A Tools Environment for Developing and Reasoning about Ontologies

Jin Song Dong, Yuzhang Feng, Yuan Fang Li, Jun Sun
School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543,
{dongjs, fengyz, liyf, sunj}@comp.nus.edu.sg

Abstract

Started in the beginning of 2001, the Semantic Web is regarded by many as the next generation of the Web. Ontology languages are the building blocks of Semantic Web as they provide basic vocabularies for data markups: the ontologies. The correctness of shared ontologies is crucial to the proper functioning of agents. Hence ensuring the consistency of ontologies is a central issue in both the design and deployment phases of any Semantic Web-aware application. Our experiences show that Semantic Web is a novel and fruitful application domain for formal languages and their mature reasoning tool support. In order to ease the application of formal methods and tools to the Semantic Web, we developed an integrated tools environment to support systematic development of OWL ontologies and then transformation, reasoning assistance and querying of them.

Keywords: Semantic Web, ontology, reasoning, tool

1. Introduction

The World Wide Web has changed our life in many dramatic ways. However, by and large, the current Web is still a medium for exchange of data and knowledge among *people*. To reach its full potential, Web need to become ubiquitous in the sense that it becomes a globe-scale medium for software agents to interchange knowledge and accomplish tasks on human's behalf. Semantic Web (SW) [3] was envisioned by Tim Berners-Lee as the next generation of the Web. It is necessary in the first place to mark-up data on the Web semantically so that they can be understood and processed by agents autonomously. Ontology languages such as DAML+OIL [30] and OWL [17] provide basic vocabularies for such markups. To support autonomous reasoning of agents, (a large portion of) these languages were designed to be decidable.

The correctness of ontologies are vital to the success of a SW-aware application as software agents are to perform autonomous discovery, processing, reasoning and integration of these ontologies. Various ontology reasoning tools such as FaCT [16] and RACER [14] have been developed to check the consistency of SW ontologies. As these tools were built to reason about decidable languages, they are made fully automated, in the sense that no user intervention is required to perform reasoning and checking tasks.

Decidability of ontology languages does not come at no cost. Ontology languages have description logics as their basis. It has been shown [4] that any description logics formula is expressible by a formula in $\ddot{\mathcal{L}}^3$, the subset of first-order predicate logic that only allow at most 3 variables. This inherent expressiveness limitation makes it difficult or even impossible for ontology languages to model certain complex properties that might be crucial to the correctness of ontologies. Our experiences showed that SW can be a new application domain for formal languages and tools such as Z [31], Alloy [20] and PVS (Prototype Verification System [26]). Z [31] is a state based formal language based on set theory and first-order predicate logic. It is by nature more expressive than ontology languages and hence can capture more properties. Alloy [20] can be viewed as a subset of Z and it has a fully automated reasoner that can pin down the origin of errors in a specification.

In our previous works [10, 9] we defined Alloy & Z semantics for DAML+OIL and applied Alloy Analyzer (AA [21]) and Z/EVES [28], both software engineering (SE) verification tools, to check the consistency of DAML+OIL ontologies. In [8], we have identified some shortcomings of SW and SE tools and proposed a complementary approach involving formal methods and tools. We proposed to use RACER, Z/EVES and AA in combination to effectively and efficiently check for inconsistencies of DAML+OIL and RDF ontologies. Our combined approach was successfully applied to a real-world military plan ontologies case study, where one ontological inconsistency and a total of 23 errors that are beyond the modelling power of DAML+OIL were discovered.

However formal methods usually make extensive use of mathematical concepts and symbols, which often present difficulties for users without the relevant mathematical background. In order to hide the underlying formal methods notations and make the combined approach more friendly to users who are not familiar with the various SW tools, an easy-to-use visual tool that supports automated creating, transforming and querying ontologies is much desired.

In this work, we extend our previous works to the latest ontology language OWL and present an integrated tools environment for developing and reasoning about ontologies, named SESeW (**S**oftware **E**ngineering for **Se**mantic **W**eb), which serves as a front-end to the tools used in our previous works and the extension to OWL under one umbrella. Moreover, in order to make the tool self-containing, we implemented a systematic methodology for creating OWL ontologies, i.e. the Methontology proposed by Fernandez and Gomez-Perez and Juristo [11].

