Developing an Easy Tool for Ontology Integration

Hai-Tao Zheng
School of Economics and Commerce, Dankook University
quicklyfly@hotmail.com

Hong-Gee Kim
School of Economics and Commerce, Dankook University
hgkim@dku.edu

Abstract

When the ontology becomes more complex, the need arises to modularize ontologies. Building an application ontology often requires integration of reusable external ontologies that are modularized. We developed an easy tool for ontology integration, named SOM(Simple Ontology Merger) that can import parts of classes from a large (or small)-scale external ontology. SOM can specify a new relationship between classes from two different ontologies. SOM can modify some values of the properties from the classes imported from the external ontology.

Keywords: ontology tool, ontology integration

Introduction

The OWL Web Ontology Language is intended to provide a language that can be used to describe the classes and relations between them that are inherent in Web documents and applications.[4][5]. Ontologies today are available in many different forms: as artifacts of a tedious knowledge-engineering process, as information that was extracted automatically from informal electronic sources, or as simple “light-weight” ontologies that specify semantic relationships among resources available on the World-Wide Web [1].

But what does a user do when he finds several ontologies that he would like to use but these ontologies are so large and complicated? When the ontology becomes more complex, the need arises to modularize ontologies. Building an application ontology often requires integration of reusable external ontologies that are modularized. We developed an easy tool for ontology integration, named SOM(Simple Ontology Merger) that can import parts of classes from a large (or small)-scale external ontology. SOM can specify a new relationship between classes from two different ontologies. SOM can modify some values of the properties from the classes imported from the external ontology.

SOM is a widget plug-in application built upon Protégé 2.0 OWL plug0in tab. Protégé is an ontology-design and knowledge acquisition tool with an OKBC-compatible [2] knowledge model,

which allows domain experts (and not necessarily knowledge engineers) to develop ontologies and perform knowledge acquisition. Protégé is a tool which allows the user to construct a domain ontology; to customize data entry forms; to enter data; Protégé is a platform which can be extended with graphical widgets for tables, diagrams, animation components to access other knowledge-based systems embedded applications; Protégé is a library which other applications can use to access and display knowledge bases.

Knowledge Model

Now we discuss some definitions about the ontology and SOM. We start with the description of the knowledge model underlying SOM. The knowledge model is frame-based and it is designed to be compatible with OKBC [2]. At the top level, there are classes , slots, facets, and instances.

- **Classes** are collections of objects that have similar properties. Classes are arranged into a subclass–superclass hierarchy with a multiple inheritance. Each class has slots attached to it. Slots are inherited by the subclasses.
- **Slots** are named binary relations between a class and either another class or a primitive object (such as a string or a number). Slots attached to a class may be further constrained by facets. Hence, we can call Slots as properties.
- **Facets** are named ternary relations between a class, a slot, and either another class or a primitive object. Facets may impose additional constraints on a slot attached to a class, such as the cardinality or value type of a slot. In fact, Facets is the values of the properties.
- **Instances** are individual members of classes. These definitions are the only restrictions that we impose on the input ontologies for SOM.

Since this knowledge model is extremely general, and many existing knowledge representation systems have knowledge models compatible with it, the solutions to merging produced by SOM can be applied over a variety of knowledge representation systems.

Why SOM?

In the world things in one domain are often conceptualized in the way that they are related to other things in a different domain. For example, the car as a transportation system can be understood better if it is described in terms of its relationship with some mechanical concepts of engine parts that are understood in the domain of mechatronics. Of course, a user can start building an ontology of transportation in that the ‘Car’ class has its own slots and facets to describe itself. Specifying properties of the ‘Engine’ class as a part of car, the user might want to import some part of mechanical concepts regarding the ‘Engine’ class from an external ontology. The concept of engine might be related to another external ontology that describes the concept of fuel.

Most transportation systems have engines that are operated differently. A submarine engine might be different from a car engine or a airplane engine in terms of its fuel system and the mechanical principle. How can we represent the differences? If there is only one type of engine in the original machine ontology, several subclasses of engine should be created when this concept is associated with the ontology of transportation system. In our example, SOM automatically creates the new subclasses such as 'car-engine' and 'submarine-engine' for the user.

There might be many different ways of reusing classes and their properties from external ontologies. We can simply include some useful classes by copying them to the ontology at work. SOM not only copy the subclasses but copy all of its slots and facets. If there is a subclass which is the same as the subclass to be copied, then the subclass will be copied as a multi-inherited subclass. SOM can also help the user to input basically whatever he wants to specify.

