Verifying DAML+OIL and Beyond in Z/EVES

Jin Song Dong[1] Chew Hung Lee[2] Yuan Fang Li[1*] Hai Wang[3]
[1] School of Computing
National University of Singapore
{dongjs,liyf}@comp.nus.edu.sg

[2] DSO National Laboratories
Defense Science Organization (DSO)
lchewhun@dso.org.sg

[3] Department of Computer Science
University of Manchester
hwang@cs.man.ac.uk

Abstract

Semantic Web emerged as the next generation of Web since the past few years. It gives data well-defined and machine-understandable meaning so that they can be processed by remote intelligent agents cooperatively. Ontology languages are the building blocks of Semantic Web as they prescribe how data are defined and related. The existing reasoning and verification tools for Semantic Web are improving however still elementary. We believe that Semantic Web can be a novel application domain for software modeling languages and tools. Z is a formal modeling language for specifying software systems and Z/EVES is a proof tool for Z. In this paper, we firstly present Z semantics for ontology language DAML+OIL. This semantic model is embedded as a Z section `daml2z` *in Z/EVES, which serves as an environment for checking and verifying Web ontologies. Then we present a tool for automatically transforming ontology documents into the specialized Z codes understood by Z/EVES. Finally, we use a very recent real application, the military plan ontologies, to demonstrate the different reasoning tasks that Z/EVES can perform. Furthermore, undiscovered errors in the original ontologies were found by Z/EVES and some of these errors are even beyond Semantic Web modeling and reasoning capabilities.*

1. Introduction

Tim Berners-Lee et al. envisioned the Semantic Web, the next generation of Web, in which not only human-human communication is possible; intelligent agents: software that can interpret and process data shared on the Web, will be able to complete complex tasks on human's behalf cooperatively [2]. Ontology languages such as DAML+OIL [20] and OWL [6] play a key role in realizing the full potential of Semantic Web as they prescribe how data are defined and related. Ontology languages are based on description logic and they are designed to be decidable [24] (except OWL Full). Ensuring the consistency of shared ontologies is crucial to the proper functioning of agents. Since Semantic Web is still evolving and in its early stage, current verification tools are improving though rudimentary. It is our belief that Semantic Web can be a new application domain for software modeling languages and tools.

Z [22] is a formal specification language well suited to model system data and state. It is based on ZF set theory and first-order predicate logic. Description logic can be regarded as a subset of predicate logic [14], therefore it is not surprising that Z is more expressive than ontology languages. Z/EVES [19] is an interactive proof tool for checking and reasoning Z specifications. The intrinsic homogeneity between semantic bases of ontology languages and Z implies that Z can be regarded as an ontology meta-language and it can even capture properties that ontology languages cannot. We believe that by defining Z semantics for ontology languages and transforming ontologies into Z specifications, Z/EVES can be used to improve the quality of ontologies by verifying properties of the Z specifications, sometimes beyond Semantic Web.

In this paper, we firstly give an overview of Semantic Web, ontology languages, the DSO military plan ontologies, Z and Z/EVES (section 2). Z semantics for ontology language DAML+OIL will be presented (section 3). This semantic model is embedded as a Z section `daml2z` in Z/EVES, which serves as an environment for checking and verifying web ontologies. Then we illustrate a tool for automatically transforming ontologies into the specialized Z codes (section 4). Finally, we use a very recent real appli-

* Author of correspondence: liyf@comp.nus.edu.sg

cation, the military plan ontologies, to demonstrate the different reasoning tasks that Z/EVES can perform (section 5). Section 6 discusses related works and concludes the paper.

2. Overview

2.1. Semantic Web & ontology languages

The Semantic Web is a vision of next generation of the Web. It is believed that in the future, the Web is no longer what we conventionally think as only meant for humans to read, it is also meant for intelligent software agents and is truly ubiquitous. Software agents will reside in, for example, household appliances (which can also be part of the Web), and be able to understand the meaning of data on the Web and undertake tasks without human's supervision. Understanding of data is built on giving data well-defined structure and meaning, for which ontologies are designed.

Ontologies are expressed in terms of ontology languages, which are built on top of XML. Compared to HTML, XML has two main advantages. Firstly, it is extensible, which means XML is actually a metalanguage: a language to describe other languages. Users can design their own markup language with different tag names and data types. The possibility of having limitless languages built on top of XML greatly enhances the Web's ability to scale. Secondly, it is strict in terms of structure; XML is well-defined, in other words. By rigorously defining the structure of data, XML provides a syntactic basis for data to be represented unambiguously. However, as XML aims at defining the structure of documents, it is almost impossible for an agent to understand the meaning of a document it encounters on the Web. We need a way for machines to understand data.

RDF (Resource Description Framework) [17] is a model of metadata defining a mechanism for describing resources that makes no assumptions about a particular application domain. It allows structured and semi-structured data to be mixed and shared across applications. XML describes documents, whereas RDF is a framework for metadata: it describes actual things. It provides a simple way to make *statements* about Web resources. Statements are of the form $\langle$*subject predicate object*$\rangle$, where *subject* is the resource we are interested in, *predicate* specifies the property or characteristic of the subject and *object* states the value of the property. RDF Schema [3] provides basic vocabularies for describing RDF documents. In order for agents to understand data unambiguously, it is necessary that these data items are strictly structured. This requirement is relaxed by RDF to allow for greater flexibility. Moreover, RDF Schema does not contain all modeling primitives users have desired.

DAML (DARPA Agent Markup Language) [20] is built on top of RDF Schema and it has a much richer set of language constructs to express class and property relationships than those allowed in RDF Schema; more refined support for data types are also incorporated in DAML. DAML combined effort with the Ontology Inference Layer (OIL) [4] project and it is now referred to as DAML+OIL. The other major extension of DAML+OIL is the ability to express restrictions on class and property definitions. By restricting existing classes and properties, new concepts can be built incrementally. This facilitates construction of new ontologies as previous ones can be reused.