SESeW is developed in Java and has an easy-to-use graphical user interface. It supports step-by-step development of OWL ontologies as well as transforming DAML+OIL/OWL/RDF ontologies to formal specification languages like Z and Alloy and PVS. We also offer useful functionalities to check the well-formedness of ontologies, to perform queries over ontologies, etc. In a nutshell, SESeW offers a complete environment for developing consistent ontologies. Our work is remotely related to works on applying formal methods to the Web domain [27, 32, 24, 5].

This paper describes SESeW in detail. The remainder of the paper is organized as follows. Section 2 briefly discusses the background information of the ontology languages, Z, Alloy and PVS. In Section 2.3, our previous works on reasoning about DAML+OIL are briefly discussed, followed by the natural extension to OWL ontologies. In Section 3, the detailed functionalities of SESeW are presented. Section 4 briefly evaluates the performance of the tool. Finally, Section 5 discusses future works and concludes this paper.

2. Overview

This section is devoted to a brief introduction of the Semantic Web languages and tools, the formal specification languages and their tools, and our previous works on using formal languages and tools as underlying reasoners for DAML+OIL and OWL ontologies.

2.1. Semantic Web Languages & Tools

The Semantic Web [3] is a vision of next generation of the Web. It is believed that in the future, the Web is no longer only for human consumption, rather, it is also a universal medium for software agents to interchange, process and integrate information autonomously and cooperatively to accomplish complex tasks on human's behalf. In order to achieve this, data on the web must be marked-up unambiguously and semantically, by ontologies. Ontologies are expressed in terms of ontology languages, which build upon layers of successful technologies such as XML, URI and RDF [23].

The development of various SW technologies have been coordinated by the World Wide Web Consortium (W3C). RDF (the Resource Description Framework) is a model for meta-data describing data on the web without assumption about the application domain. DAML+OIL [30] is an ontology language based on RDF Schema [6], enforcing syntactic well-formedness and with more language primitives to express constraints. It was designed to be decidable. The Web Ontology Language (OWL) [25], the successor of DAML+OIL, became a W3C recommendation as the ontology language. It takes into consideration of requirements of different user groups and consists of three increasingly expressive sub languages: OWL Lite, DL and Full. OWL Lite and DL are decidable and Full is not (refer to [22] for details).

A number of ontology reasoning tools have been developed to assist in the development of ontologies. Three of them are briefly mentioned in the following. FaCT (**Fa**st **C**lassification of **T**erminologies) [16], developed at University of Manchester, supports automated TBox (concept-level) reasoning, namely class subsumption, consistency and classification reasoning. It does not support ABox (instance-level) reasoning. It can classify a DAML+OIL/OWL ontology by performing subsumption reasoning so as to reduce redundancy and detect any inconsistency within it. FaCT++[1] is a new generation of the FaCT reasoner. It is an OWL Lite reasoner that features a new internal architecture and new optimization. RACER, the **R**enamed **A**Box and **C**oncept **E**xpression **R**easoner [14], supports both TBox and ABox reasoning for the description logic $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D})^-$ [13]. RACER's functionalities include developing ontologies, querying, retrieving and evaluating the knowledge base, etc. It supports RDF, DAML+OIL and OWL. In addition, OilEd [2] is an ontology editor that can use both FaCT and RACER as background reasoner. It supports both DAML+OIL and OWL. RACER and OilEd are used in our combined approach [8].

2.2. Z, Alloy & PVS

Z [31] is a state-based formal specification language based on the established mathematics of set theory and first-order logic. It has been used to specify data and functional

1 `http://owl.man.ac.uk/factplusplus/`

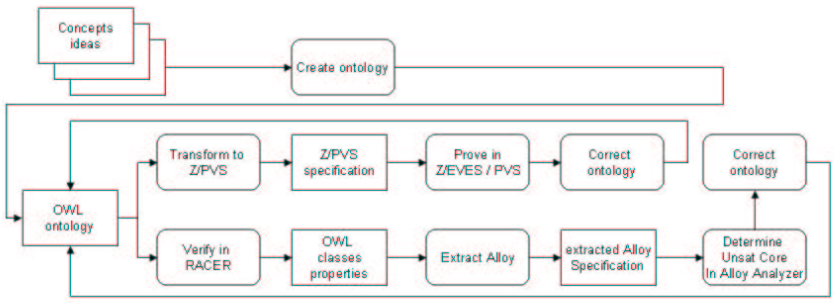


Figure 1. Data Flow Diagram of SESeW

models of a wide range of systems, including transaction processing systems and communication protocols. It has first-order predicate logic as its foundation so it is more expressive than description-logics-based ontology languages such as DAML+OIL and OWL. Z/EVES [20] 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.

