Analysing Web Ontology in Alloy: A Military Case Study

Jin Song Dong Jun Sun Hai Wang
School of Computing,
National University of Singapore,
{dongjs,sunjing,wangh}@comp.nus.edu.sg

Chew Hung Lee Hian Beng Lee
Center for Decision Support
Defence Science Organisation, Singapore
{lhianben,lchewhun}@dso.org.sg

Abstract

Correctness is the essential requirement criteria for military web ontology based information systems. Reasoning and consistency checking can be useful at many stages during the design, maintenance and deployment of Semantic Web (SW) ontology. Formal methods can provide automatic reasoning and consistency checking services for SW. In this paper, we use military plan ontology as a case study to demonstrate how Alloy can be applied to check SW ontology.

keywords: *Semantic Web, Alloy, DAML+OIL, military ontology*

1 Introduction

For mission planning, military commanders often need to get timely updates on the composition and strength of army units. A research team at DSO National Laboratories in Singapore has recently developed a *plan ontology* [10] using Semantic Web (SW) [2] languages.

It is essential that this military web-based ontology is consistent and correct. Reasoning can be useful at many stages during the design, maintenance and deployment of ontology. The software modeling language Alloy [8] is a first order declarative language based on relations. We believe SW is a new novel application domain for Alloy as relationships between web resources are the focus points in SW. Furthermore, Alloy specifications can be analyzed automatically using the Alloy Analyzer (AA). Given a finite scope for a specification, AA translates it into a propositional formula and uses SAT solving technology to generate instances that satisfy the properties expressed in the specification. We believe that if the semantics of the SW languages can be encoded into then Alloy can be used to provide automatic reasoning and consistency checking services for SW. Various reasoning tasks can be supported effectively by AA. Some of those reasoning tasks are unique for AA, as currently no other SW reasoning tools can support those tasks.

The remainder of the paper is organized as follows. Section 2 briefly introduces the Alloy and Semantic Web. In section 3 semantic domain and functions for the DAML+OIL Web Ontology Language [12] constructs are defined in Alloy. Section 4 presents the transformation techniques from DAML+OIL to Alloy with applications to military plan ontology. Section 5 demonstrates various reasoning and checking tasks using DAML+OIL-Alloy tool. Section 6 concludes the paper.

2 Alloy and Semantic Web overview

2.1 Alloy overview

Alloy [8] is a structural modelling language based on first-order logic, for expressing complex structural constraints and behavior. Alloy treats relations as first class citizens and uses relational composition as a powerful operator to combine various structured entities. The Alloy Analyzer (AA) is a tool for analyzing models written in Alloy. Given a formula and a scope – a bound on the number of atoms in the universe – AA determines whether there exists a model of the formula that uses no more atoms than the scope permits, and if so, return it. It supports two kinds of automatic analysis: simulation, in which the consistency of an invariant or operation is demonstrated by generating a state or transition, and checking, in which a consequence of the specification is tested by attempting to generate a counterexample.

2.2 Semantic web overview

The Semantic Web is a vision for a new kind of Web with enhanced functionality which will require semantic-based representation and processing of Web information. W3C has proposed a series of technologies that can be applied to achieve this vision. The Semantic Web extends the current

Web by giving the web content a well-defined meaning, better enabling computers and people to work in cooperation.

Resource Description Framework (RDF) [9] is a foundation for processing metadata; it provides interoperability between applications that exchange program-understandable information on the Web. RDF Schema provides the basic vocabulary to describe RDF documents. Similar to XML Schema which give specific constraints on the structure of an XML document, RDF Schema provides information about the interpretation of the RDF statements. The DARPA Agent Markup Language (DAML) [12] is an AI-inspired description logic-based language. DAML combined with Ontology Interchange Language (OIL) and features from other ontology systems is called DAML+OIL and contains richer modelling primitives than RDF.

3 DAML+OIL semantic encoding

DAML+OIL has a well-defined semantics which has been described in a set of axioms [6]. In this section based on the semantics of DAML+OIL, we define the semantic functions for some important DAML+OIL primitives in Alloy. The complete DAML+OIL semantic encoding can be found in [5].

3.1 Basic concepts

The semantic models for DAML+OIL are encoded in the module `DAMLOIL`. Users only need to import this module to reason DAML+OIL ontology in Alloy. All the things described in Semantic web context are called resources. A basic type `Resource` is defined as:

```
module DAMLOIL
sig Resource {}
```

All other concepts defined later are extended from the `Resource`.