In 2003, W3C published a new ontology language, the OWL (Web Ontology Language) [23]. Based on DAML+OIL, OWL consists of three sublanguages: Lite, DL and Full, with increasing expressiveness. The three sublanguages are meant for user groups with different requirements of expressiveness and decidability. OWL Lite and DL are decidable whereas OWL Full is generally not.

The consistency of ontologies is essential to the proper functioning of agents. For example, we can imagine how chaotic it can be if an online marriage registry agent allows a person already married to register for marriage again. This could happen if the marriage ontology does not constrain that a person can only have at most one spouse. A consistent ontology satisfies the following two criteria: *realization*, every class has at least one instance and *retrieval*, every individual is an instance of some class [16].

Alongside ontology languages, a number of ontology-related tools have been developed. [18] provides an extensive survey. Here we briefly introduce two reasoners that support DAML+OIL: FaCT and RACER. FaCT (**Fa**st **C**lassification of **T**erminologies) [9] is a T-Box (concept level) reasoner, whose major functionalities include concept subsumption and satisfiability testing. RACER (**R**enamed **A**Box and **C**oncept **E**xpression **R**easoner) [8] implements a TBox and ABox (instance level) reasoner for the description logic $\mathcal{SHIQ}$ [11]. Compared to FaCT, it has much richer functionalities, including creating, maintaining, deleting ontologies, concepts, roles and individuals, querying, retrieving and evaluating the knowledge base, etc. Moreover, they perform their functions automatically, which means by "pushing a button", these tools return a definitive answer without intermediate steps.

2.2. Military Plan Ontology

DSO National Laboratories (DSO) developed a DAML+OIL military plan ontology [15], defining concepts in the military domain, including military organizations, specialities, geographic features, etc. For example, the class `LandMineField`, a sub class of `LandArea`, is defined as follows.

```
<daml:Class rdf:about="http://www.dso.org.sg/
    PlanOntology#LandMineField">
  <rdfs:label>LandMineField</rdfs:label>
  <rdfs:subClassOf>
    <daml:Class rdf:about="http://teknowledge.com/
```

```
      ontology/Merge.txt#LandArea"/>
  </rdfs:subClassOf>
</daml:Class>
```

A number of plan instances of this ontology were also generated from plain text by an information extraction (IE) engine developed by DSO. Military plans are typically prepared as both graphical overlays and textual documents detailing the plans. IE is used as the first part of a process to transform the textual documents into ontological data. A typical IE workflow consists of word segmentation & stemming, POS (part of speech) tagging, Named Entity recognition, syntactic processing, etc. With all information gathered from the various steps, the IE engine then fills the slots in pre-defined templates, which are subsequently transformed into a RDF document. Generally speaking, an instance ontology is made up of the following four main ingredients.

- A set of military operations and tasks, defining their types, phases and the logic order.
- A set of military units, which are the participants of the military operations and tasks,
- A set of geographic locations, where such operations take place and
- A set of time points for constraining the timing of such operations.

2.3. Z & Z/EVES

Z [22] is a formalism based on ZF set theory and first-order predicate logic. It is specially suited to model system data and state. Z has a number of language constructs including given type, abbreviation type, axiomatic definition, state and operation schema definitions, etc.

Since ontology languages are based on description logic, which can be regarded as a subset of predicate logic [14], Z is by nature more expressive than ontology languages. Hence, it is able to capture more facets of information than ontology languages can. In the following section, we will present the Z semantics for DAML+OIL.

Z/EVES [19] is an interactive system for composing, checking, and analyzing Z specifications. It supports the analysis of Z specifications in a number of ways: syntax and type checking, schema expansion, precondition calculation, domain checking, general theorem proving, etc.

In Z/EVES, Z specifications are in the form of *sections* to improve reuse. The built-in section `toolkit` defines basic constants and operators. Specifications are built hierarchically by including existing sections as their parents.

3. Z Semantics for DAML+OIL

To use Z/EVES to check ontologies, it is necessary to define Z semantics for the ontology language. This semantic model serves as a reasoning environment for verification using Z/EVES. In this section, we define Z semantics for a subset of DAML+OIL language primitives. The complete model can be found in Appendix A. Note that we present the Z definitions in Z/EVES syntax, where extra parentheses around predicates are sometimes needed for Z/EVES to correctly parse them. In the second part, proof support for Z/EVES is discussed. Examples of various constructs defined in this section can be found later in this paper.

It may be noted that we build Z semantics for DAML+OIL using only axiomatic definitions. This is because in this paper, we are only interested in checking the static properties of ontologies, which can be well captured by axiomatic definitions. Schema definitions, on the other hand, are best used to model dynamic properties, as found in Semantic Web services such as DAML-S [5].

3.1. Z Semantics for DAML+OIL

Basic Concepts
Everything in Semantic Web is a `Resource`. So we model it as a given type in Z.

$$[Resource]$$

`Class` corresponds to a concept, which has a number of resources associated with it: the `instances` of this class. Hence, we model class as a subset of resource and instances as a function from a class to a set of resources.

$$Class : \mathbb{P}\, Resource$$
$$instances : Class \rightarrow \mathbb{P}\, Resource$$

`Property` is also a subset of Resource, disjoint with class. A property relates resources to resources. The function `sub_val` maps each property to the resources it relates.

$$Property : \mathbb{P}\, Resource$$
$$\hline$$
$$Property \cap Class = \varnothing$$

$$sub\_val : Property \rightarrow (Resource \leftrightarrow Resource)$$

Class relationships
The property `subClassOf` is defined as a relation from class to class. For a class c_1 to be the sub class of class c_2, the instances of c_1 must be a subset of instances of c_2. Other properties such as `disjointWith` are similarly defined. Note that the subset relationship is expressed in terms of membership relationship to make proof in Z/EVES more automated.

$$
\begin{array}{|l}
subClassOf : Class \leftrightarrow Class \\
disjointWith : Class \leftrightarrow Class \\
\hline
\forall c_1, c_2 : Class \bullet \\
c_1 \ \underline{subClassOf}\ c_2 \Leftrightarrow instances(c_1) \in \mathbb{P}\, instances(c_2) \\
c_1 \ \underline{disjointWith}\ c_2 \Leftrightarrow instances(c_1) \cap instances(c_2) = \varnothing
\end{array}
$$

