

A UNIFIED TOOL FOR EDITING INFORMATION OF DIFFERENT LEVELS OF ABSTRACTION

Vassili Orlov, Alexander Kleschev

Expert Systems Department, Institute for Automation and Control Processes, Russian Academy of Sciences

5 Radio Street, 690041, Vladivostok, Russia

Email: v_orlov@yahoo.com, kleschev@iacp.dvo.ru

Web site: <http://www.iacp.dvo.ru/es/>

Keywords: Information acquisition, representation, unified editing tool, compatibility, intelligent support

Abstract: Ontology specification languages, ontologies of different levels of abstraction, knowledge and data are useful in the life cycle of knowledge-based systems. Since methods of processing some of these kinds of information are still in a research stage, new formalisms for representation of this information are proposed. As a result, extra efforts on developing tools for editing information represented using new formalisms are needed. These tools often turn out to be mutually incompatible. Meanwhile, the mentioned kinds of information are closely interrelated. This interrelation results in additional efforts on providing the editing tools compatibility. Users of the editing tools need essentially different level of support in the editing process. As a result, considerable efforts and resources are spent for developing experimental information editing tools, which provide their users with an appropriate support, and for establishing the tools compatibility. This paper presents a model of a unified editing tool that is intended for solving this problem.

1 INTRODUCTION

The following kinds of information are useful in the life cycle of knowledge-based systems (Kleschev and Orlov, 2001): classifications of artificial languages (ALs), specifications of ALs (in particular ontology specification languages, OSLs), ontologies of different levels of abstraction, domain knowledge and data. Hence the task of their computer editing appears. Since methods of processing some of these kinds of information are still in a research stage, different experimental formalisms for representation of this information are proposed. As a result, *extra efforts on developing experimental tools* for editing information represented using the experimental formalisms are needed. These tools often turn out to be mutually incompatible. Meanwhile, with respect to the editing task, the mentioned kinds of information are closely interrelated (Kleschev and Orlov, 2001): any OSL is just a kind of an artificial language, ontologies are described in terms of OSLs or in terms of ontologies of a higher level of abstraction, domain knowledge – in terms of corresponding ontologies and domain data are interpreted in terms of a particular knowledge base. This interrelation results in *additional efforts on*

providing the editing tools compatibility. Potential users of the editing tools are information carriers. Classes of users are linguists, knowledge engineers, domain experts and domain specialists (Kleschev and Orlov, 2001). Namely, linguists are carriers of information about kinds of ALs and about ALs, knowledge engineers are carriers of information of high level of abstraction, domain experts are domain ontologies and domain knowledge carriers, and domain specialists are facts of observed reality carriers. Users of different classes solve essentially different tasks and have essentially different skills in utilization of computer. They need *essentially different support* in the editing process therefore.

As a result, considerable efforts and resources are spent for developing experimental information editing tools that provide the corresponding classes of users with an appropriate support, and for establishing the tools compatibility.

The purpose of this work is to present a model of a unified editing tool, which is intended for editing different kinds of information that have different levels of abstraction and for providing all the classes of its users with an appropriate support during the editing process. Let us introduce the following pronounceable abbreviation: the unified tool for editing Information of Different levels (Extents) of Abstraction is further referred to as *IDEA Editor*.

2 BACKGROUND: CUE AND PROTÉGÉ-2000

This section presents a brief review of two main approaches to solving the mentioned compatibility and usability problems by the example of CUE workbench (Gruber, 1992; van Heijst, Schreiber, and Wielinga, 1996) and Protégé-2000 system (Grosso et al., 1999; Noy, Fergerson, and Musen, 2000).

2.1 CUE Workbench

CUE is aimed at editing application-specific knowledge. With its peculiar approach, it supposes preparing task models, application-specific ontologies, knowledge acquisition (KA) agenda and strategies. However, it is noticed in (Corcho, Fernandez-Lopez, and Gomez-Perez, 2001) that application-independent ontologies are vital as well.

