. The sg family: Extensions of simple conceptual graphs. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'2001)*, pages 205–210, 2001.
3. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. Oiled: a reason-able ontology editor for the semantic web. In *Proceedings of KI2001, Joint German/Austria Conference on Artificial Intelligence*, volume 2174, pages 396–408. Springer Verlag LNAI, 2001.
4. H. Boley, S. Tabet, and G. Wagner. Design rationale of ruleml : a markup language for semantic web rules. In *Proceedings of the Semantic Web Working Symposium (SWWS'2001)*, 2001.
5. J. Bouaud, B. Bachimont, J. Charlet, and P. Zweigenbaum. Methodological principles for structuring an ontology. In ACM Press, editor, *Proceedings of IJCAI'95 Workshop: Basic Ontological Issues in Knowledge sharing*, 1995.
6. OntoWeb consortium (coordinated by Asuncion Gomez Perez). A survey on ontology tools. technical report IST-2000-29243, IST, 2002.
7. J. Domingue. Tadzebao and webonto: Discussing, browsing and editing ontologies on the web. In *Proceedings of the Eleventh Knowledge Acquisition Workshop (KAW'98)*, 1998.
8. D. Fensel and C. Bussler. Semantic web enabled web services. In *Proceedings of International Semantic Web Conference (ISWC'2002)*, volume 2342, pages 1–2. Springer-Verlag LNCS, 2002.
9. F. Fürst, M. Leclère, and F. Trichet. Contribution of the ontology engineering to mathematical knowledge management. *Annals of Mathematics and Artificial Intelligence, Kluwer Academic Publishers*, (38):65–89, 2003.

10. D. Genest and E. Salvat. A platform allowing typed nested graphs : how cogito became cogitant. In *Proceedings of the International Conference on Conceptual Structures (ICCS'98)*, volume 1453, pages 154–161. Springer-Verlag LNAI, 1998.
11. J.H. Gennari, M.A. Musen, R.W. Fergerson, W.E. Grosso, M. Crubezy, H. Eriksson, N.F. Noy, and S.W. Tu. The evolution of protégé: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58:89–123, 2003.
12. T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
13. O. Lhomme, P. Kuzo, and P. Macé. Desargues, a constraint-based system for 3d projective geometry. *Geometric Constraint Solving and Applications*, ISBN:3-540-64416-4, 1998.
14. A. Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.
15. CoGITaNT Home Page. http://cogitant.sourceforge.net/docs/index.html.
16. Ontology Web Language Home Page. http://www.w3.org/tr/2002/wd-owl-guide-20021104/.
17. J. Sowa. *Conceptual Structures : information processing in mind and machine*. Addison-Wesley, 1984.
18. S. Staab. An extensible approach for modeling ontologies in rdf(s). In *presentation at the ECDL2000 Workshop on the Semantic Web, http://www.ics.forth.gr/isl/SemWeb/PPT/Staab.ppt*, 2000.
19. S. Staab and A. Maedche. Axioms are objects too: Ontology engineering beyong the modeling of concepts and relations. Research report 399, Institute AIFB, Karlsruhe, 2000.
20. Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke. Ontoedit: colllaborative ontology development for the semantic web. In *Proceedings of the International Semantic Web Conference*, volume 2342, pages 221–235. Springer-Verlag LNCS, 2002.