Class & Property
The property `toClass` attempts to establish a maximal possible set of resources as a class. It states that any resource a_1 is an instance of class c_1 if either: a_1 is defined for property p and $(a_1, a_2) \in sub\_val(p)$ implies that a_2 is an instance of class c_2; or that a_1 is not defined for p at all.

$$
\begin{array}{|l}
toClass : (Class \times Property) \leftrightarrow Class \\
\hline
\forall c_1, c_2 : Class;\ p : Property \bullet (c_1, p)\ \underline{toClass}\ c_2 \Leftrightarrow \\
(\forall a_1, a_2 : Resource \bullet a_1 \in instances(c_1) \Leftrightarrow \\
\quad ((a_1, a_2) \in sub\_val(p) \Rightarrow a_2 \in instances(c_2)))
\end{array}
$$

Property `hasValue` states that all instances of class c have resource r for property p.

$$
\begin{array}{|l}
hasValue : (Class \times Property) \leftrightarrow Resource \\
\hline
\forall c : Class;\ p : Property;\ r : Resource \bullet \\
(c, p)\ \underline{hasValue}\ r \Leftrightarrow \\
\quad (\forall a : instances(c) \bullet (a, r) \in sub\_val(p))
\end{array}
$$

Property relationships
The property `subPropertyOf` states that a property p_1 is a sub property of another property p_2 iff $sub\_val(p_1)$ is a subset of $sub\_val(p_2)$.

$$
\begin{array}{|l}
subPropertyOf : Property \leftrightarrow Property \\
\hline
\forall p_1, p_2 : Property \bullet p_1\ \underline{subPropertyOf}\ p_2 \Leftrightarrow \\
sub\_val(p_1) \in \mathbb{P}\, sub\_val(p_2)
\end{array}
$$

3.2. Proof support for Z/EVES

The semantic model is contained in a section `daml2z`, on top of `toolkit`. According to the authors of Z/EVES, definitions alone are not sufficient to exploit the full power of Z/EVES. An ample stock of rewrite rules, forward rules and assumption rules is needed to make proof processes more automated. Based on the semantic model, we constructed a section, called `DAML2ZRules`, of rules which describes the above definitions in more than one angle. This section has `daml2z` as parent.

The rewrite rule `toClassDisjointWithRule1`, for example, relates two properties: toClass and disjointWith. This rule states that for classes c_1, c_2, c_3 and property p, if c_2 and c_3 are disjoint and $((c_1, p), c_3)$ satisfies property toClass, then it can be implied that $((c_1, p), c_2)$ does not satisfy toClass.

theorem rule toClassDisjointWithRule1

$$
\begin{array}{l}
\forall c_1, c_2, c_3 : Class;\ p : Property \bullet \\
(c_3, c_2) \in disjointWith \wedge ((c_1, p), c_3) \in toClass) \Rightarrow \\
\neg\, ((c_1, p), c_2) \in toClass
\end{array}
$$

Ontologies are built layer on layer. Other domain specific ontologies are built in terms of basic concepts presented in this section and their corresponding Z models will have `DAML2ZRules` or its descendent sections as parents.

4. Transforming DAML+OIL to Z

Using a Java package for Semantic Web, the "Jena Frameworks" [12], we have developed a tool in Java to automatically transform ontologies into Z. Given a DAML+OIL or RDF ontology, it iterates through all elements and transforms them into Z definitions.

We used this tool to transform the military plan ontology introduced in section 2 into Z section `military`, with `DAML2ZRules` as parent.

To better utilize Z/EVES's proof power, We made the following enhancements to the `military` section:

- *labels* are systematically added to Z predicates during transformation to make them axioms (either rewrite rules or assumption rules) recognized by Z/EVES, which will assume an assumption rule to be true and rewrite the left-hand side of a rewrite rule to its right-hand side during the proof process.
- Since MilitaryProcess and its sub classes have a start and end time, `start` and `end` are modeled as functions from MilitaryProcess to integer, so that Z/EVES can perform reasoning over integer domain.
- A set of theorems specific to these military definitions are formulated. These theorems describe the relationships among the various military entities. For example, we have theorems stating sub task relationship between different kinds of military tasks, transitivity of sub task relationship, etc.

The class `LandMineField` presented earlier in section 2.2 is transformed into the following axiomatic definition. Note that the predicate $(LandMineField, LandArea) \in subClassOf$ is marked as an assumption rule so that during proof, Z/EVES will automatically that `LandMineField` is a sub class of `LandArea`.

$$
\begin{array}{|l}
LandMineField : Class \\
\hline
\langle\!\langle \text{grule LandMineField\_subClassOf\_LandArea} \rangle\!\rangle \\
(LandMineField, LandArea) \in subClassOf
\end{array}
$$

Our tool also supports transforming instance ontologies into Z specifications. For example, a fragment of an instance plan ontology, planE.daml, and its Z model are shown below (names in ontology and Z section are shortened to save space, when necessary).

```
<rdf:Description rdf:about='G. DALLAS'>
  <rdf:type rdf:resource='http://teknowledge.com/
     ontology/Merge.txt#GeographicArea'/>
```

```
</rdf:Description>
<rdf:Description rdf:about='TF 1'>
  <rdf:type rdf:resource='http://www.dso.org.sg/
     PlanOntology#Task_Force'/>
</rdf:Description>
<rdf:Description rdf:about='PLAN-P3-P6-P1'>
  <NS4:subTaskOf rdf:resource='PLAN-P3-P6'/>
  <rdf:type rdf:resource=
     'http://www.dso.org.sg/PlanOntology
     #EstablishPosition-MilitaryTask'/>
  <NS0:start rdf:resource='7'/>
  <NS0:end rdf:resource='12'/>
  <NS4:assignedTo
     rdf:resource='ArmouredBattalion_51b'/>
</rdf:Description>
```