Alloy [14] is a structural modelling language emphasizing on automated reasoning support. It treats relations as first class citizens and uses relational composition as a powerful operator to combine various structural entities. The design of Alloy was influenced by Z and hence it can be (roughly) viewed as a subset of Z. Alloy Analyzer (AA) is a constraint solver that provides fully automatic simulation and checking. AA works as a compiler: it compiles a given problem into a (usually huge) boolean formula, which is subsequently solved by a SAT solver, and the solution is then translated back to AA. Inevitably, a scope - a bound on the size of the domains - must be given to make the problem finite. AA determines whether there exists a model of the formula. When AA finds an assertion to be false, it generates a counterexample, which (in some cases) makes tracing the error easier, compared to theorem provers. Similar to Z/EVES, Alloy specifications are in the form of modules, organized into a tree. Existing modules can be reused by commands `open` or `use`.

PVS [26] provides an expressive specification language that augments classical high-order logic with a sophisticated type system with predicate subtypes and dependent types with parameterized theories. Therefore, we may express and verify properties that are not expressible in OWL or Z or Alloy.

2.3. SE Approach to Reasoning about Ontologies

Ontology reasoners such as RACER and FaCT have been developed to reason ontologies with a high degree of automation. However, complex ontology-related properties may not be expressible within the current web ontology languages, consequently they may not be checkable by RACER and FaCT. Hence, in our previous work [8], a combined approach to reason ontologies was proposed. We proposed to use the SE (Software Engineering) techniques and tools, i.e. Z/EVES and AA, to complement the ontology tools for checking Semantic Web documents. In this approach, Z/EVES is first applied to remove trivial syntax and type errors of the ontologies. Next, RACER is used to identify any ontological inconsistencies, whose origins can be traced by AA. Finally Z/EVES is used again to express complex ontology-related properties and reveal errors beyond the modelling capabilities of the current web ontology languages. We have successfully applied this approach to check a set of military plan ontologies. In another work [7], we used PVS to reason about both OWL and OWL Rules Language (now called SWRL [18]) which extends OWL with Horn-like rules.

In this work, the approach is extended to handle OWL ontologies by defining the Z and Alloy semantics for OWL ontology language. We skip the details of the semantics in this paper as it is very similar to the ones presented in [8].

3. The Tools Environment: SESeW

This section is devoted to an introduction of our integrated tools environment for developing and reasoning DAML+OIL and OWL ontologies. Figure 2 shows the main window of SESeW, with a military-domain OWL ontology opened. It has four tabbed text areas for ontologies, Z, Alloy and PVS specifications respectively. Transformed Z, Alloy and PVS specifications are displayed in the respective text areas. Figure 3 illustrates the data flow in SESeW. In the following sections, the details of the main functionalities are presented.

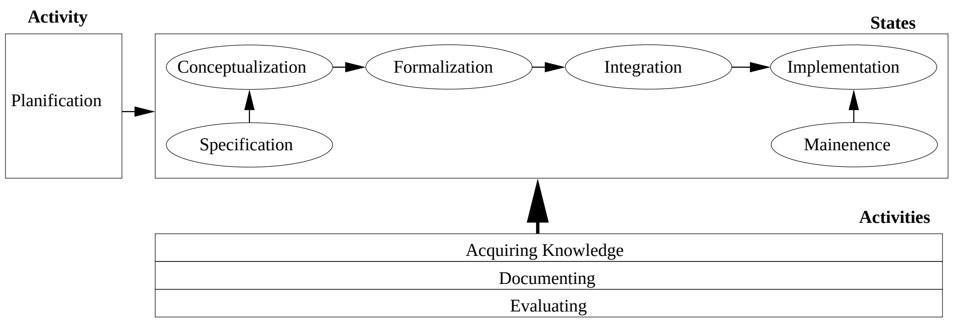
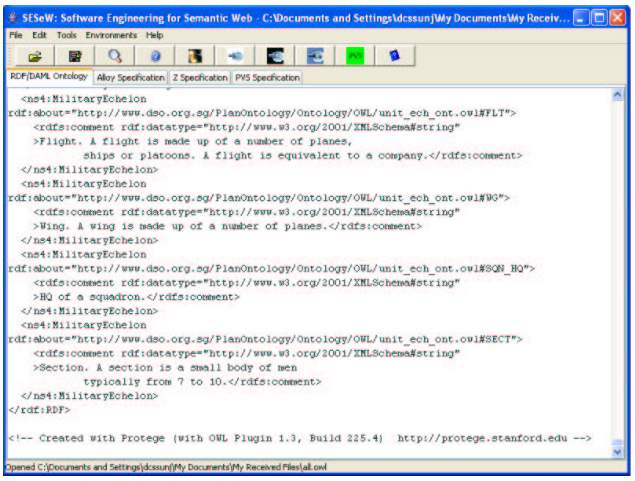