The SOM Algorithm

As shown in figure 1, firstly, SOM shows the original ontology that the user wants to specify, including all the classes and the related slots. Secondly the user can import an external ontology file for merging. After importing the external ontology, all the classes of the external ontology will be shown, and the user can choose one of the classes to specify the chosen slot in the original file; As soon as choosing the class, all the allowed slots of the class will be shown, and the facets related to the slots can be selected too. Thirdly, the user should confirm that he has chosen the class and the related slots in the original ontology he wants to specify and make sure that he has provided the facet of the slot which can be obtained from the external ontology. Otherwise, the required information of SOM will be insufficient.

When the user has chosen all the conditions for SOM, SOM can begin the process of ontology integration. That is, SOM will help the user extend any class in the working ontology by adding relationship with a class and properties from the external ontology.

Given the ontology of transportation system, in our toy example, there are four classes including 'TransprotTools' which is the upper most class and its subclasses: 'Car', 'Plane', and 'Submarine'. And there is one slot named hasPart for every class that is inherited from the 'TransportTools' class.

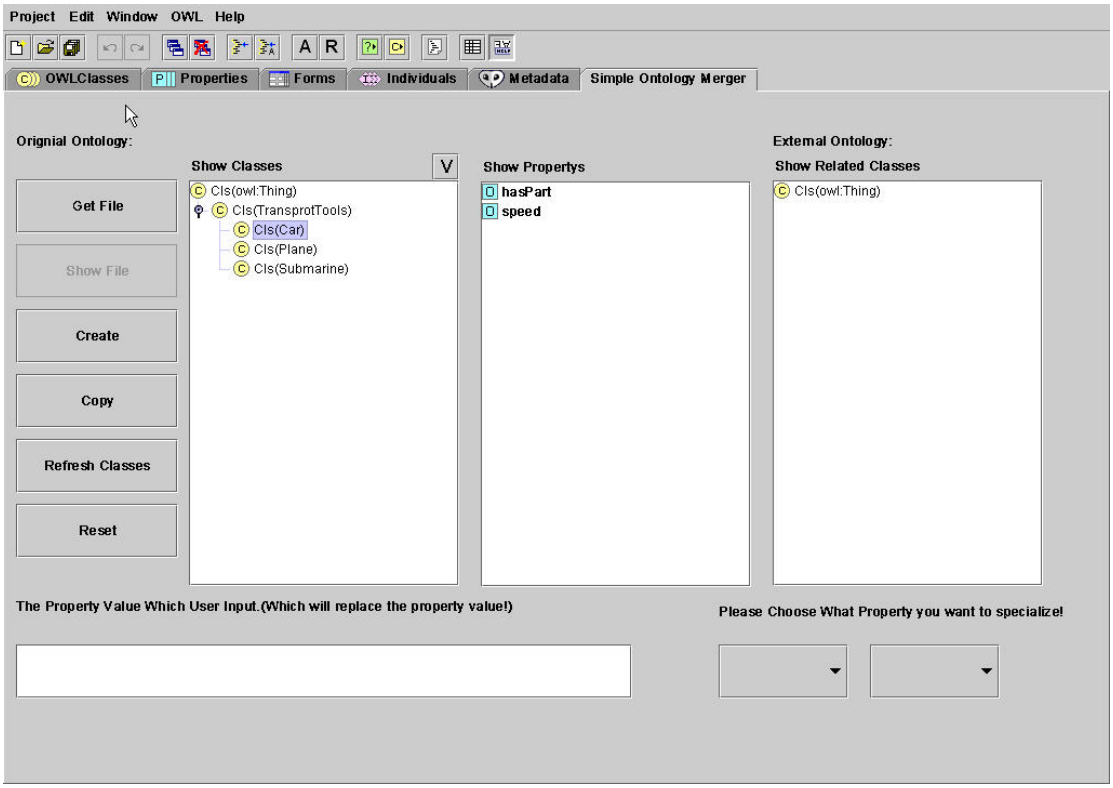

Figure 1.Showing the original ontology file

Now we want to extend the 'Car' class into the one having an engine by specifying the 'hasPart' slot of the 'Car' class. We import the external ontology concerning 'machine' that contains the knowledge of 'engine'.