The class corresponds to the generic concept of type or category of resource. Each `Class` maps a set of resources via the relation `instances`, which contains all the instance resources. The keyword `disj` is used to indicate the `Class` and `Property` are disjoint.

```
disj sig Class extends Resource
    {instances: set Resource}
```

3.2 Class elements

The `subClassOf` is a relation between classes. The instances in a subclass are also in the superclasses. A parameterized formula (a function in Alloy) is used to represent this concept.

```
fun subClassOf(csup, csub:  Class)
    {csub.instances in csup.instances}
```

3.3 Property restrictions

A `toClass` function states that all instances of the class `c1` have the values of property *P* all belonging to the class `c2`.

```
fun toClass (p: Property, c1: Class, c2: Class)
    {all r1, r2: Resource |
            r1 in c1.instances <=>
            r2 in r1.(p.sub_val) =>r2 in c2.instances}
```

3.4 Property elements

The `subPropertyOf` function states that `psub` is a subproperty of the property `psup`. This means that every pair (subject,value) that is in `psup` is in the `psub`.

```
fun subPropertyOf (psup, psub: Property)
    {psub.sub_val in psup.sub_val}
```

4 DAML+OIL to Alloy Transformation

In the previous section we defined the semantic model for the DAML+OIL constructs, so that analyzing DAML+OIL ontology in Alloy can be easily and effectively achieved. We also developed a Java program for the automatic transformation from DAML+OIL file into Alloy model. The details of the Java program and other information on this project can be found at:
http://nt-appn.comp.nus.edu.sg/fm/alloy/.

A set of transformation rules transforming from DAML+OIL ontology to Alloy program are developed. For example, the following rule show *DAML + OIL* class transformation.

$$\frac{C \in DAML\_class}{static\ disj\ sig\ C\ extends\ Class\{\}}$$

A DAML_class `C` will be transferred into a scalar `C`, constrained to be an element of the signature `Class`.

4.1 Application to the military web ontology

A research team at DSO National Laboratories (Singapore) has developed a ontology in DAML+OIL for representing plans [10]. The purpose of this ontology is to build a description of the military units used in planning. It contains semantically linked information on the capability of military units and the command/control relationships between the units.

The partial military ontology will be used to illustrate how the transformation and analysis could be achieved.

The following DAML+OIL ontology defines a class `ModernMilitaryUnit` [1] which represents all the functionally independent military forces. Most large

modern militaries are traditionally broken into several branches of groups. E.g. the `navy unit` and `air force units`. The two disjoint classes `NavalUnit` and `AirForceUnit` are the subclasses of `ModernMilitaryUnit`. A `military task` is an independent entity that is assigned to a particular unit to execute. A unit can execute a military task. The `airForceTask` and `navalTask` is a military task assigned to air force and navy respectively.

```
<daml:Class rdf:ID="ModernMilitaryUnit">
 <rdfs:label>ModernMilitaryUnit</rdfs:label> </daml:Class>
<daml:Class rdf:ID="NavalUnit">
  <rdfs:label>NavalUnit</rdfs:label>
   <rdfs:subClassOf rdf:resource="#ModernMilitaryUnit"/>
   </daml:Class>
<daml:Class rdf:ID="AirForceUnit">
  <rdfs:label>AirForceUnit</rdfs:label>
  <rdfs:subClassOf rdf:resource="#ModernMilitaryUnit"/>
  <daml:disjointWith rdf:resource="#NavalUnit"/>
  </daml:Class>
 <daml:ObjectProperty rdf:about="assignTo">
  <rdfs:label>assignTo</rdfs:label> </daml:ObjectProperty>
<daml:ObjectProperty rdf:about="execute">
   <rdfs:label>execute</rdfs:label>
   <daml:inverseOf rdf:resource="#assignTo"/>
   </daml:ObjectProperty>
<daml:Class rdf:ID="militaryTask">
   <rdfs:label>militaryTask</rdfs:label> </daml:Class>
<daml:Class rdf:ID="airForceTask">
 <rdfs:label>airForceTask</rdfs:label>
  <rdfs:subClassOf rdf:resource="#militaryTask"/>
  <rdfs:subClassOf> <daml:Restriction>
   <daml:onProperty rdf:resource="#assignTo"/>
     <daml:toClass rdf:resource="#AirForceUnit"/>
   </daml:Restriction> </rdfs:subClassOf></daml:Class>
 <daml:Class rdf:ID="navalTask">
   <rdfs:label>navalTask</rdfs:label>
  <rdfs:subClassOf rdf:resource="#militaryTask"/>
  <rdfs:subClassOf><daml:Restriction>
     <daml:onProperty rdf:resource="#assignTo"/>
     <daml:toClass rdf:resource="#NavalUnit"/>
   </daml:Restriction> </rdfs:subClassOf></daml:Class>
```