Ontologies are represented in the Ontolingua language. Knowledge bases are collections of instances of classes from corresponding ontologies. CUE can translate ontology definitions written in Ontolingua into a variety of existing ALs. This approach is aimed at achieving interoperability at the level of ALs. However, special translators have to be developed for each new AL.

The tools of CUE are QUITE, QUOTE and QUAKE. Knowledge engineers construct task models via QUITE; in cooperation with domain experts, they construct application ontologies via QUOTE, and map task models onto application ontologies via QUITE. Knowledge engineers also have to specify KA agenda and strategies via QUAKE. QUAKE utilizes the defined ontologies and strategies in order to support domain experts in editing application knowledge. Thus, the editing process seems to be rather intricate.

QUOTE provides its users with task-oriented terminology from the previously built task models, and allows checking ontologies against Ontolingua syntax. QUAKE is primarily intended for supporting domain experts in the KA process. It can be used in the two modes of interaction: passive, where its user guides the KA dialogue himself / herself, and active, where the tool acts as an interviewer on the basis of corresponding KA strategy. QUAKE also supports each KA process with a kind of consistency and completeness checking. Consistency checking in QUAKE means checking entered pieces of knowledge against the corresponding definitions in the application ontology. In the active mode QUAKE is also responsible for the following kind of completeness checking: it is able to register which parts of the desired knowledge are fully instantiated, partially instantiated, or empty.

2.2 Protégé-2000

Adjusting the terminology from (Grosso et al., 1999; Noy, Fergerson, and Musen, 2000), Protégé-2000 is aimed at editing ALs non-native for it, and information of different levels of abstraction, namely ontologies and knowledge bases.

The AL of Protégé-2000 is based on OKBC (Chaudhri et al., 1998). It is used for representing any information edited via the system. Protégé-2000 is interoperable with other OKBC-compatible KBSs; special translators still have to be developed in order to make Protégé-2000 interoperable with OKBC-incompatible KBSs. So-called *meta-class architecture* of Protégé-2000 enables developers to use the system for editing information using terminology of ALs non-native for the system; unfortunately, it is recognized in (Noy, Fergerson, and Musen, 2000) that this meta-class architecture is not enough for describing ALs in the common case.

The central tenet of the Protégé-2000 methodology is the use of meta-information in acquiring information. Namely, knowledge engineers define (to the extent the tool allows the definition) ALs non-native for Protégé-2000, as well as meta-information. Meta-information can be acquired on the basis of the meta-class architecture, or using terms of a previously defined AL. Further, knowledge engineers are to generate domain-specific ontology- and knowledge-acquisition systems via Protégé-2000, and tailor interfaces of the generated systems to the potential users. Domain experts edit ontologies (via previously generated ontology-acquisition systems), and knowledge bases (via previously generated KA systems).

Protégé-2000 enables structured information entry by using information-acquisition forms centred around classes defined in the meta-information and values for their slots. Class hierarchy defined in meta-information guides a desired information acquisition dialog. The forms utilize names of classes and their slots as user-specific terminology. Meta-information contains descriptions of facets and axioms associated with classes and slots; these facets and axioms help in verifying information being entered for formal correctness.

2.3 Summary

The reviewed approaches show what level of support for editing process can be achieved by separating meta-information and desired information levels. Unfortunately, these approaches fail to solve the editing tools incompatibility problem that occurs on the topmost level of the kinds of information hierarchy – on the level of various ALs.

3 A DESCRIPTION OF THE IDEA EDITOR

The following problems arise in the IDEA Editor development process. Problem 1: how to make the single tool sufficient to solve the task of editing all the mentioned levels of information? Problem 2: how to make information of higher level of abstraction, which is formed by means of the tool, useful in the process of editing information of lower level of abstraction by means of the same tool? Problem 3: how to ensure that the single tool provides all its users of corresponding classes with a necessary intelligent support?