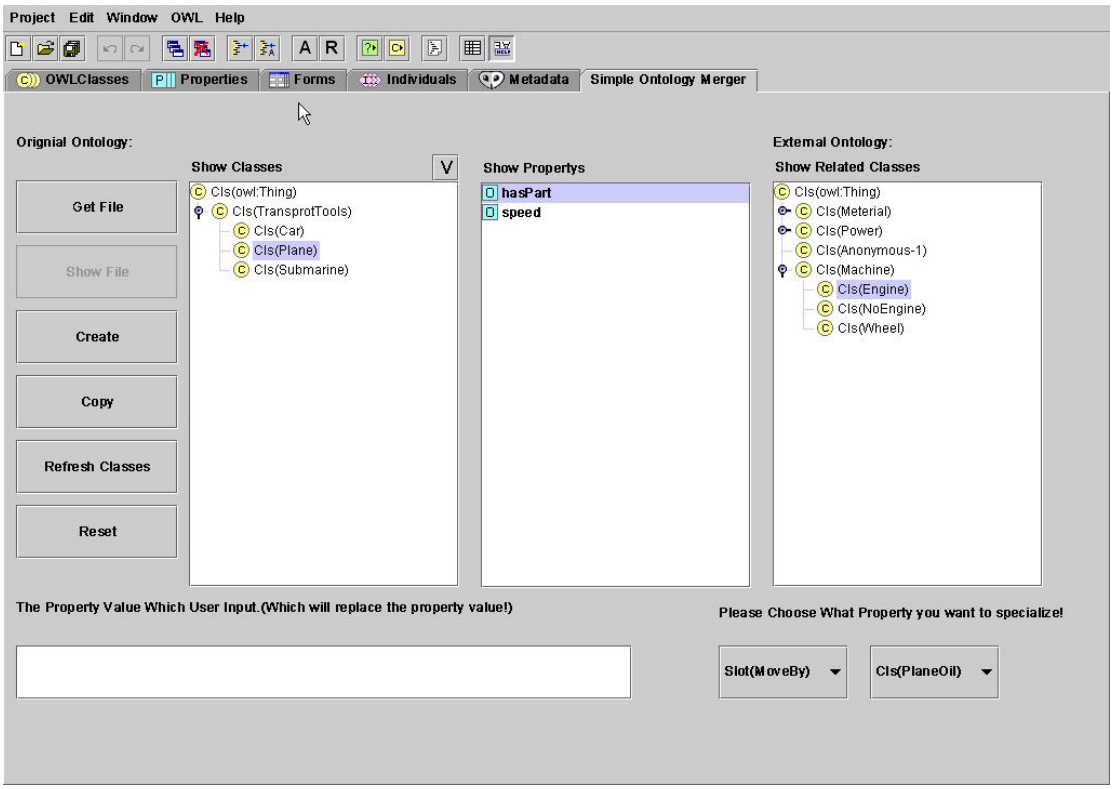

Figure 2.Showing the external ontology file

The engine has slots including MakeOf and MoveBy. And when we select the ‘moveBy’ slot, one of four slot values can be specified: namely, CarOil, PlaneOil, GoodOil, normalOil. Because the engine is used for the car, we select the CarOil for the engine. Then, we can opertate the ontology simple merger. At last, SOM will create the CarEngine for the car class; and the CarEngine is MoveBy the CarOil.

