Z Approach to Semantic Web

Jin Song Dong, Jing Sun, and Hai Wang

School of Computing,
National University of Singapore,
{dongjs,sunjing,wangh}@comp.nus.edu.sg

Abstract. The Semantic Web (SW) service is a web application using Semantic Web techniques which usually involve cooperation between several intelligent agents. The design of SW systems requires precise modelling techniques to capture ontology domain properties and application functionalities. We believe that Z as a specification technique can contribute to the Semantic Web-based system development in many ways. In this paper, we firstly conduct a case study of applying Z to the design of a SW service system (online talk discovery), and then present translation techniques and tools which can extract the SW ontology from the Z model automatically. Furthermore, we discuss how existing Z tools, i.e. Z/EVES, can be used to improve the quality of SW ontology design.

Keywords: Z, Semantic Web

1 Introduction

In recent years, researchers have begun to explore the potential of associating web content with explicit meaning so that the web content becomes more machine-readable and intelligent agents can retrieve and manipulate pertinent information readily. The Semantic Web (SW) [3] proposed by W3C is one of the most promising and accepted approaches. It has been regarded as the next generation of the web. The Semantic Web service is a web application using Semantic Web techniques which usually involve cooperation between several intelligent agents. Some Semantic Web services have been successfully developed recently, e.g. ITTAKLS [2].

The development of Semantic Web systems requires precise modelling techniques to capture ontology domain properties and application functionalities. The Z notation[12] is a formal specification language based on set theory and predicate calculus. We believe that Z as a specification technique can contribute to the Semantic Web-based system development in many ways. In this paper, we take a Semantic Web service example, i.e. the online talk discovery system, and apply Z to the design this Semantic Web service system. This online talk discovery system is a simplified version of the ITTALKS system [2] which is a real life Semantic Web service case study.

The remainder of the paper is organized as follows. Section 2 briefly introduces the Semantic Web. Section 3 formally specifies the functionalities of the Semantic Web service example (Talks discovery system). Section 4 presents the tools which extract the ontology used by the SW service from the Z design model automatically. Section 5 discusses how Z tools can be used to improve the quality of Semantic Web design. Section 6 concludes the paper.

C. George and H. Miao (Eds.): ICFEM 2002, LNCS 2495, pp. 156–167, 2002.

2 Semantic Web Overview

The Semantic Web is a series of technologies proposed by W3C as the next generation web. It extends the current one by giving the web content a well-defined meaning, better enabling computers and people to work in cooperation. HTML, the current Web data standard, is aimed at delivering information to the end user for human-consumption (e.g. display this document). XML is aimed at delivering data to systems that can understand and interpret the information. XML is focused on the syntax (defined by the XML schema or DTD) of a document and it provides essentially a mechanism to declare and use simple data structures. However there is no way for a program to actually understand the knowledge contained in the XML documents.

Resource Description Framework (RDF) [7] is a foundation for processing metadata; it provides interoperability between applications that exchange machine-understandable information on the Web. RDF uses XML to exchange descriptions of Web resources and emphasizes facilities to enable automated processing. The RDF descriptions provide a simple ontology system to support the exchange of knowledge and semantic information on the Web. RDF Schema [4] provide the basic vocabulary to describe RDF vocabularies. RDF Schema can be used to define properties and types of the web resources. Similar to XML Schema which give specific constraints on the structure of an XML document, RDF Schema provide information about the interpretation of the RDF statements. The DARPA Agent Markup Language (DAML) [10] is a semantic markup language based on RDF/RDF-Schema and XML for Web resources. DAML currently combines Ontology Interchange Language (OIL) [5] and features from other ontology systems. It is now called DAML+OIL (DAML for short) and contains richer modeling primitives. DAML+OIL can dramatically improve traditional ad hoc information retrieval because its semantics will improve the quality of retrieval results. Semantic Web is highly distributed, and different parties may have different understanding for the same concept. Ideally, the program must have a way to discover the common meanings from the different understandings. It is central to another important conception in Semantic Web service – ontology. The ontology for a Semantic Web service is a document or file that formally defines the relations among terms. The most typical kind of ontology for the Web has a taxonomy and a set of inference rules. Ontologies can enhance the functioning of the Web in many ways, and RDFS and DAML supply the language to define the ontologies. For example, the following DAML code specifies that a ‘talk’ (a DAML class) has a property ‘talk_place’, having only one value ‘place’ (also a DAML class).

```
<daml:class rdf:ID="talk"> <rdfs:label>Talk</rdfs:label></daml:class>
<daml:class rdf:ID="place"><rdfs:label>Place</rdfs:label></daml:class>
<daml:ObjectProperty rdf:ID="talk_place">
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
  <rdf:domain rdf:resource="#talk"/><rdf:range rdf:resource="#place"/>
</daml:ObjectProperty>
```