In this paper the *necessary intelligent support* is understood as follows. It is in the supplying users of an intelligent editing tool with intelligence (represented in a form and terminology understandable to the users) about what information is necessary in each step of a particular editing process, and in the providing them with assistance in entering formally correct information. The assistance is in the performing computations (when possible) without user participation, preventing (when possible) formal errors in information to be entered by offering a choice of correct pieces of information, and (in the cases when it is impossible to prevent errors) verifying formal errors in information entered manually.

IDEA Editor is an interpreter of a specially introduced Language for representing Information of Different levels (Extents) of Abstraction (IDEA Language). A solution of the first problem mentioned above is in the fact that the IDEA Language has a set of primitives sufficient for describing any of the mentioned levels of information. Sufficiency for specifying various ALs is one of the main design objectives for the IDEA Language. It is worth notification that according to the theory of ALs, each AL is characterized by its abstract syntax, context conditions, semantics, and concrete syntax. Further, regarding an information editing process, an abstract syntax, context conditions restricting the set of well-formed propositions, and a concrete syntax of a language are important at the phase of editing information represented in the language. Generally, information in the IDEA Language is a semantic network – net of concepts, where each concept is assigned a number of propositions that define attributes, dialog templates, value areas and concrete syntax related to it; moreover, a number of context conditions that apply mutual restrictions on the intended meanings of the concepts is assigned to the whole net. The IDEA Language competes with the MOF Model (Meta Object Facility Specification, 2002).

Semantics of the IDEA Language primitives enables IDEA Editor to interpret input information represented in the IDEA Language, to ask the user about new information, to resolve ambiguities occurred in the interpretation process, and to form output information in the same IDEA Language. The input information is further referred to as *metainformation* and the output information – as *desired information*. This approach allows IDEA Editor to utilize desired information of high level of abstraction as metainformation in the process of editing information of lower level of abstraction, and thus solves the second problem mentioned above.

IDEA Editor operates according to the following (given informally) description of a unified editing process. The unified editing process defines a relationship between two states of each editing process: between metainformation and desired information. Generally, the metainformation specifies relations among abstract concepts, while these concepts are defined concretely in the desired information. Relations between concepts from the metainformation are saved in the desired information, and new relations can be added. Metainformation also describes the necessary intelligent support that users are to be provided with in the process of editing the desired information. Thus, metainformation provides IDEA Editor with the following intelligence: which information (on the basis of the concepts and their attributes), in what sequence (on the basis of relations among the concepts) and form (on the basis of the dialog templates) is to be acquired from a user; which pieces of correct information can be offered to the user as variants (on the basis of the value areas) so that he can choose instead of guessing, or how to verify formal errors in information that he enters manually (on the basis of the context conditions); as well as which calculations to perform without user participation (on the basis of the computational attributes). The description of a necessary intelligent support is put into metainformation at the phase of editing the metainformation. This approach is directed towards solving the third problem mentioned above. The content of metainformation allows IDEA Editor to provide its users with structured data entry facilities.

IDEA Editor works as an interviewer. It explores the desired information (to the extent it is defined). Exploration is in the looking through the network of concepts (that is defined by concepts and relations among them). Exploration always begins with a distinguished concept-origin of the semantic network of the desired information (user is asked to define such a concept if it is not defined). IDEA Editor asks its user whether he/she wants to change the concept being explored (linguists and knowledge

engineers can also change propositions assigned to the concept), delete the concept or change its structure. When the user tries to change or delete the concept, IDEA Editor checks related context conditions in the metainformation as to whether the operation can be allowed. To change the structure of a concept is to define new sub-concepts for it – new concepts in the desired information. In the process of changing the structure of a concept IDEA Editor uses metainformation in order to provide its user with necessary intelligent support. Information acquired from a user on a particular step of an editing process is utilized on the next steps of the process in order to improve the intelligent support.