Figure 3. Flow of Ontology Creation


SESeW: Software Engineering for Semantic Web - C:\Documents and Settings\dcssunj\My Documents\My Receiv...
File Edit Tools Environments Help
RDF/DAML Ontology
Alloy Specification
Z Specification
PVS Specification

```
  <ns4:MilitaryEchelon
rdf:about="http://www.dso.org.sg/PlanOntology/Ontology/OWL/unit_ech_ont.owl#FLT">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Flight. A flight is made up of a number of planes,
            ships or platoons. A flight is equivalent to a company.</rdfs:comment>
  </ns4:MilitaryEchelon>
  <ns4:MilitaryEchelon
rdf:about="http://www.dso.org.sg/PlanOntology/Ontology/OWL/unit_ech_ont.owl#WG">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Wing. A wing is made up of a number of planes.</rdfs:comment>
  </ns4:MilitaryEchelon>
  <ns4:MilitaryEchelon
rdf:about="http://www.dso.org.sg/PlanOntology/Ontology/OWL/unit_ech_ont.owl#SQN_HQ">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >HQ of a squadron.</rdfs:comment>
  </ns4:MilitaryEchelon>
  <ns4:MilitaryEchelon
rdf:about="http://www.dso.org.sg/PlanOntology/Ontology/OWL/unit_ech_ont.owl#SECT">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Section. A section is a small body of men
            typically from 7 to 10.</rdfs:comment>
  </ns4:MilitaryEchelon>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 1.3, Build 225.4)  http://protege.stanford.edu -->
```

Opened C:\Documents and Settings\dcssunj\My Documents\My Received Files\all.owl

Figure 2. Main Window of SESeW

A user may load an existing ontology created using other editors like Protégé [12], in which case SESeW provides a standard text editing environment and functionality for editing the ontologies. Simple validation functions like well-formedness checking are offered to make sure the syntactical correctness of the ontologies.

3.1. Ontology Creation

We implemented a systematic methodology for creating ontologies, namely Methontology [11]. Basically, Methontology is a set of activities in ontology development process, a life cycle to build ontologies based on evolving prototypes, and a well-structured methodology used to build ontologies from scratch. The ontology life cycle contains the following states: specification, conceptualization, integration, implementation, and maintenance (Figure 3, borrowed from [11]). The specification phase is to produce either an informal, semi-formal or formal ontology specification document either in natural language, or as a set of intermediate representations or using competency questions. In the conceptualization phase, the domain knowledge is structured in a conceptual model. A complete glossary of terms, i.e. concepts, instances, verbs, and properties, is built. The integration phase speeds up the construction of the ontology by considering reuse of definitions already built into other ontologies. Ontology implementation requires the use of an environment that supports the meta-ontology and ontologies selected at the integration phase. Knowledge acquisition is an independent activity in the ontology development process. Experts, books, handbooks, figures, tables and even other ontologies are sources of knowledge from which the knowledge can be elucidated.