We will use the following notations to summarize the DAML constructs:

Table 1. A Partial Summary of the DAML constructs

Abstract DAML constructs	**Description**
daml_class	classes
daml_subclass$[C]$	subclasses of C
daml_objectproperty$[D \leftrightarrow R]$	relation properties with domain D, range R
daml_objectproperty$[D \rightarrow R]$	function properties with domain D, range R
daml_subproperty$[P]$	sub properties of P
instanceof $[C]$	instances of the DAML class C

3 The Talks Discovery System

In this section, an online talks discovery system is used as an example to demonstrate how Z notation can be applied to the Semantic Web service development.

3.1 System Scenario

The Talks Discovery system is a web portal offering access to information about talks, seminars. This web portal can provide not only the talk's information corresponding to the user's profile in terms of his interest and location constraints, but also can further filter the IT related talks based on information about the user's personal schedule, etc.

In the course of operation, the Talks Discovery system discovers that there is an upcoming talk that may interest a registered user based on information in the user's preferences, which have been obtained from his online, DAML-encoded profile. Upon receiving this information, the user's User Agent needs to know more; it consults with its Calendar agent to determine the user's availability, and with the MapQuest agent to find the distance from the user's office to the talk's venue. We assume that a user only wants to attend the talks located within five miles from his office. Finally, after evaluating the information and making this decision, the User Agent will send a notification back to the TALK discovery agent indicating that the user will/will not plan to attend. The completed functionality of the ITTALKS system can be found at http://www.ittalks.org/jsp/Controller.jsp.

3.2 Formal Models of the Talk Discovery System

The system involves four different intelligent agents which communicated interactively. They are the user's Calendar agent, MapQuest agent, user's personal agent, and Talks discovery agent.

Calendar Agent. Firstly, the Date and Time set are defined by the Z given type definitions. As this paper focuses only on demonstrating the approach, we try to make the

model simple. Z given type is chosen to define TIME, DATE and some other conceptions. These conceptions can be subdivided into detailed components, e.g. the TIME comprises hour, minute, and second. The more detailed the model is, the more detailed ontology derived automatically from our tool. This tool will be further discussed in the later section.

The *DateTime* was defined as a schema with two attributes date and time.

$$[TIME, DATE]$$

DateTime

$$date : Date; \quad time : Time$$

Each user has its own Calendar agent which maintains the user's schedule and supplies some related services.

The *status* defined by the Z free type definition indicates if a person is free or busy.

$$status ::= FREE \mid BUSY$$

Calendar

$$timetable : DateTime \rightarrow status$$

Update is used to update the timetable. The operation *Check_free* is used to check whether a person is available or not for a particular time slot.

Update

$$\Delta Calendar$$
$$t? : DateTime; \quad s? : status$$
$$timetable' = timetable \oplus \{(t?, s?)\}$$

Check_free

$$\Xi Calendar$$
$$dt? : DateTime$$
$$timetable(dt?) = FREE$$

MapQuest Agent. MapQuest agent is a third party agent supplying the service for calculating the distance between two places.