In some cases there is no metainformation or it is insufficient (to the user's opinion) for forming the desired information. For instance, there is no metainformation in the process of editing classifications of ALs and specifications of ALs. Linguists and knowledge engineers are competent in recognition and resolving this problem. They are qualified enough to solve the problem using the IDEA Language and auxiliary (any already had been defined by that time) information. Thus, the IDEA Language is the means (linguists- and knowledge engineers-oriented) intended for defining information dynamically as needed.

In view of the levels of information that are under consideration in this work, users of appropriate classes utilize IDEA Editor in editing processes consisting of the following phases.

1. The linguist classifies ALs and specifies necessary ALs (in particular, OSLs). There is no metainformation at this phase; this process is supported with IDEA Language therefore.

2. The knowledge engineer edits ontologies of different levels of abstraction, and embeds description of the necessary intelligent support, which the user is to be provided with on the next phase, into them. A previously defined OSL can be utilized as metainformation for editing an ontology of a level of abstraction in terms of the OSL. Provided an ontology of a high level of abstraction is defined, it can be utilized as metainformation for editing ontologies of lower levels of abstraction. The knowledge engineer also extracts domain knowledge ontology and domain data ontology from the previously defined domain ontology.

3. Provided a domain meta-ontology is defined, it can be utilized as metainformation for acquisition of a corresponding domain ontology from domain experts. Domain knowledge ontology is utilized as metainformation for acquisition of a corresponding domain knowledge from domain experts.

4. Domain data ontology is utilized as metainformation for acquisition of facts of observed reality from domain specialists.

4 CONCLUSIONS

This paper has considered the problem of compatibility of intelligent editing tools that are intended for editing information of different levels of abstraction. A special IDEA Language has been suggested in order to represent information of different levels of abstraction. A unified IDEA Editor that is intended for providing all classes of its users with the necessary intelligent support during the editing process has been introduced in order to solve the problem. A description of a unified editing process has been presented. The proposed IDEA Editor accepts the merits of present approaches, and suggests solutions for the considered problems of the approaches. It also meets the requirements that are stated in (Kleschev and Orlov, 2001) on tools for editing information contents of computer banks of knowledge.

REFERENCES

- Chaudhri, V.K., Farquhar, A., Fikes, R., Karp, P.D. and Rice, J.P., 1998. *Open Knowledge Base Connectivity*, Version 2.0.3. Retrieved January 18, 2000, from <http://www.ai.sri.com/~okbc/spec.html>.
- Corcho, O., Fernandez-Lopez, M., Gomez-Perez, A., 2001. *OntoWeb Technical Roadmap*, Version 1.0. Retrieved March 12, 2002, from <http://www.ontoweb.org>.
- Grosso, W.E., Eriksson, H., Ferguson, R.W., Gennari, J.H., Tu, S.W., and Musen, M.A., 1999. Knowledge modeling at the millennium: The design and evolution of Protégé-2000, Technical Report, SMI-1999-0801, Stanford Medical Informatics, Stanford Univ.
- Gruber, T. R., 1992. Ontolingua: A mechanism to support portable ontologies, Technical Report, KSL-91-66, Knowledge System Laboratory, Stanford Univ.
- Kleschev, A.S., Orlov, V.A., 2001. Requirements on a computer bank of knowledge. In *Proceedings of the Pacific Asian Conference on Intelligent Systems 2001*, Seoul, Korea.
- Meta Object Facility Specification*, Version 1.4, April 2002. Retrieved August 5, 2002, from <http://www.omg.org>.
- Noy, N.F., Ferguson, R.W., Musen, M.A., 2000. The knowledge model of Protégé-2000: combining interoperability and flexibility, Technical Report, SMI-2000-0830, Stanford Medical Informatics, Stanford Univ.
- van Heijst, G., Schreiber, A.Th., Wielinga, B.J., 1996. Using explicit ontologies in KBS development. *International Journal of Human-Computer Studies* 46 (2-3): 183-292