

Hozo: An Ontology building environment

Riichiro Mizoguchi and Kouji Kozaki

*ISIR, Osaka University, 8-1 Mihogaoka, Ibaraki, Osaka, 567-0047 Japan
{kozaki, miz}@ei.sanken.osaka-u.ac.jp*

INTRODUCTION

This demonstration presents how to use Hozo which is a tool for building ontologies in a distributed environment. It has more than 1,000 users in the world and has been used to implement OMNIBUS ontology[3], the world-first heavy-weight ontology of learning and instructional theories. Hozo can be downloaded at <http://www.hozo.jp/> and OMNIBUS at <http://edont.gee.jp/omnibus/doku.php>.

OVERVIEW OF HOZO

Figure 1 shows the block diagram of Hozo[1]. It consists of three major components: Ontology editor, Ontology server and Ontology manager. Onto Studio is specialized to the support of task ontology building. Users build and use ontology through Ontology editor which has a friendly GUI which is popular among users. Ontology manager manages projects in which several ontologies are built collaboratively in a distributed environment through internet. It also manages versions of ontologies. Ontology server stores ontologies and instances and provides APIs for clients.

Among many existing ontology building tools, the main characteristics of Hozo are summarized as follows:

- Unlike OWL, its conceptual level is closer to that of humans.
- OWL is a low level language which is good for an interlingua for ontology exchange. If one uses it as an ontology representation language directly, it would degrade user's understanding of ontology by restricting their idea to semantic-network/description-logics levels which are inappropriate for understanding ontology.
- Single inheritance
- Its representation scheme is based on the frame structure
- It helps users build ontologies with Roles in a natural way supported by the advanced theory of Roles [4].
- It is easy to represent a nested structure of slots. That is, any slot can have its own slots.
- Inheritance information is explicit and is always accessible.
- Two ways of inheritance: one from super classes through *is-a* link and the other from *class constraint*
- A friendly GUI is available
- Version management is available with a useful function for displaying changes.
- Ontology building in a distributed environment over internet is supported.
- APIs are available for accessing ontologies and instances.
- An instance model builder is available with the same GUI of the ontology editor.
It is useful when you build a model of a particular plant using plant ontology.

ROUGH SCHEDULE OF THE DEMO

The demo will be organized as follows:

- Basic operations using common examples
- Making nodes, adding/removing slots by inheritance, etc.
- Inheritance from a super concept

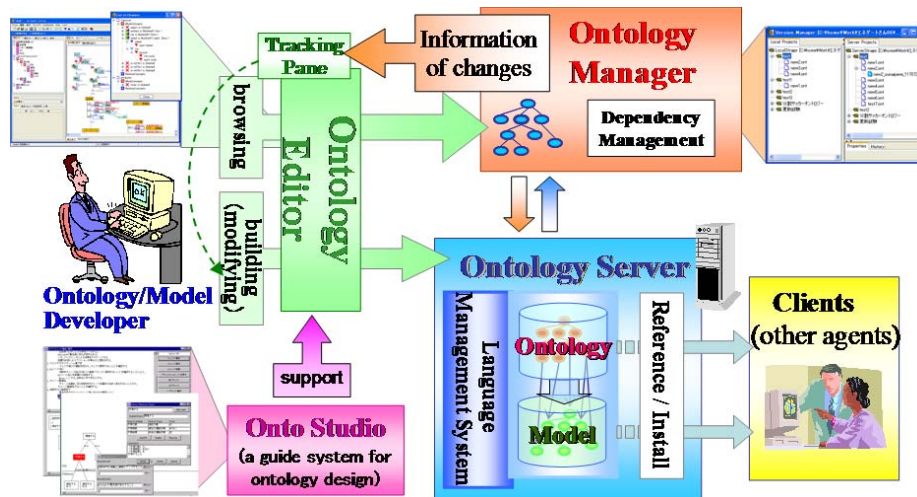


Figure 1: Block diagram of Hozo.