This DAML+OIL ontology will be transferred into Alloy as follow,

```
module military
/*import the library module we defined*/
open DAMLOIL
/* ModernMilitaryUnit is transferred to a class instance,
the key word static is used to a signature contains
exactly one element. */
static disj sig ModernMilitaryUnit extends Class {}
/* NavalUnit and AirForceUnit are transfer to
   two class instance */
static disj sig AirForceUnit, NavalUnit extends Class {}
/* The disjoin and subclass element was
  transferred into fact in Alloy */
fact {disjointWith(AirForceUnit, NavalUnit)}
fact {subClassOf(ModernMilitaryUnit, AirForceUnit)}
fact {subClassOf(ModernMilitaryUnit, NavalUnit)}
/*assignTo, execute are transfer to  instances*/
static disj sig execute, assignTo extends Property {}
fact {inverseOf(execute, assignTo)}
static disj sig militaryTask,airForceTask,
        navalTask extends Class{}
fact{subClassOf(militaryTask, airForceTask)}
fact{subClassOf(militaryTask, navalTask)}
fact{toClass(assignTo, navalTask, NavalUnit)
   && toClass(assignTo, airForceTask,AirForceUnit)}
```

We can check the consistency of the DAML+OIL ontology and do some reasoning readily.

5 Analyze DAML+OIL ontology

Reasoning is one of the key tasks for semantic web. It can be useful at many stages during the design, maintenance and deployment of ontology.

There are two different levels of checking and reasoning, the conceptual level and the instance level. At the conceptual level, we can reason about class properties and subclass relationships. At the instance level, we can do the membership checking (instantiation) and instance property reasoning. The DAML+OIL reasoning tool, i.e. FaCT [7], can only provide conceptual level reasoning, while AA can perform both. The Fact system originally is designed to be a terminological classifer (TBox) which concerns only about the concepts, roles and attributes, not instances. The semantic web reasoner based on the Fact, like OILED, dose not support instance level reasoning well.

5.1 Class property checking

It is essential that the ontology shared among autonomous software agents is conceptually consistent. Reasoning with inconsistent ontology may lead to erroneous conclusions. In this section we give some examples of inconsistent ontology that can arise in ontology development, and demonstrate how these inconsistencies can be detected by the Alloy analyzer. For example, we define another class `submarineUnit` which is a subclass of `NavalUnit` and executed the `bombTask` which is a subclass of `airForceTask`. There is an inconsistency since by the ontology definition airForceTask can only be assigned to AirForceUnit. The NavalUnit and AirForceUnit are disjoint.

```
<daml:Class rdf:ID="bombTask">
 <rdfs:label>bombTask</rdfs:label>
 <rdfs:subClassOf rdf:resource="#airForceTask"/>
</daml:Class>
<daml:Class rdf:ID="submarineUnit">
 <rdfs:label>submarineUnit</rdfs:label>
 <rdfs:subClassOf rdf:resource="#NavalUnit"/>
 <rdfs:subClassOf><daml:Restriction>
   <daml:onProperty rdf:resource="#execute"/>
   <daml:hasValue rdf:resource="#bombTask"/>
 </daml:Restriction></rdfs:subClassOf></daml:Class>
```

We transform the ontology into an Alloy program, add some facts to remove the trivial models (like everything type is empty set) and load the program into the Alloy Analyzer. The Alloy Analyzer will automatically check the consistency. We conclude that there is an inconsistency in the military ontology since Alloy cannot find any solution satisfying all facts within the scope (Figure 1). Note that when Alloy can't find a solution, it may also be due to the scope being too small. The AA performs the analysis within certain scope which constrains the number of items each basic type have. By picking a large enough scope, "no solution found" is very likely to mean that an inconsistency has occurred.