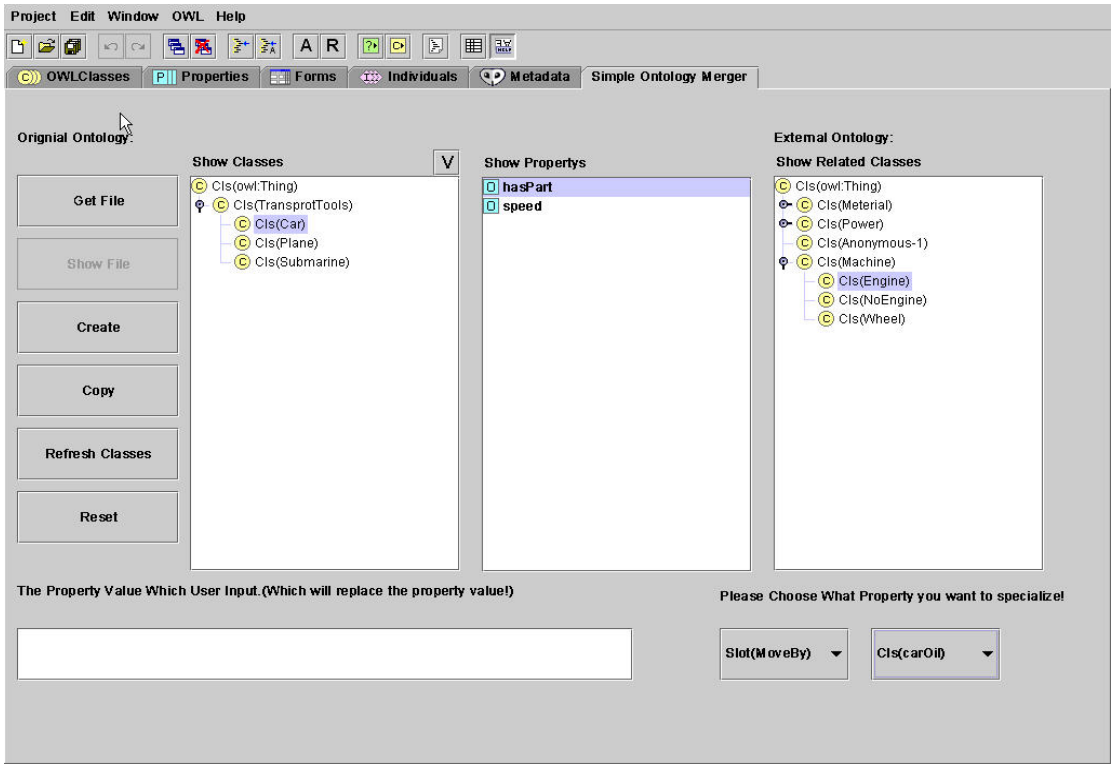

Figure 3.Showing the merging condition

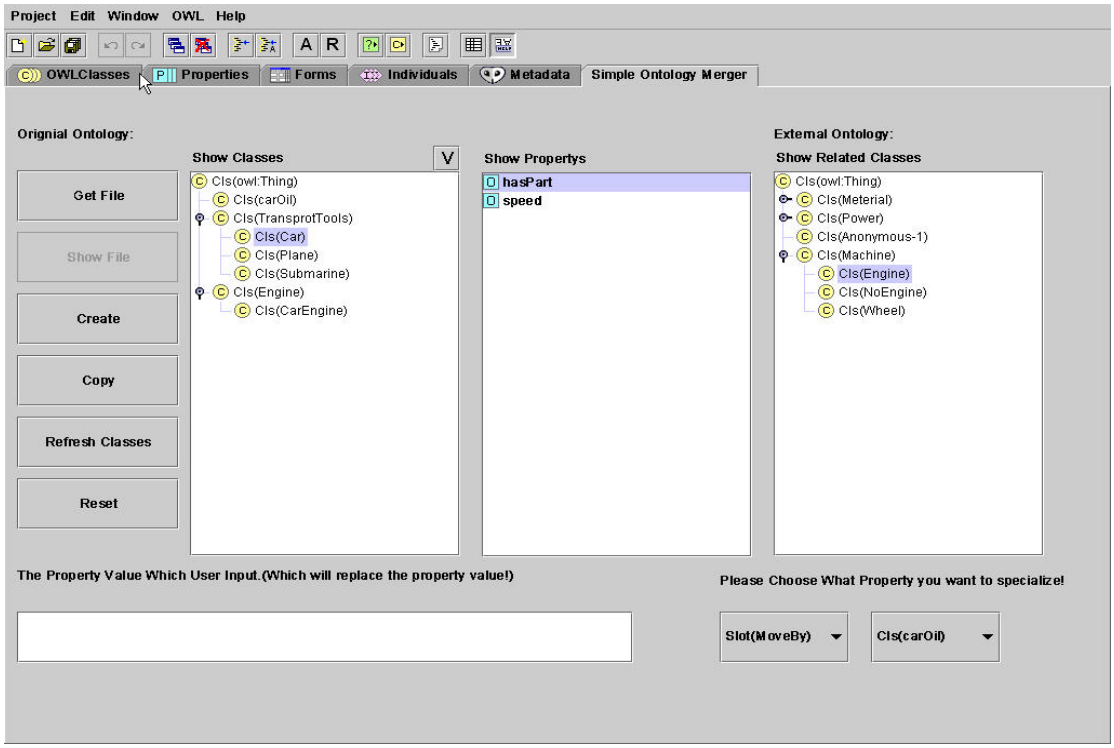

Figure 4.Showing the classes of merging result

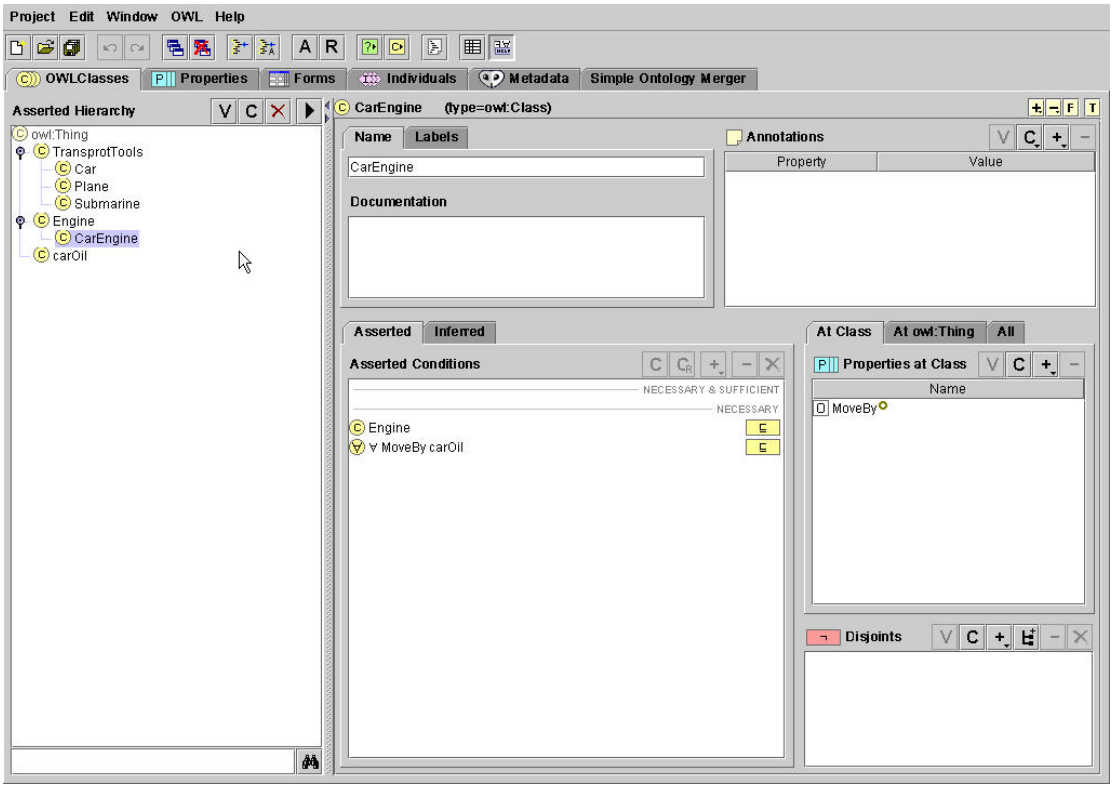

Figure 5.Showing new property of ontology file

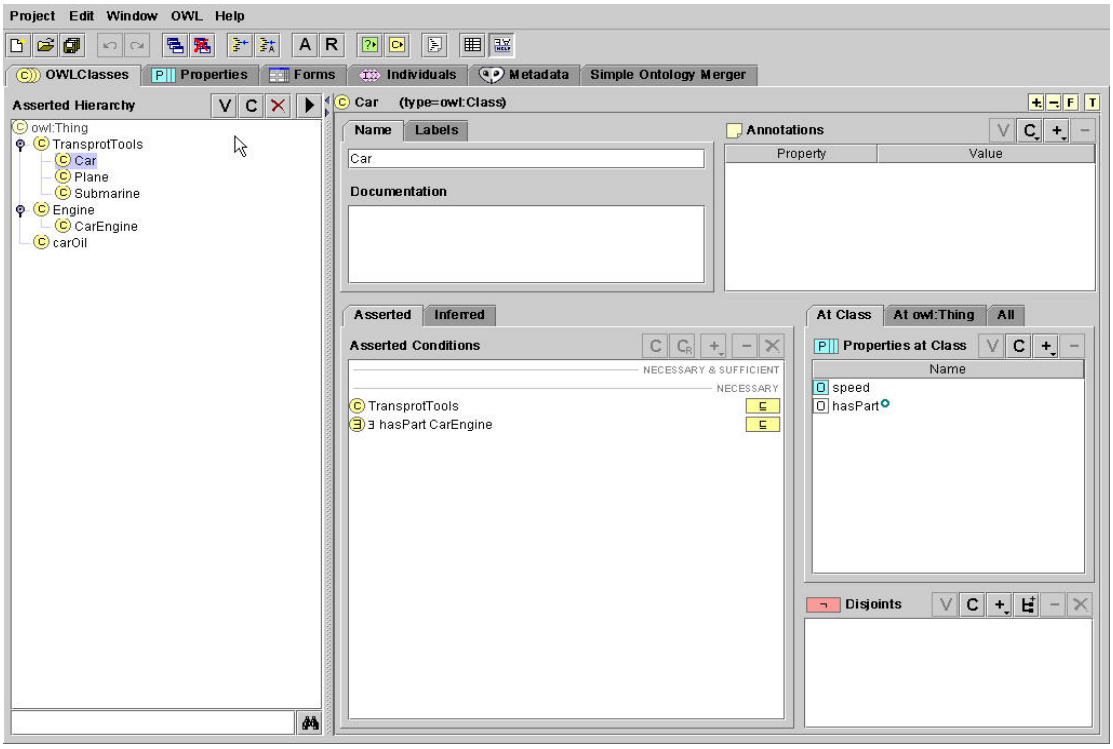