In SESeW, we assume the existence of a list of gathered terms from text files or other ontologies generated using knowledge acquisition techniques such as text analysis, structured interview or brainstorming. In the conceptualization phase, users are required to identify the classes from a list of possible classes, the instances of each class, and the relationships between individuals and classes. Properties are distinguished by whether they relate individuals to individuals or individuals to datatypes. Datatype properties may range over RDF literals or simple XML Schema datatypes. To create a datatype property, users are required to provide a property name by either selecting one from the Glossary of Terms or typing a name into the text field. The user then selects the property domain and range. Figure 4 shows the window for introducing new datatype properties. A datatype property can be a $FunctionalProperty$ which has the following definition.

$$P(x,y) \wedge P(x,z) \Rightarrow y = z$$

The creation of object properties is similar to the creation of datatype properties, except that an object property has classes as its range (instead of datatypes). Besides $FunctionalProperty$, it may be of three other property types: $TransitiveProperty$, $SymmetricProperty$, and $InverseFunctionalProperty$. The respective definitions for the three property characteristics are:

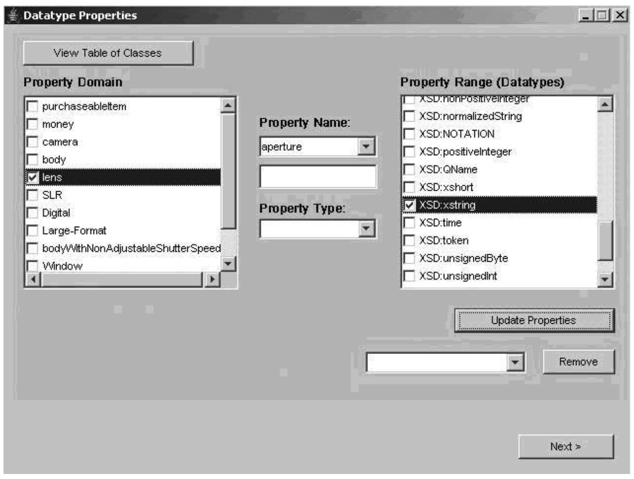


Figure 4. Creating Datatype Property

$$
\begin{array}{rcl}
P(x,y) \wedge P(y,z) & \Longrightarrow & P(x,z) \\
P(x,y) & \Longleftrightarrow & P(y,x) \\
P(y,x) \wedge P(z,x) & \Longrightarrow & y = z
\end{array}
$$

In addition to designating property characteristics, it is possible to further constrain classes with property restrictions. The six types of restrictions defined in OWL are all supported in SESeW:

- $hasValue$: which allows users to restrict classes by requiring the existence of particular property values.
- $allValuesFrom$: which requires that for every instance of the class that has the specified property, the values of the property are all members of the class indicated by the $allValuesFrom$ clause.
- $someValuesFrom$: which requires that for every instance of the class that has the specified property, at lease one value of the property is a member of the class indicated by the $someValueFrom$ clause.
- $cardinality$: which permits the specification of exactly the exact number of elements in a relation.
- $minCardinality$: which permits the specification of the minimum number of elements in a relation.
- $maxCardinality$: which permits the specification of the maximum number of elements in a relation.

Lastly, the ontology is generated automatically (shown in the its text area and saved into the file designated).

3.2. Ontology Querying

A friendly user interfaceis provided for querying a given ontology. A user may input queries in an SQL-like language RDQL [1]. The query engine is a part of the ontology toolkit, the Jena Framework [19], which is bundled with SESeW. An RDF model can be viewed as a graph, often expressed as a set of triples. An RDQL consists of a graph pattern, expressed as a list of triple patterns. Each triple pattern is comprised of named variables and RDF values (URIs and literals). An RDQL query can additionally have a set of constraints on the values of those variables, and a list of the variables required in the answer set. A typical query would have the structure "`SELECT...WHERE...USING...`", where ontology entities of interest are specified after the keyword `SELECT` along with constraints after the keyword `WHERE` in the namespace given after the keyword `USING`.

As this tool was initially developed as part of a military-related research project, frequently used query patterns in the military planning domain are categorized and templates are created to ease the creation of such queries. For example, the category "instantiation" provides a template to create queries to find out all instances of a particular class. Once a user selects a query type, the text area for typing in query is updated with the corresponding templates. After a query is entered, user may perform syntax checking of the query before submitting it. As the target users do not have the required expertise to identify the namespaces of a given ontology, we have hard-wired the namespaces for the military plan ontologies into the tool, which may cause inconvenience for use with other domains. We plan to change this, which is detailed in Section 5.

For the military plans case study, we have developed a set of 14 query templates, including queries to find the sub-task/super-task relationship with regard to a particular military task, queries to find all military tasks whose start and end time fall into a particular time frame, queries to find all military tasks that proceeds/follows a given task, and queries to find a military unit assigned to execute a given task, etc. This set of templates greatly eases the query and understanding of the ontology.