- Introducing the concept of Roles and how to define and use Roles.
- Inheritance from the class constraint(role-player)
- Version management with the display function of changes
- OWL code generation
- Map generation tool
- Exploration of OMNIBUS ontology[3]

CONCLUDING REMARKS

Hozo is appropriate for building a heavy-weight ontology, especially, a philosophically-sound ontology rather than a light-weight ontology. Some people say it is not easy to define Roles. I agree with them. At the same time, however, I would like to stress the importance of training people to be able to cope with Roles, since the world is full of Roles such as teacher, wife, husband, friend, son, president, product, cause, effect, input, fuel, etc. One important heuristic for building a good ontology is to forget about OWL which is a yet another DL language and to learn what an ontology is language-independently, and then use Hozo.

REFERENCES

- [1] <http://www.hozo.jp/>
- [2] [Kozaki 2007] Kouji Kozaki, Eiichi Sunagawa, Yoshinobu Kitamura, Riichiro Mizoguchi, A Framework for Cooperative Ontology Construction Based on Dependency Management of Modules, In Proc. of International Workshop on Emergent Semantics and Ontology Evolution (ESOE2007), Nov. 12, 2007.
- [3] Mizoguchi, R., Hayashi, Y., and Bourdeau, J.: "Inside Theory-Aware and Standards-Compliant Authoring System", Proc. of SWEL'07, pp. 1-18, 2007.
- [4] Mizoguchi R., Sunagawa E., Kozaki K. and Kitamura Y., A Model of Roles within an Ontology Development Tool: Hozo, J. of Applied Ontology, Vol.2, No.2, pp.159-179. Sep. 2007.

PépiGen: a Multi-criteria Assessment Authoring Tool

Élisabeth Delozanne¹, Dominique Prévité², Brigitte Grugeon³, Françoise Chenevotot³

¹*L'UTES - Université Paris VI - 4 pl. Jussieu - 75005 PARIS – France,
elisabeth.delozanne@upmc.fr*

²*LIUM-Avenue Laennec 72085 Le Mans Cedex 9– France, dominique.previt@bretagne.iufm.fr*

³*DIDIREM - Université Paris VII - 2 place Jussieu - 75251 PARIS Cedex 5 – France,
brigitte.grugeon@amiens.iufm.fr, francoise.chenevotot@lille.iufm.fr*

Abstract. In this demo we present PépiGen a prototype to author assessments of complex competence involving open-ended questions. PépiGen is based on our previous work on Pépite, an automatic cognitive diagnosis tool that capitalizes on educational research results. To adapt an assessment to a specific classroom context (e.g. level of difficulty, time, learning objectives) an interface allows an IT non expert (e.g. a teacher) to generate new instances of exercises by filling parameters in a pattern. The originality of our research lies in the fact that our system generates the automatic analysis of students' simple or complex answers, such as algebraic reasoning. This is an ongoing work but preliminary evaluation shows that PépiGen is already successful in generating and analyzing most answers on several classes of problems.

We developed a cognitive diagnosing tool, derived from Educational Research [6], called Pépite. In the present stage of the project, the aim is to offer an authoring tool, called PépiGen, to generate different Pépite-like diagnosis tools adapted to different school contexts and teachers' objectives.

We focussed on the following design scenario: a teacher chooses a prototypic exercise in the bank and, if need be, asks for another equivalent one retrieved from the bank, or adapts the statement of the exercises by filling in forms. We will demonstrate how PépiGen allows an author to adapt a pattern of exercise.

PépiGen provides a Graphical Interface to enter the parameters. The author enters one parameter (e.g. the statement in natural language), and PépiGen generates the other parameters (e.g. the corresponding global algebraic expression and its reduced form). A software component based on a grammar and a finite state machine is used to interpret users' input in a constrained natural language and to translate it into algebra. This component is also used for analysis of students' input in other diagnosis tasks.

When parameters are set, a procedure specific to the pattern is called by PépiGen, to automatically generate all the information necessary to diagnose the students' answers to the exercise. This procedure is simple when answers are preformatted. In case of open-ended questions involving the dimensions "Algebraic calculation" or "Numerical calculation" in the pattern description, a software component, called Pépinière, builds a tree representing all anticipated solutions to the exercise and codes each solution on several dimensions.

PépiGen is implemented in Java. It creates, initializes and saves, in an XML database, instances of the different classes representing the dynamic part of a pattern of diagnostic tasks (Table 1).

Table 1: Extract of a file generated to describe the diagnostic part of the clone for one exercise

<Correct solution>