Figure 6.Showing the new property value of ontology file

Figure1 to Figure 6 show how SOM help the user include, extend, and specify concepts in an ontology. Through the merging process, the original ontology will be specified according to the requirement of the user.

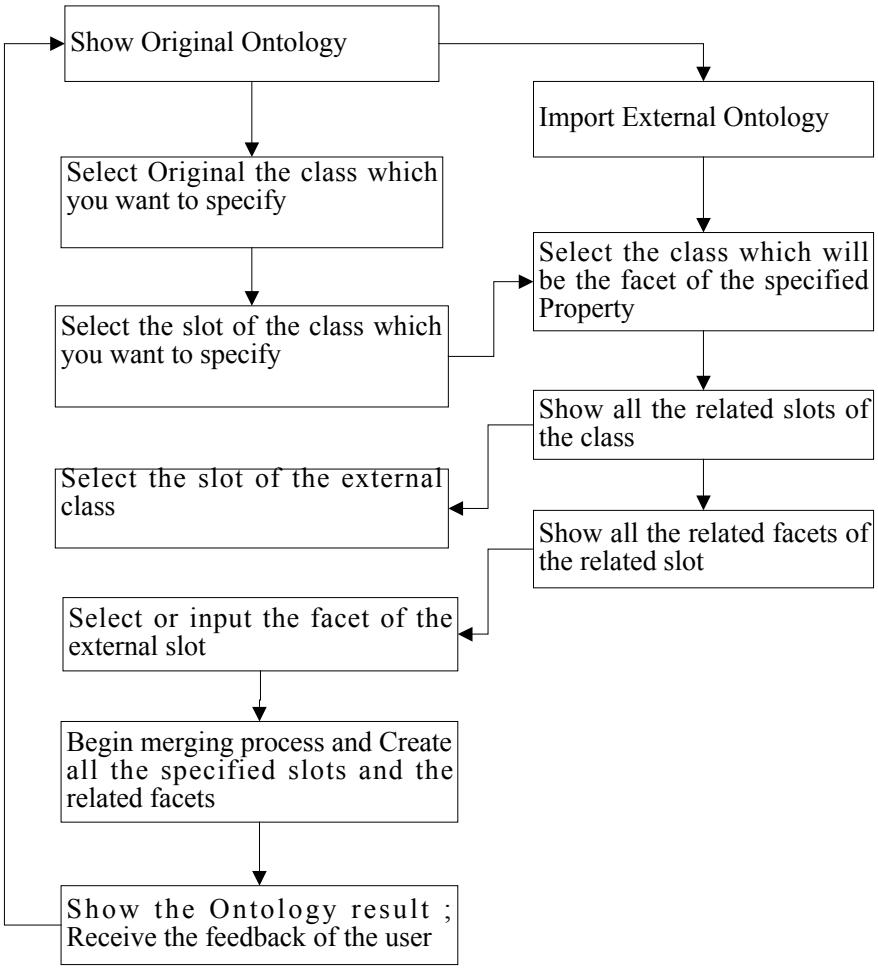

Figure 7 the algorithm of the Simple Ontology Merger

The Figure7 shows the general process of how the SOM works. The user can input the specified facet for the slot which is related with the class imported from the external ontology file.