Firstly, the *place* is defined as a Z given type. The MapQuest agent contains a set of places in its domain and a database storing the distance between any two places.

$$[PLACE]$$

MapQuest

$$places : \mathbb{P}\, PLACE$$
$$distance : places \times places \rightarrow \mathbb{R}^+$$

Operation *Get_dis* will output the distance between two places.

Get_dis

$$\Xi Map$$
$$p_1?, p_2? : places$$
$$dis! : \mathbb{R}^+$$
$$distance(p_1?, p_2?) = dis!$$

Near

$$dis? : \mathbb{R}^+$$
$$dis? < 5$$

$$Check\_near \mathrel{\widehat{=}} Get\_dis \gg Near$$

In our system we assume that a user only wants to attend the talks located within five miles from his office. The schema*Near* and *Check_near* will be used to ensure a talk is held within the desired range.

Personal Agent. The personal agent keeps the user's profile including user's name, office location, interesting etc.

$[NAME, SUBJECT]$

Person

$name : NAME$
$office : PLACE$
$interests : \mathbb{P}\, SUBJECT$

Operation *Get_office* will output the user's office place. The personal agent uses operations *Talks_time* and *Free* to communicate with his calendar agent to check whether the user is free or not.

Get_office

$\Xi Person$
$o! : PLACE$

$o! = office$

Talks_time

$tk? : Talk$
$dt! : DateTime$

$tk?.dt = dt!$

$$Free \mathrel{\widehat{=}} Talk\_time \gg Check\_free$$

The personal agent use operations *Distance* and *CheckNear* to communicate with the MapQuest agent to ensure the talks will be held nearby.

Distance

$tk? : Talk;\ \ p_1! : Place$

$tk?.place = p_1!$

$$CheckNear \mathrel{\widehat{=}} (Distance \wedge Get\_office[p_2!/o!]) \gg Check\_near$$

The personal agent will notify the Talks discovery system if the client will attend the talk.

$[NOTIFY]$

Sendnotify

$no! : NOTIFY$

Talks Discovery Agent. Schema *Talk* is defined for a general talk type. *interested_talks* records the interested talks for the users. The Schema *Talks* records a set of talks and users in the database.

Talk

$place : PLACE$
$dt : DateTime$
$subject : \mathbb{P}\, SUBJECT$

$later : DateTime \leftrightarrow DateTime$

$interested\_talks : Person \leftrightarrow Talk$

Talks

$talks : \mathbb{P}_1\, Talk$
$users : \mathbb{P}_1\, Person$

The relation *later* determines which *DataTime* happens first. The function *find_nearest* finds the next talk among a set of talks.

$$
\begin{array}{|l}
\hline
find\_nearest : \mathbb{P}_1\ Talk \rightarrow Talk \\
\hline
\forall\, tks : \mathbb{P}_1\ Talk; tk : Talk \bullet \\
\quad tk \in tks \land \nexists tk_1 : tks - \{tk\} \bullet (tk_1.dt, tk.dt) \in later \\
\quad find\_nearest(tks) = tk \\
\hline
\end{array}
$$

The operation *Find_interested* will find the upcoming talks in which a user has interest.

$$
\begin{array}{|l}
\hline
Find\_interested \\
\hline
\Xi Talks \\
u? : users;\ \ tk! : talks \\
\hline
tk! = find\_nearest(interested\_talks(\!|\ \{u?\}\ |\!)) \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
Acceptnotify \\
\hline
no? : NOTIFY \\
\hline
\end{array}
$$

$$System \mathrel{\widehat=} (Find\_interested \gg (Free \land CheckNear \land sendnotify)) \gg Acceptnotify$$

The number of instances can be created also.

$$
\begin{array}{|l}
\hline
National\_University\_Singapore : Place;\ \ atalk : Talk \\
\hline
atalk.place = National\_University\_Singapore \\
\ldots \\
\hline
\end{array}
$$

4 Extracting DAML Ontology from the Z Model

It is important to have a thoroughly designed ontology since it will be shared by different agents and it forms the foundation of all agents' service. However designing a clear and consistent ontology is not a trivial job. It is useful to have some tool support in designing the ontology.