$$\frac{G\_DALLAS : Resource}{\langle\!\langle \text{grule G\_DALLAS\_type} \rangle\!\rangle \\ G\_DALLAS \in instances(GeographicArea)}$$

$$\frac{TF\_1 : Resource}{\langle\!\langle \text{grule TF\_1\_type} \rangle\!\rangle \\ TF\_1 \in instances(Task\_Force)}$$

$$\frac{PLAN\_P3\_P6\_P1 : Resource}{\langle\!\langle \text{grule PLAN\_P3\_P6\_P1\_type} \rangle\!\rangle \\ PLAN\_P3\_P6\_P1 \in \\ \quad instances(EstablishPosition\_MilitaryTask) \\ \langle\!\langle \text{rule PLAN\_P3\_P6\_P1\_assignedTo} \rangle\!\rangle \\ (sub\_val(assignedTo))(\!| \{PLAN\_P3\_P6\_P1\} |\!) = \\ \quad \{ArmouredBattalion\_51b\} \\ \langle\!\langle \text{rule PLAN\_P3\_P6\_P1\_end} \rangle\!\rangle \\ end(PLAN\_P3\_P6\_P1) = 12 \\ \langle\!\langle \text{rule PLAN\_P3\_P6\_P1\_start} \rangle\!\rangle \\ start(PLAN\_P3\_P6\_P1) = 7$$

Sometimes, manual works are necessary to make the Z definitions acceptable by Z/EVES and to make the proof process more automated. To minimize the manual works and fully utilize the automated proof power of Z/EVES, we fine-tuned the transformation tool as follows:

- For the same reasons as in plan ontology, labels are added to all Z predicates.
- Z definitions and predicates are re-ordered during transformation to avoid advance or circular references, which are not allowed by Z/EVES.
- In military plans, information about to which military units a military task is assigned to is captured but the inverse relation is not. In order for more automated reasoning about the temporal relationships of these tasks, information about military units executing tasks is collected separately by the tool and the corresponding Z definitions are put to the end of the specification.
- In ontology languages, different names refer to different entities (Unique Name Assumption [8]). However, in Z, different names can refer to the same entity. We use cardinality of sets to make Z/EVES work the same way. Whenever two military tasks are related by sub task or super task relationship, we construct a set containing the two tasks and assume the cardinality of the set is two, as follows:

$$\langle\!\langle \text{grule PLAN\_P3\_P6\_P1\_disj\_PLAN\_P3\_P6} \rangle\!\rangle \\ \#\{PLAN\_P3\_P6\_P1, PLAN\_P3\_P6\} = 2$$

The benefits of these enhancements in transformation will be seen when we discuss reasoning beyond DAML+OIL in section 5.

5. Checking Ontologies Using Z/EVES

In this section, we demonstrate how Z/EVES can be used to verify properties of ontologies through the military ontologies case study. The demonstration consists of two parts. In the first part, standard Semantic Web reasoning: (class) inconsistency, subsumption and instantiation testing, whose formal definitions can be found in [1], are performed. In the second part, we will show that Z/EVES can check ontology properties beyond Semantic Web. In this section, labels are shown only when they are used by Z/EVES during the proof process, either automatically or interactively.

By applying Z/EVES, we discovered an error in the plan ontology in the first part. In the second part, more errors beyond Semantic Web modeling capabilities were found by Z/EVES.

5.1. Standard Semantic Web reasoning

Inconsistency Checking
Ensuring the consistency of individual classes is an important task as the property of overall ontology consistency can be reduced to class consistency [10].

After transforming the plan ontology into Z section `military`, We applied Z/EVES to section `military` to systematically check consistency for its classes. During checking, we identified the following closely-related Z definitions.

$$\frac{PrepareDemolition\_MilitaryTask : Class}{(PrepareDemolition\_MilitaryTask, MilitaryTask) \in \\ \quad subClassOf}$$

$$\frac{EngineerUnit : Class}{(EngineerUnit, ModernMilitaryUnit) \in subClassOf \\ \langle\!\langle \text{grule EngineerUnitSpeciality} \rangle\!\rangle \\ ((EngineerUnit, speciality), \\ \quad EngineeringMilitarySpeciality) \in hasValue \\ \langle\!\langle \text{grule DemolitionAssignedtoEngin} \rangle\!\rangle \\ ((PrepareDemolition\_MilitaryTask, assignedTo), \\ \quad EngineerUnit) \in toClass}$$

$$
\begin{array}{|l}
\hline
\mathit{EngineerSection} : \mathit{Class} \\
\hline
\langle\!\langle \text{grule SectionIsSubClassOfUnit} \rangle\!\rangle \\
(\mathit{EngineerSection}, \mathit{EngineerUnit}) \in \mathit{subClassOf} \\
((\mathit{EngineerSection}, \mathit{echelon}), \mathit{SECT}) \in \mathit{hasValue} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\mathit{ArtilleryFiringUnit} : \mathit{Class} \\
\hline
\langle\!\langle \text{FUIsMUnit} \rangle\!\rangle \\
(\mathit{ArtilleryFiringUnit}, \mathit{ModernMilitaryUnit}) \in \\
\quad \mathit{subClassOf} \\
\langle\!\langle \text{grule FiringUnitDisjWithEngin} \rangle\!\rangle \\
(\mathit{ArtilleryFiringUnit}, \mathit{EngineerUnit}) \in \mathit{disjointWith} \\
\langle\!\langle \text{grule DemolitionAssignedToFU} \rangle\!\rangle \\
((\mathit{PrepareDemolition\_MilitaryTask}, \mathit{assignedTo}), \\
\quad \mathit{ArtilleryFiringUnit}) \in \mathit{toClass} \\
\hline
\end{array}
$$

With the assumption rule label DemolitionAssignedToFU removed, we issue the following goal to test the consistency of the above definitions.

```
try ((PrepareDemolition\_MilitaryTask,
assignedTo),ArtilleryFiringUnit) \in toClass;
```

We enter a sequence of commands into Z/EVES. The first 2 are axioms (labelled predicates) from the specification and the 3^{rd} is a theorem defined in section `DAML2ZRules`. The final command **`reduce`** performs simplification and rewriting.