PROMPT versus SOM

PROMPT that is the most well known Protégé plug-in tool for ontology merge provides an algorithm that provides a semi-automatic approach to ontology merging and alignment. PROMPT performs some tasks automatically and guides the user in performing other tasks for which his intervention is required. PROMPT also determines possible inconsistencies in the state of the ontology, which result from the user's actions, and suggests ways to remedy these inconsistencies. PROMPT is based on an extremely general knowledge model and therefore can be applied across various platforms. Our formative evaluation showed that a human expert followed 90% of the suggestions that PROMPT generated and that 74% of the total knowledge-base operations invoked by the user were suggested by PROMPT.[3]

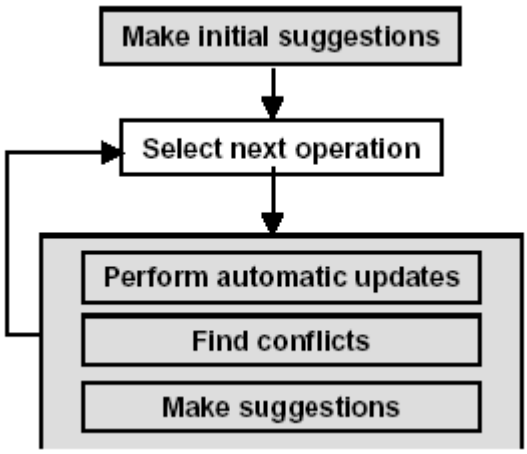

Figure 8 shows the data flow of PROMPT algorithm. The gray boxes indicate the actions performed by PROMPT. The white box indicates the action performed by the user.

SOM also provides a semi-automatic approach to merging the ontology. Compared to PROMPT tab, our protégé tab provides a more flexible merging method. The PROMPT tab is responsible for the whole ontology file's merging. When the user merge two ontology files, PROMPT will process the inference and tell the user which part he should process to merge. But sometimes the user just wants to specify only some properties of a class by using external ontology or giving what the value he wants to the properties, or just simply copy some classes to from the external ontology to the original ontology .In this situation, PROMPT is too complicated for the user and maybe he will be confused and can't specify the original ontology in the way he expect .

6. Conclusion

We have described a general approach to constructing a complicated ontology through an external ontology. We developed an easy tool for ontology integration, named SOM(Simple Ontology Merger) that can import parts of classes from a large (or small)-scale external ontology. SOM can specify a new relationship between classes from two different ontologies. SOM can modify some values of the properties from the classes imported from the external ontology. The strategies and algorithms described in this paper are based on a general OKBC-compliant knowledge model. Therefore, these results are applicable to a wide range of knowledge representation and ontology-development systems. We extended Protégé knowledge-modeling environment with a tool based on the algorithm and performed an empirical evaluation of the tool. Our results showed that SOM was very effective in helping researcher to construct his own ontology.

Reference

[1] Brickley, D. and Guha, R.V. (1999). Resource Description Framework (RDF) Schema Specification. Proposed Recommendation, World Wide Web Consortium**:** http://www.w3.org/TR/PR-rdf-schema.

[2] Chaudhri, V.K., Farquhar, A., Fikes, R., Karp, P.D. and Rice, J.P. (1998). OKBC: A Programmatic Foundation for Knowledge Base Interoperability. In: *Proceedings of theFifteenth National Conference on Artificial Intelligence(AAAI-98)*, Madison, Wisconsin, AAAI Press.

[3] Natalya Fridman Noy and Mark A. Musen, "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment"

[4] Michael K. Smith, Electronic Data Systems, Chris Welty, IBM Research, Deborah L. McGuinness, Stanford University, "OWL Web Ontology Language Guide",W3C Proposed Recommendation 15 December 2003:http://www.w3.org/TR/2003/PR-owl-guide-20031215/

[5] Deborah L. McGuinness (Knowledge Systems Laboratory, Stanford University) Frank van Harmelen (Vrije Universiteit, Amsterdam) ,"OWL Web Ontology Language Overview "W3C Proposed Recommendation 15 December 2003,http://www.w3.org/TR/2003/PR-owl-features-20031215/