In this section, we will demonstrate the development of an XSL [11] program to automatically extract the ontology related domain properties from Z formal models (encoded in ZML format[9]) to DAML. The ontology for the system can be resolved readily from the static parts of Z design documents.

ZML is an XML environment for Z family notations. It encodes the Z documents in XML format so that the Z model can be easily browsed by the Internet Explore 5. For example, in ZML the schema *Get_office* defined n section 3 will be encoded as following:

```
<schemadef layout="simpl" align="left">
  <name>Get_o ffice</name><xi><type>Person</type></xi>
  <decl><name>o?</name><dtype><type>PLACE</type></dtype>
  </decl><st/><predicate>o? = office</predicate></schemadef>
```

This XML file can be manually created by any XML editor. A tool to automatically transform the Z model in LaTeX format into XML format is currently under development. The eXtensible Stylesheet Language (XSL) [11] is a stylesheet language to describe rules for matching and transforming XML documents. In our case we transform the ZML to DAML. A set of transformation rules transforming from Z model (in ZML) to DAML ontology are developed in the following presentation.

4.1 Given Type Transformation

The given types in the Z model are directly transformed into DAML classes. This transformation can be expressed as the following rule:

$$\frac{[T]}{T \in daml\_class}$$

For example, given type *TIME* can be transformed into two classes in DAML with *time* and *date* as ID.

```
<daml:class rdf:ID="time"> <rdfs:label>TIME</rdfs:label> </daml:Class>
```

4.2 Z Axiomatic (Function and Relation) Definition Transformation

The transformation from functions and relations in Z to DAML ontology requires several cases.

$$\frac{\begin{array}{|l} R : B \leftrightarrow (\rightarrow, \twoheadrightarrow) C \\ \hline \ldots \end{array} \qquad B, C \in daml\_class}{R \in daml\_objectproperty[B \leftrightarrow (\rightarrow, \twoheadrightarrow) C]}$$

The relation R will be transformed into a DAML property with B as the domain class and C as the range class. For total functions we restrict the $daml : cardinality$ property to be one and for partial functions we restrict the $daml : maxCardinality$ property to be one.

In our Talks Discovery example, the relation *interested_talks* can be transformed into DAML as:

```
<daml:ObjectProperty rdf:ID="interested_talks">
 <rdfs:domain rdf:resource="#person"/><rdfs:range rdf:resource="#talk"/>
</daml:ObjectProperty>
```

4.3 Z Axiomatic (Subset and Constant) Definition Transformation

Subset. In this situation, if N corresponds to a DAML class, then M will be transformed into a DAML subclass of N. If N corresponds to a DAML property, then M will be transformed into a DAML subproperty of N. The transformation rules for the subsets are:

$$\frac{\begin{array}{|l} M : \mathbb{P}\, N \\ \hline \ldots \end{array} \qquad N \in daml\_class}{M \in daml\_subclass[N]} \qquad \frac{\begin{array}{|l} M : \mathbb{P}\, N \\ \hline \ldots \end{array} \qquad N \in daml\_objectproperty}{M \in daml\_subclass[N]}$$

Constant. In this situation, X will be transformed into an instance of Y. The following is the transformation rule:

$$\frac{x : Y}{\ldots} \qquad Y \in daml\_class$$

$$x \in instantceof\,[Y]$$

For example, the *National_University_Singapore* and *atalk* defined in section 3 can be transformed to

```
<place rdf:ID="National_University_Singapore"/>
<talk rdf:ID="atalk">
  <rdfs:label>atalk</rdfs:label>
  <talk_place  rdf:resource="#National_University_Singapore"/> ...</talk>
```

4.4 Z Schema Transformation

A Z state schema can be transformed into a DAML class. Its attributes are transformed into DAML properties with the schema name as domain DAML class and the Z type declaration as range DAML class. In order to resolve the name conflict between the same attribute names used in different schemas we use the schema name appended with attribute name as the ID for the DAML property.