Proof
use FiringUnitDisjWithEngin;
use DemolitionAssignedtoEngin;
*apply disjointWithRule*0;
reduce;

Z/EVES returns the following predicate as the remaining goal to be proven.

$$\neg\ \mathit{instances\ EngineerUnit} \cap \mathit{instances\ ArtilleryFiringUnit} = \{\}$$

We suspect that there is potentially an inconsistency since the disjointness of the above two classes is stated in the specification. Since it is very hard for a theorem prover to prove falsity, we use the usual trick: negate the goal and retry.

```
try \lnot ((PrepareDemolition\_MilitaryTask,
assignedTo),ArtilleryFiringUnit) \in toClass;
```

With the same sequence of commands entered, Z/EVES manages to return `true`. Hence we know that the predicate is inconsistent with the section. After checking the original ontology, we found that there is indeed an inconsistency, which was intentionally inserted by DSO staff as a test case for our tool without our pre-knowledge.

Subsumption Reasoning
The task of subsumption reasoning is to infer a DAML+OIL class is a sub class of another class. It is supported by Z/EVES with a high degree of automation: usually a `reduce;` command will prove the goal.

Instantiation Reasoning
Instantiation reasoning asserts that one resource is an instance of a class. Some Semantic Web reasoning tools, such as FaCT, are designed to only support TBox reasoning, hence reasoning involving instances cannot be performed. We demonstrate through an example that Z/EVES supports instance level reasoning.

In one of the instance ontologies, planE.daml, an instance of ModernMilitaryUnit is assigned to an instance of PrepareDemolition_MilitaryTask. We want to deduce that it is an instance of the class EngineerUnit.

$$
\begin{array}{|l}
\hline
\mathit{ModernMilitaryUnit\_8ad} : \mathit{Resource} \\
\hline
\langle\!\langle \text{grule ModernMilitaryUnit\_8ad\_type} \rangle\!\rangle \\
\mathit{ModernMilitaryUnit\_8ad} \in \\
\quad \mathit{instances}(\mathit{ModernMilitaryUnit}) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\mathit{PLAN\_P2\_P4} : \mathit{Resource} \\
\hline
\langle\!\langle \text{grule PLAN\_P2\_P4\_type} \rangle\!\rangle \\
\mathit{PLAN\_P2\_P4} \in \\
\quad \mathit{instances}(\mathit{PrepareDemolition\_MilitaryTask}) \\
\langle\!\langle \text{rule PLAN\_P2\_P4\_assignedTo} \rangle\!\rangle \\
(\mathit{sub\_val}(\mathit{assignedTo}))(\!|\ \{\mathit{PLAN\_P2\_P4}\}\ |\!) = \\
\quad \{\mathit{ModernMilitaryUnit\_8ad}\} \\
\hline
\end{array}
$$

```
try ModernMilitaryUnit\_8ad \in
    instances(EngineerUnit);
```

With two axioms from the specification and two theorems from section `DAML2ZRules` used, a final `prove;` command cleans up the proof and Z/EVES returns `true`.

Proof
use imageTupleRule[$p := \mathit{assignedTo}$,
$x := \mathit{PLAN\_P2\_P4}, y := \mathit{ModernMilitaryUnit\_8ad}$];
use DemolitionAssignedtoEngin;
use PLAN_P2_P4_type;
*use toClassInstanceRule*2
[$c_1 := \mathit{PrepareDemolition\_MilitaryTask}$,
$c_2 := \mathit{EngineerUnit}, a_1 := \mathit{PLAN\_P2\_P4}$,
$a_2 := \mathit{ModernMilitaryUnit\_8ad}, p := \mathit{assignedTo}$];
prove;
■

Instance Property Reasoning
Another important reasoning task in the Semantic Web domain is instance property reasoning, which is often regarded as knowledge base querying. In Semantic Web, a promising vision is that intelligent agents can discover information that is not explicitly stored in the knowledge base. We

illustrate Z/EVES's capability of instance property reasoning using an example.

In the beginning of this section, we know that the speciality of EngineerUnit is EngineeringMilitarySpeciality and that EngineerSection is a sub class of EngineerUnit. We want to know whether EngineerSection's speciality is also EngineeringMilitarySpeciality. The goal is established as follows:

```
try ((EngineerSection, speciality),
EngineeringMilitarySpeciality) \in hasValue;
```

With the following commands issued, Z/EVES proves the goal to be `true`.

Proof
use EngineerUnitSpeciality;
use SectionIsSubClassOfUnit;
*use subClassHasValueRule*1
$[c_1 := EngineerSection, c_2 := EngineerUnit,$
$p := speciality, r := EngineeringMilitarySpeciality]$;
reduce;
■

5.2. Checking beyond DAML+OIL

The above examples demonstrate Z/EVES's capability of performing consistency, subsumption and instantiation reasoning on Semantic Web ontologies with a certain degree of automation. Moreover, Z/EVES can check more complex properties that Semantic Web languages cannot capture. For example, Semantic Web languages have problems dealing with concrete domains. DAML+OIL, for instance, can only specify min, max and exact values of cardinality constraints over integer. Z/EVES, however, can perform basic arithmetic operations and comparisons, which improves proof power beyond Semantic Web.

This added power of Z/EVES is illustrated through the following real-world example. One of our aims is to check ontologies in the military domain. To ensure the correctness of a military ontology, it is not enough to check properties discussed in the last sub section. It is necessary to ensure, for example, that no military unit is assigned to two or more military tasks at the same time, and that no military task is a sub task of itself. Semantic Web ontology languages cannot capture this idea. Using Z/EVES, we can check whether these constraints are satisfied.

In this subsection, we consider one of the instance ontologies: planE.daml. Using the tool described previously, it is transformed into a Z section, which was partially shown in Section 4. A brief statistics of this ontology and the corresponding Z section can be found in Table 1. The inconsistency we discovered in the military ontology earlier was also summarized in the table (ontology error).