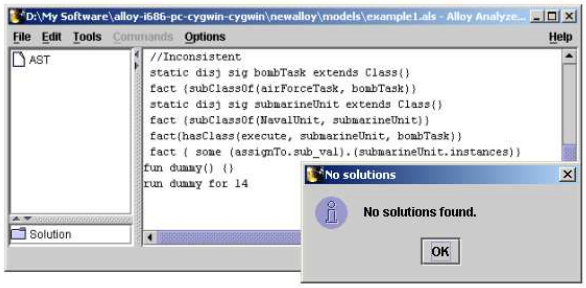

Figure 1. Inconsistence example

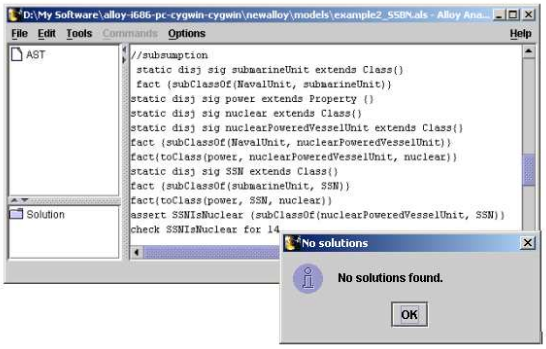

Figure 2. Subsumption example

5.2 Subsumption reasoning

The task of subsumption reasoning is to infer a DAML+OIL class is the subclass of another DAML+OIL class. For example, in the military ontology a property `power` is defined. A `nuclearPoweredVesselUnit` class is a subclass of the `NavalUnit` which uses nuclear-powered vessels as their equipment.

```
<daml:ObjectProperty rdf:ID="power"/>
<daml:Class rdf:ID="nuclear">
 <rdfs:label>nuclear</rdfs:label></daml:Class>
<daml:Class rdf:ID="nuclearPoweredVesselUnit">
 <rdfs:label>nuclearPoweredVesselUnit</rdfs:label>
 <rdfs:subClassOf rdf:resource="#NavalUnit"/>
<rdfs:subClassOf>
 <daml:Restriction>
  <daml:onProperty rdf:resource="#power"/>
  <daml:toClass rdf:resource="#nuclear"/>
 </daml:Restriction></rdfs:subClassOf></daml:Class>
```

We also define a class `SSBN` (ballistic missile nuclear-powered (SSBN) submarines), a subclass of `submarineUnit` which use `nuclear` reactor to supply the power.

```
<daml:Class rdf:ID="SSBN">
 <rdfs:label>SSBN</rdfs:label>
 <rdfs:subClassOf rdf:resource="#submarineUnit"/>
 <rdfs:subClassOf>
  <daml:Restriction>
  <daml:onProperty rdf:resource="#power"/>
 <daml:toClass rdf:resource="#nuclear"/>
 </daml:Restriction></rdfs:subClassOf>
</daml:Class>
```

Several of the classes were upgraded to being defined when their definitions constituted both necessary and sufficient conditions for class membership, e.g., a `NavalUnit` unit is a `nuclearPoweredVesselUnit` if and only if it use `nuclear` reactor to supply the power. Additional subclass relationships can be inferred i.e. the `SSBN` is also a subclass of `nuclearPoweredVesselUnit`. We transfer this ontology into an Alloy program and make an assertion that the `SSBN` is the subclass of `nuclearPoweredVesselUnit`. The Alloy analyzer will check the correctness of this assertion automatically (Figure 2). The Alloy Analyzer checks whether an assertion holds by trying to find a counterexample. Note that "no solution" means no counterexamples is found, in this case, it indicates that the assertion is sound.

5.3 Instantiation

Instance level reasoning is one of the main contributions for reasoning over DAML+OIL ontology using Alloy. Nowadays many successful DAML+OIL reasoners like FaCT are based on description logics (DL), which lacks support for instances. In Alloy every expression denotes a relation. The scalars will be represented by singleton unary relations - that is, relations with one column and one row. The instance level reasoning can be supported readily in Alloy.

Instantiation is a reasoning task which tries to check if an individual is an instance of a class. For example, we define a resource `aDeepSeaPatrol` as an instance of `militaryTask` which is assigned to the submarine `SSN17`. People may want to check if `aDeepSeaPatrol` is a `navalTask`.

```
 <submarineUnit rdf:ID="SSN17">
  <rdfs:label>SSN17</rdfs:label></submarineUnit>
 <militaryTask rdf:ID="aDeepSeaPatrol">
  <rdfs:label>aDeepSeaPatrol</rdfs:label>
  <assignTo rdf:resource="SSN17"/></militaryTask>
```

We transfer the ontology into an Alloy program and make an assertion as following:

```
static disj sig SSN17,
     aDeepSeaPatrol extends Resource{}
fact {SSN17 in submarineUnit.instances &&
      aDeepSeaPatrol in militaryTask.instances}
fact {(aDeepSeaPatrol->SSN17) in assignTo.sub_val}
 assert isDeepSeaPatrolnavalTask
 {(aDeepSeaPatrol in navalTask.instances)}
 check isDeepSeaPatrolnavalTask for 14
```

AA concludes that this assertion is correct.