$$\frac{S \;\; X : T_1;\;\; Y : \mathbb{P}\, T_2}{\ldots} \qquad T_1, T_2 \in daml\_class$$

$$S \in daml\_class,\; X \in daml\_objectproperty[S \rightarrow T_1], Y \in daml\_objectproperty[S \leftrightarrow T_2]$$

For example the *Talk* schema defined in previous section can be transformed to DAML as

```
<daml:classrdf:ID="talk"> <rdfs:label>Talk</rdfs:label></daml:Class>
<daml:ObjectProperty rdf:ID="talk_place">
 <rdf:type rdf:resource=" http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
 <rdf:domain rdf:resource="#talk"/> <rdf:range rdf:resource="#place"/>
</daml:ObjectProperty>  ...
<daml:ObjectProperty rdf:ID="talk_subject">
 <rdf:domain rdf:resource="#talk"/><rdf:range rdf:resource="#subject"/>
</daml:ObjectProperty>
```

Other transformation rules are omitted as the aim of this paper is to demonstrate the approach rather than providing the complete XSL program design.

5 Improve the Ontology Quality through Z Tools

In the previous section, we discussed a tool which can extract the DAML ontology automatically from the Z model. However if the Z model itself contains some flaws, the inconsistences will be brought into the ontology also. Fortunately Z has various semi-automatic reasoning and checking tools. A number of tools, including a type checker,

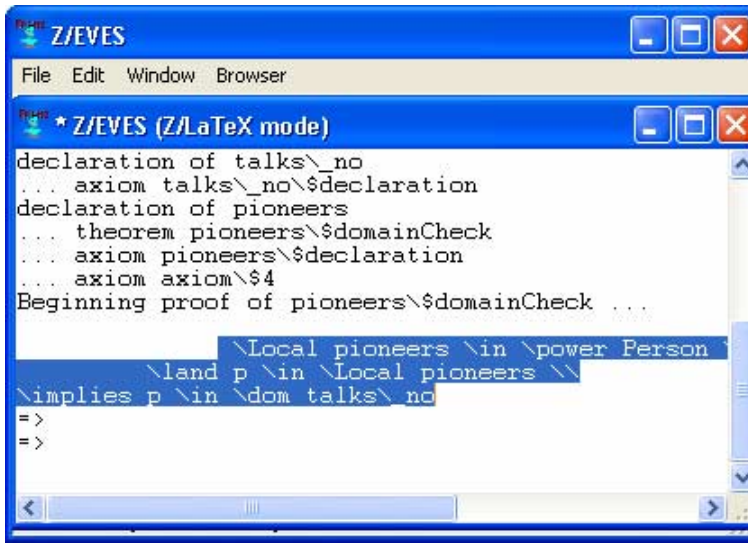

Fig. 1. Domain checking example

model checker, animator and theorem prover for Z, have been successful developed. With assistance from these tools, some inconsistences in the Z model can be detected and removed, so that the quality of the transformed ontology will be improved.

In this paper Z/EVES [8] was used to demonstrate how different kinds of inconsistences can be removed. Z/EVES is an interactive system for composing, checking, and analyzing Z specifications. It supports the analysis of Z specifications in several ways: syntax and type checking, schema expansion, precondition calculation, domain checking, and general theorem proving.

Firstly, we discuss how domain checking can be used to improve the ontology quality. Suppose for each person we also keep the number of talks he gave during last the year. This function can be modelled by a Z relation. Such as

$$talks\_no : Person \pfun \mathbb{N}_1$$

$$pioneers : \mathbb{P}\, Person \mid \forall p : pioneers \bullet talks\_no(p) > 1$$

We also want to identify a group of frequent speakers who are the pioneers in their areas. We assume that each pioneer gave at least two talks each year.