Items	**Numbers**
Resources	138
Operations, tasks, phases	56
Units	47
Geographic areas	35
Statements (in planE.daml)	592
Ontology error (in military ontology)	1
Transformed Axiomatic Defns (in Z)	138
Transformed Predicates (in Z)	410
Type errors	22
Hidden errors	***9***

Table 1. Statistics of the military ontology & instance ontology planE.daml

Note that 22 type errors were detected by Z/EVES. Most of these errors are caused by the inaccuracy of the IE engine: for example, `Task_Force` was defined as a class in section `military`; it is redefined as a resource of type Thing in this instance ontology.

After correcting all syntax and type errors in the Z model, we use a set of theorems to systematically test properties beyond Semantic Web. 9 *hidden errors* are discovered. 2 of them are caused by military tasks having start time greater than end time; 4 are caused by military tasks without end time defined and 3 are caused by a military unit being assigned to different tasks simultaneously. The rest of this section is devoted to showing how various checking beyond Semantic Web can be performed by Z/EVES.

The following theorem tests that for a given military task, its start time is less than or equal to its end time and it is not a sub task of itself.

theorem MilitaryTaskTimeSubTaskTest1
$\exists x : instances(MilitaryTask) \bullet$
$start(x) < end(x) \land$
$x \notin (sub\_val(subTaskOf))(\!| \{x\} |\!)$

For example, one military task is tested using the following proof scripts.

Proof
*try lemma MilitaryTaskTimeSubTaskTest*1;
instantiate x == *PLAN_P2_P7_S1_P1*;
cases;
use cardCup [*Resource*] [$S := \{PLAN\_P2\_P7\_S1\_P1\}$,
$T := \{PLAN\_P2\_P7\_S1\}$];
reduce;
next;
■

The proof process is intuitive: we consider the super tasks of x as sub goals one by one. When all sub goals are com-

pleted, this goal is proven. The rule *cardCup* is defined in section `toolkit`, it is used here to make the two military tasks distinct, as we discussed in the end of section 4.

We test another military task: `PLAN_P3_P3_S1`. Similar commands as in the previous example are issued. With the command **`next`** issued, Z/EVES returns `false`, which suggests that something is wrong. Command **`reduce`** results in the following remaining goal:

$$\exists x : \mathit{instances}\ \mathit{MilitaryTask} \bullet \\ \mathit{start}\ x < \mathit{end}\ x \land \\ \neg\ x \in \mathit{sub\_val}\ \mathit{subTaskOf}\ (\!|\ \{x\}\ |\!)$$

which is equivalent to the original theorem. This means that the instantiation to `PLAN_P3_P3_S1` does not prove the goal. By negating the theorem and trying again, Z/EVES does return `true`. After checking the ontology, we found that start time is 7 but end time is 4, hence it is indeed an error, which cannot be found by Semantic Web-specific tools.

Applying this checking to all military tasks revealed 2 such errors. There are two possible sources: inaccuracy of the IE engine or human error. After checking with the developers at DSO, it was found out that the errors were in the original text document, which is the input of the IE engine. Hence it was human error.

The discovery of this kind of errors motivated us to perform some more advanced reasoning. The following theorem states that for a given military unit and two military tasks assigned to this unit, the durations of the two tasks do not overlap. Since we have ensured that start time is less than the end time for each military task, the predicate $\mathit{end}(y) \leq \mathit{start}(z) \lor \mathit{end}(z) \leq \mathit{start}(y)$ is sufficient.

theorem MilitaryUnitTest

$$\exists x : \mathit{instances}(\mathit{ModernMilitaryUnit}) \bullet \\ \exists y, z : \mathit{instances}(\mathit{MilitaryTask}) \mid \\ (x, y) \in \mathit{assignedTo} \land (x, z) \in \mathit{assignedTo} \bullet \\ \mathit{end}(y) \leq \mathit{start}(z) \lor \mathit{end}(z) \leq \mathit{start}(y)$$

We systematically apply this theorem to appropriate military units and tasks. As stated in section 4, we have collected information about what tasks each military unit executes, it is easy to proceed in this case. The proof process of one such combination is shown below.

Proof

try lemma MilitaryUnitTest;
instantiate x == *TF*_1;
instantiate y == *PLAN_P*3_*P*5_*S*1;
reduce;
cases;
instantiate z == *PLAN_P*3_*P*5_*S*3;
reduce; [*a potential error*!]

During the proof process, we note that although we repeatedly apply *instantiate z* == *PLAN_P*3_*P*5_*S*3; , the remaining goal is unchanged. Hence, we sense that there is a potential error and we negate the theorem and prove it.

After issuing the same sequence of commands, we proved the negated goal. We found in the original ontology that the start and end time of these two military tasks are the same. Hence this is indeed an error that cannot be discovered by Semantic Web-specific checking tools.

6. Conclusion

In this paper, we demonstrated the Z semantics for ontology language DAML+OIL and automatic transformation of DAML+OIL and RDF ontologies into Z specifications. Through a recent case study, the military plan ontologies, we showed that Z/EVES can be used to check properties of ontologies. Undiscovered errors in the original ontologies were found by Z/EVES. Some of these errors are beyond the modeling capabilities of ontology languages and hence cannot be found by Semantic Web reasoning tools.

In our previous works, the reverse approach [7] was investigated, in which DAML+OIL ontologies can be extracted from Z requirement models. We also applied the Alloy [13] model checker to DAML+OIL to perform automated reasoning [21]. There are some pros and cons to Alloy approach. Being a model checker, reasoning in Alloy is fully automated and if there is an inconsistency, Alloy can give a counter example so that it is easier to trace the origin of the inconsistency. On the other hand, Alloy is not very scalable. Since it performs exhaustive search, it can only handle ontologies with no more than twenty entities. Moreover, Alloy does not support concrete domains such as integer. These characteristics make Alloy more automated, but less powerful and expressive than Z/EVES.