5.4 Instance property reasoning

Instance property reasoning (often regarded as knowledge querying) is important in Semantic Web applications. Since one of the promising strengths of Semantic Web technology is that it gives the agents the capability to do more

accurate and more meaningful searches. The agent can answer some questions for which the answer is not explicitly stored in the knowledge base.

For example, the `subordinateTo` and `controlOver` are two properties, which are inverse to each other. Military unit `A subordinateTo B` if `B` has administrative, logistics and command control over the `A`. `subordinateTo` is transitive. Three military unit `aCompany`, `aBattalion` and `aBrigade` are defined. `aCompany subordinateTo aBattalion` and `aBattalion subordinateTo aBrigade`. One possible question people may ask is that whether `aBrigade` is `controlOver aCompany`. With the assistance of Alloy reasoner, agents can answer such questions.

```
fact
 {TransitiveProperty(subordinateTo)}
static disj sig aCompany, aBattalion,
       aBrigade extends Resource{}
fact {aCompany in ModernMilitaryUnit.instances
 && aBattalion in ModernMilitaryUnit.instances
 && aBrigade in ModernMilitaryUnit.instances}
fact {(aCompany->aBattalion) in subordinateTo.sub_val}
fact {(aBattalion->aBrigade) in subordinateTo.sub_val}
assert hum {(aBrigade->aCompany) in controlOver.sub_val}
check hum for 14
```

6 Conclusion

Reliability of web-based military information system requires techniques and tools for reasoning about military web ontology. In this paper, we have constructed the semantic models for DAML+OIL language constructs in Alloy and developed the systematic transformation rules and a Java program which can translate DAML+OIL ontology to Alloy automatically. With the assistance of Alloy Analyzer (AA), we also demonstrated that the consistency of the military web-based plan ontology can be checked automatically and different kinds of reasoning tasks can be supported[1].

As Alloy is based on a relational logic, where relations between web resources are the central points in DAML, we believe reasoning web ontology is a new novel application domain for Alloy.

Recently, XML environment for formal design techniques has been developed, e,g. [11]. RDF and DAML+OIL has also been used to construct a Semantic Web environment for supporting, extending and integrating various specification languages [3]. However there has not been much work done on the application of formal design technique/tools for Semantic Web (SW). In our previous work [4], we tried to extract DAML+OIL web ontology from Z requirement models, which is a very different approach from the techniques demonstrated in this paper – checking and reasoning web ontology by encoding the semantics of DAML+OIL into the Alloy system. We believe the DAML+OIL-Alloy transformation techniques and tool can complement the existing Web ontology reasoning tools, i.e. FaCT.

[1]The technique/tool developed here can be used for checking not only military domain but also other domains.

Acknowledgements

We would like thank Daniel Jackson for providing many useful info and demo on Alloy. We also like thank Khee Yin How, Kum Lan Chan and Chew Lock Pin for valuable comments. This work is supported by the Defence Innovative Research Grant *Formal Design Methods and DAML* from Defence Science & Technology Agency (DSTA) Singapore.

References

[1] William Andersen and Brian Peterson. An ontology of modern military organizations and their structure. In *Workshop on the IEEE Standard Upper Ontology (IJCAI'01)*, 2001.

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

[3] 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.

[4] J. S. Dong, J. Sun, and H. Wang. Z Approach to Semantic Web Services. In *International Conference on Formal Engineering Methods (ICFEM'02)*, Shanghai, China, October 2002. LNCS, Springer-Verlag.

[5] Jin Song Dong and Hai Wang. Checking and reasoning about semantic web through alloy. Technical Report TRB9/02, National University of Singapore, 2002.

[6] Richard Fikes and Deborah L. McGuinness. An axiomatic semantics for rdf, rdf schema, and daml+oil. Technical Report KSL-01-01, Knowledge Systems Laboratory, 2001.

[7] I. Horrocks. The FaCT system. *Tableaux'98, Lecture Notes in Computer Science*, 1397:307–312, 1998.

[8] D. Jackson. Micromodels of software: Lightweight modelling and analysis with alloy. Available: http://sdg.lcs.mit.edu/alloy/book.pdf, 2002.

[9] 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.

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

[11] 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.

[12] 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.