When the definition of *pioneers* is checked in Z/EVES, the status of the tool shows that it is syntactically and type correct, however it has an unproved goal (see Figure 1):

This means that Z/EVES might be unable to prove the predicates; in this case, this predicate is not true. In fact, the domain checking conditions show that we have forgotten some constraints in the axiom. The condition concerns the well-definedness of the final condition in the axiom, i.e.:

$$\forall p : pioneers \bullet talks\_no(p) > 1$$

In this context, *p* is known only to be a pioneer. However, not every person gives talks. So, this quantification should, in fact, range over only those persons who give at least one talk. (A closer inspection of the definition shows that *talks_on* is a partial function, with an unspecified domain.) In the light of this analysis, we can revise the definition to eliminate these flaws:

$$pioneers : \mathbb{P}\, Person$$
$$\hline$$
$$pioneers \subseteq \mathrm{dom}\, talks\_no \land \forall p : pioneers \bullet speaker(p) > 1$$

The ontology extracted from the former error model will be:

```
 <daml:class rdf:ID="pioneers">
  <rdfs:subClassOf rdf:resource="#person"/> <rdfs:subClassOf>
</daml:class>
```

After the correction, the DAML ontology we get is:

```
<daml:class rdf:ID="pioneers">
  <rdfs:subClassOf rdf:resource="#person"/>
  <rdfs:subClassOf>
    <daml:Restriction daml:minCardinality="1">
       <daml:onProperty rdf:resource="#talks_no"/>
    </daml:Restriction>
  </rdfs:subClassOf> </daml:class>
```

Note that this kind of error cannot be detected by the current DAML reasoner (i.e. OILEd, http://oiled.man.ac.uk/), therefore the DAML ontology which has been transformed from a (Z/EVES checked) Z model will be unlikely to have this kind of error.

Another kind of inconsistency which the DAML reasoner cannot check but Z/EVES can is illustrated in the following case.

Suppose we divided the talks into two categories *FreeTk* and *TicketTk*. The public can attend the *FreeTk*s freely, while they have to buy a ticket to attend the *TicketTk*. *FreeTk* and *TicketTk* are disjoint. *ftk* is a *FreeTK* and *ttk* is a *TicketTk*.

$$FreeTk, TicketTk :: \mathbb{P}\, Talk$$
$$\hline$$
$$FreeTk \cap TicketTk = \varnothing$$

$$ftk : FreeTk$$
$$ttk : TicketTk$$

A thrifty person will only attend *FreeTk*. Tom and Jerry are two thrifty persons. Tom had attended *ftk* and Jerry had attended *ttk*.

$$Thrifty$$
$$attended\_tk : \mathbb{P}\, FreeTk$$

$$Tom, Jerry : Thrifty$$
$$\hline$$
$$Jerry.attended\_tk = \{ftk\}$$
$$Tom.attended\_tk = \{ttk\}$$

There is an inconsistency in this model. Tom is declared as a thrifty person. At same time Tom also attended a talk – *ttk* which requires buy a ticket. *FreeTk* and *TicketTk* sets are disjoint. This inconsistency cannot be detected by the DAML reasoner. Using Z/EVES the inconsistency can be detected easily at the Z level before transformation to DAML. Suppose we ask Z/EVES to prove the correctness of Jerry and Tom. Jerry is fine, however we get *false* for Tom (see Figure 2).

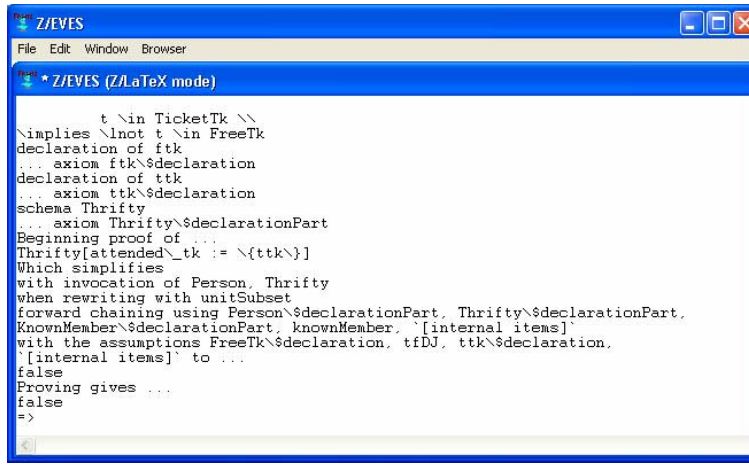

Fig. 2. Thrifty example

After we study the proving steps the error can be removed.