Compared to Semantic Web-specific reasoning tools and Alloy, the apparent disadvantage of Z/EVES, being a theorem prover, is that it has a lower degree of automation and can only perform reasoning tasks interactively. However, the high degree of expressiveness of Z language implies that it can capture properties beyond ontology languages and applying Z/EVES to checking ontologies gives us more confidence in the correctness of ontology related properties.

The new ontology language OWL Full is designed to be very expressive and reasoning will generally be undecidable [23]. Therefore, proof will be inevitably interactive and Z/EVES is a natural choice for reasoning OWL. Extending the support to OWL will be one future work direction. Modeling and checking behaviors of Semantic Web services in Z or other formalisms such as process algebra would be another future work.

Acknowledgement
The authors would like to thank Chan Kum Lan, Chew Lock Pin, How Khee Yin and Lee Hian Beng for their collaboration and support. We are also grateful to anonymous refer-

ees for their valuable comments. This work is partially supported by the Defense Innovative Research Project (DIRP) research grant “Formal Design Methods and DAML”.

References

[1] F. Baader and W. Nutt. Basic description logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *The description logic handbook: theory, implementation, and applications*, pages 43–95. Cambridge University Press, 2003.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. Scientific American, May 2001.

[3] D. Brickley and R. G. (editors). Resource description framework (rdf) schema specification 1.0. http://www.w3.org/TR/2000/CR-rdf-schema-20000327/, March, 2000.

[4] 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.

[5] M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng. Daml service. http://www.daml.org/services/daml-s/2001/05/.

[6] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. S. (editors). OWL Web Ontology Language 1.0 Reference. http://www.w3.org/TR/owl-ref/, 2002.

[7] J. S. Dong, J. Sun, and H. Wang. Z Approach to Semantic Web. In *International Conference on Formal Engineering Methods (ICFEM’02)*, pages 156–167, Shanghai, China, Oct. 2002. LNCS, Springer-Verlag.

[8] V. Haarslev and R. Möller. *RACER User’s Guide and Reference Manual: Version 1.7.6*, Dec. 2002.

[9] I. Horrocks. The FaCT system. *Tableaux’98, LNCS*, 1397:307–312, 1998.

[10] I. Horrocks. DAML+OIL: a description logic for the semantic web. *IEEE Data Engineering Bulletin*, 25(1):4–9, 2002.

[11] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.

[12] HP Labs. Jena Semantic Web Toolkit - version 1. http://www.hpl.hp.com/semweb/jena1.htm.

[13] D. Jackson. Alloy: A lightweight object modeling notation. Available: http://sdg.lcs.mis.edu/alcoa, 1999.

[14] P. Lambrix. Description Logics home page. http://www.ida.liu.se/labs/iislab/people/patla/DL/index.html.

[15] C. H. Lee. Phase I Report for Plan Ontology. DSO National Labs, Singapore, 2002.

[16] D. Nardi and R. J. Brachman. An introduction to description logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *The description logic handbook: theory, implementation, and applications*, pages 1–40. Cambridge University Press, 2003.

[17] 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.

[18] Ontoweb Ontology-Based Information. Deliverable 1.3: A survey on ontology tools. http://ontoweb.aifb.uni-karlsruhe.de/About/Deliverables/D13_v1-0.zip.

[19] M. 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 *Lect. Notes in Comput. Sci.*, pages 72–85. Springer-Verlag, 1997.

[20] F. van Harmelen, P. F. Patel-Schneider, and I. H. (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.

[21] H. Wang, J. S. Dong, and J. Sun. Checking and Reasoning about Semantic Web through Alloy. In *Proceedings of Formal Methods Europe: FME’03*, volume 2805 of *Lect. Notes in Comput. Sci.*, pages 796–814, Pisa, Italy, Sept. 2003. LNCS, Springer-Verlag.

[22] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall International, 1996.

[23] World Wide Web Consortium (W3C). OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/, Mar. 2003.

[24] World Wide Web Consortium (W3C). Web Ontology Language (OWL) Use Cases and Requirements. http://www.w3.org/TR/webont-req/, Mar. 2003.

A. Z Semantics for DAML+OIL

A.1. Basic concepts

This section defines how basic language constructs in DAML+OIL are modeled in Z. Note that `Thing` is the super class of all classes and that `Nothing` is the sub class of all classes.

$$[Resource] \qquad | \; DataType : \mathbb{P}\, Class$$

$$\begin{array}{|l} Class : \mathbb{P}\, Resource \\ Property : \mathbb{P}\, Resource \\ \hline Class \cap Property = \{\} \end{array} \qquad \begin{array}{|l} Thing, Nothing : Class \\ \hline \forall r : Resource \bullet \\ r \in instances(Thing) \\ r \notin instances(Nothing) \end{array}$$

$$| \; instances : Class \rightarrow \mathbb{P}\, Resource$$

A.2. Class elements

This section presents the Z model of DAML+OIL language constructs that model the inter-class relationships.

$$subClassOf, disjointWith, sameClassAs : Class \leftrightarrow Class$$

$$\forall c_1, c_2 : Class \bullet$$
$$c_1 \ \underline{subClassOf}\ c_2 \Leftrightarrow instances(c_1) \in \mathbb{P}\, instances(c_2)$$
$$c_1 \ \underline{disjointWith}\ c_2 \Leftrightarrow instances(c_1) \cap instances(c_2) = \varnothing$$
$$c_1 \ \underline{sameClassAs}\ c_2 \Leftrightarrow instances(c_1) = instances(c_2)$$

$$intersectionOf, unionOf : \operatorname{seq} Class \leftrightarrow Class$$

$$\forall cl : \operatorname{seq} Class;\ c : Class \bullet$$
$$cl \ \underline{intersectionOf}\ c \Leftrightarrow$$
$$\quad instances(c) = \bigcap \{x : \operatorname{ran} cl \bullet instances(x)\}$$
$$cl \ \underline{unionOf}\ c \Leftrightarrow$$
$$\quad instances(c) = \bigcup \{x : \operatorname{ran} cl \bullet instances(x)\}$$

$$disjointUnionOf : \operatorname{seq} Class \leftrightarrow Class$$

$$\forall cl : \operatorname{seq} Class;\ c : Class \bullet cl \ \underline{disjointUnionOf}\ c \Leftrightarrow$$
$$\quad cl \ \underline{unionOf}\ c \land$$
$$\quad (\forall x, y : \operatorname{ran} cl \bullet x \neq y \Rightarrow x \ \underline{disjointWith}\ y)$$

A.3. Property restrictions

A property restriction defines for a class whose instances satisfy certain restriction.

$$sub\_val : Property \rightarrow (Resource \leftrightarrow Resource)$$

$$toClass : (Class \times Property) \leftrightarrow Class$$

$$\forall c_1, c_2 : Class;\ p : Property \bullet (c_1, p) \ \underline{toClass}\ c_2 \Leftrightarrow$$
$$(\forall a_1, a_2 : Resource \bullet a_1 \in instances(c_1) \Leftrightarrow$$
$$\quad ((a_1, a_2) \in sub\_val(p) \Rightarrow a_2 \in instances(c_2)))$$

$$hasValue : (Class \times Property) \leftrightarrow Resource$$

$$\forall c : Class;\ p : Property;\ r : Resource \bullet$$
$$(c, p) \ \underline{hasValue}\ r \Leftrightarrow$$
$$\quad (\forall a : instances(c) \bullet (a, r) \in sub\_val(p))$$

$$hasClass : (Class \times Property) \leftrightarrow Class$$

$$\forall c_1, c_2 : Class;\ p : Property \bullet (c_1, p) \ \underline{hasClass}\ c_2 \Leftrightarrow$$
$$(\forall a : instances(c_1) \bullet$$
$$\quad sub\_val(p)(\!|\ \{a\}\ |\!) \cap instances(c_2) \neq \varnothing)$$

$$cardinality, maxCardinality, minCardinality :$$
$$\quad (Class \times Property) \rightarrow \mathbb{N}$$

$$\forall c : Class;\ p : Property;\ n : \mathbb{N} \bullet$$
$$cardinality(c, p) = n \Leftrightarrow$$
$$\quad (\forall a : instances(c) \bullet \#(sub\_val(p)(\!|\ \{a\}\ |\!)) = n)$$
$$maxCardinality(c, p) = n \Leftrightarrow$$
$$\quad (\forall a : instances(c) \bullet \#(sub\_val(p)(\!|\ \{a\}\ |\!)) \leq n)$$
$$minCardinality(c, p) = n \Leftrightarrow$$
$$\quad (\forall a : instances(c) \bullet \#(sub\_val(p)(\!|\ \{a\}\ |\!)) \geq n)$$

$$cardinalityQ, maxCardinalityQ, minCardinalityQ :$$
$$\quad (Class \times Property \times Class) \rightarrow \mathbb{N}$$

$$\forall c_1, c_2 : Class;\ p : Property;\ n : \mathbb{N} \bullet$$
$$cardinalityQ(c_1, p, c_2) = n \Leftrightarrow$$
$$\quad (\forall a : instances(c_1) \bullet$$
$$\quad \#(sub\_val(p)(\!|\ \{a\}\ |\!) \cap instances(c_2)) = n)$$
$$maxCardinalityQ(c_1, p, c_2) = n \Leftrightarrow$$
$$\quad (\forall a : instances(c_1) \bullet$$
$$\quad \#(sub\_val(p)(\!|\ \{a\}\ |\!) \cap instances(c_2)) \leq n)$$
$$minCardinalityQ(c_1, p, c_2) = n \Leftrightarrow$$
$$\quad (\forall a : instances(c_1) \bullet$$
$$\quad \#(sub\_val(p)(\!|\ \{a\}\ |\!) \cap instances(c_2)) \geq n)$$

A.4. Property elements

In this subsection, inter-property relationships, such as `subPropertyOf`, and properties about properties, such as `TransitiveProperty`, are defined.

$$subPropertyOf, samePropertyAs, inverseOf :$$
$$\quad Property \leftrightarrow Property$$

$$\forall p_1, p_2 : Property \bullet$$
$$p_1 \ \underline{subPropertyOf}\ p_2 \Leftrightarrow$$
$$\quad sub\_val(p_1) \in \mathbb{P}\, sub\_val(p_2)$$
$$p_1 \ \underline{samePropertyAs}\ p_2 \Leftrightarrow$$
$$\quad sub\_val(p_1) = sub\_val(p_2)$$
$$p_1 \ \underline{inverseOf}\ p_2 \Leftrightarrow$$
$$\quad sub\_val(p_1) = (sub\_val(p_2))^{\sim}$$

$$domain, range : Property \leftrightarrow Class$$

$$\forall p : Property;\ c : Class \bullet$$
$$p \ \underline{domain}\ c \Leftrightarrow \operatorname{dom}(sub\_val(p)) \in \mathbb{P}\, instances(c)$$
$$p \ \underline{range}\ c \Leftrightarrow \operatorname{ran}(sub\_val(p)) \in \mathbb{P}\, instances(c)$$

$$TransitiveProperty : \mathbb{P}\, Property$$

$$\forall p : Property \bullet p \in TransitiveProperty \Leftrightarrow$$
$$(\forall x, y, z : Resource \bullet$$
$$(x, y) \in sub\_val(p) \land (y, z) \in sub\_val(p) \Rightarrow$$
$$\quad (x, z) \in sub\_val(p))$$

$$UniqueProperty : \mathbb{P}\, Property$$

$$\forall p : Property \bullet p \in UniqueProperty \Leftrightarrow$$
$$(\forall x, y, z : Resource \bullet$$
$$(x, y) \in sub\_val(p) \land (x, z) \in sub\_val(p) \Rightarrow$$
$$\quad y = z)$$

$$UnambigousProperty : \mathbb{P}\, Property$$

$$\forall p : Property \bullet p \in UnambigousProperty \Leftrightarrow$$
$$(\forall x, y, z : Resource \bullet$$
$$(x, z) \in sub\_val(p) \land (y, z) \in sub\_val(p) \Rightarrow$$
$$\quad x = y)$$