The inverse transformation (transforming a DAML ontology into a Z model) can be readily achieved through another XSL sheet file. So then the Z tools can be used to improve the quality of the existing ontology. For example, all the flaws in ontology presented in this section could exist in any existing ontology. After transformation of the DAML ontology to Z model and with the assistance from diverse Z tools, these flaws can be detected and removed. The Z can then be transformed back to DAML with an improved quality.

6 Conclusion

In this paper, we demonstrate that Z can capture various requirements of SW services including ontology and service functionalities. The main contribution of this paper is that it develops systematic transformation rules and tools which can project Z models to DAML ontology automatically. Another contribution of this paper is that we demonstrate some ontology related flaws in Z model can be detected and removed with the assistance of Z/EVES so that the transformed DAML ontology from checked Z model will have better quality. One obvious further work is to fully develop the reverse transformation tools from DAML ontology to the Z model then to use Z/EVES tools to detect domain and logical errors that the current DAML reasoner is not able to detect. Transformation from Z operation schemas to DAML-S [1] actions will be another interesting future work.

From a complete different direction, we also recently investigated how RDF and DAML can be used to build a Semantic Web environment for supporting, extending and integrating various formal specification languages [6]. One additional benefit is that RDF query techniques can facilitate formal specification comprehension.

In summary, there is a clear synergy between SW languages and formal specifications. The investigation between those two paradigms will lead great benefits for both areas.

Acknowledgements

We would like to thank Hugh Anderson and anonymous referees for many helpful comments. This work is supported by the Academic Research grant *Integrated Formal Methods* from National University of Singapore and Defence Innovative Research grant *Formal Design Methods and DAML* from Defence Science & Technology Agency (DSTA) Singapore.

References

1. Daml service. http://www.daml.org/services/daml-s/2001/05/.
2. Ittaks homepage. http://www.ittalks.org/jsp/Controller.jsp.
3. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. Scientific American, May 2001.
4. D. Brickley and R.V. Guha (editors). Resource description framework (rdf) schema specification 1.0. http://www.w3.org/TR/2000/CR-rdf-schema-20000327/, March, 2000.
5. J. Broekstra, M. Klein, S. Decker, D. Fensel, and I. Horrocks. Adding formal semantics to the web: building on top of rdf schema. In *ECDL Workshop on the Semantic Web: Models, Architectures and Management*, 2000.
6. J. S. Dong, J. Sun, and H. Wang. Semantic web for extending and linking formalisms. In L.-H. Eriksson and P. A. Lindsay, editors, *Proceedings of Formal Methods Europe: FME'02*, Copenhagen, Denmark, July 2002. Springer-Verlag.
7. O. Lassila and R. R. Swick (editors). Resource description framework (rdf) model and syntax specification. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, Feb, 1999.
8. Mark Saaltink. The Z/EVES system. In J. P. Bowen, M. G. Hinchey, and D. Till, editors, *ZUM'97: Z Formal Specification Notation*, volume 1212 of *Lecture Notes in Computer Science*, pages 72–85. Springer-Verlag, 1997.
9. J. Sun, J. S. Dong, J. Liu, and H. Wang. Object-Z Web Environment and Projections to UML. In *WWW-10: 10th International World Wide Web Conference*, pages 725–734. ACM Press, May 2001.
10. F. van Harmelen, P. F. Patel-Schneider, and I. Horrocks (editors). Reference description of the daml+oil ontology markup language. Contributors: T. Berners-Lee, D. Brickley, D. Connolly, M. Dean, S. Decker, P. Hayes, J. Heflin, J. Hendler, O. Lassila, D. McGuinness, L. A. Stein, ..., March, 2001.
11. World Wide Web Consortium (W3C). Extensible stylesheet language (xsl). http://www.w3.org/Style/XSL.
12. J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall International, 1996.