3.3. Ontology Transformation

The main purpose of the tool is to realize our approach of using SE techniques and tools such as model-checking and theorem proving to verify DAML+OIL/OWL/RDF ontologies. Thus, SESeW provides fully automated transformation from ontologies to Alloy, Z and PVS specifications.

The transformation from ontologies to Alloy and Z specifications is discussed in detail in [8]. Originally the transformation from DAML+OIL to Alloy was accomplished with an XSLT stylesheet. To allow greater flexibility and accuracy, the transformation program is re-written using Java language. The transformation is based on the semantics library for DAML+OIL built in Alloy and Z. It is straightforwardly extended to OWL by defining the Alloy and Z semantics for the OWL language. The Z semantics is contained in a section $owl2z$, on top of $toolkit$. 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 $OWL2ZRules$, of rules which describes the above definitions in more than one angle and is used to help Z/EVES perform reasoning tasks. This section has $owl2z$ as its parent. Ontologies are built layer on layer. Other domain specific ontologies are built in terms of the basic concepts and their corresponding Z models will have $OWL2ZRules$ or its descendent sections as parents. The Alloy semantics is contained in a module called owl. Similar to the Z case, the Alloy models will import module owl or its descendants.

The transformed Alloy or Z specification is presented in its own text area. The first lines of the transformed specification imports the Alloy or Z library for DAML+OIL or OWL constructs. The transformed specification is ready to be imported to AA or Z/EVEs for various reasoning tasks. The transformation from DAML+OIL/OWL/RDF to Z has been fine tuned to make the proof using Z/EVES. Since in Z/EVES, a name must be declared before use, the transformation program extracts all the declared classes, properties and individuals first, put them at the beginning of the generated Z specification. In subsequent passes, predicates about these names are then grouped and generated. As these predicates are used in proof process, *labels* are systematically added to all the predicates for easy referencing later on. The transformation to Alloy is similarly achieved.

In addition, SESeW also includes the fully automated transformation from ontology languages to PVS so as to verify both OWL and ORL (OWL Rules Language). In order to use PVS to verify and reason about ontologies with ORL axioms, it is necessary to define the PVS semantics for OWL and ORL. This semantic model forms the reasoning environment for verification using PVS theorem prover. The complete PVS semantics for OWL language primitives and the newly proposed ORL are available online[2]. To make the proving process of PVS more automated, a set of rewrite rules and theorems are defined. They aim to hide certain amount of underlying model from the verification and reasoning and to achieve abstraction and automation. Usually these rules relate several classes and properties by defining the effect of using them in a particular way. PVS is used for standard SW reasoning like inconsistency checking, subsumption reasoning, instantiation reasoning as well as checking ORL and beyond. For instance, OWL and ORL cannot deal with the concrete domains: it can only make assertions about linear (in)equalities of cardinalities of property instances over integer. PVS, on the other hand, can perform basic arithmetic operations and comparisons.

2 `http://nt-appn.comp.nus.edu.sg/fm/ORL2PVS/OWL2PVS.pvs.txt`

3.4. External Tools Connection

SESeW also integrates existing tools for developing and reasoning about ontologies so that a user may choose his/her favorite tool(s) to prepare the ontology before using our approach for reasoning about or verifying the finished ontology to obtain confidence. Shortcuts to the following tools are provided:

- Alloy Analyzer: To identify the source of an ontological error,
- Z/EVES: To type-check and theorem-prove the generated Z specification for properties that DAML+OIL/OWL cannot model,
- RACER: To act as a background reasoner for ontology editors OilEd and Protégé,
- OilEd: To develop, visualize ontologies and reason about them as well.

These tools are used in combination to reason about (transformed) ontologies. In the current version, we only invokes the external programs. Tighter connections between our SESeW with these tools are being pursued. For example, SESeW may interact with RACER through RACER's Java API so that a user can know whether the ontology is consistent immediately without the hassle of opening OilEd. Moreover, users may also make use of the rich query facilities RACER provides, which OilEd does not support.

AA is also developed in Java so that it is possible to develop programmatic ways of accessing functionalities of AA if the API is provided. In this way, SESeW becomes a more integrated formal environment. As AA can pin point the source of identified error, it will be more user-friendly if SESeW can directly command AA to bring up the identified erroneous source statements. We are currently involving people to explore the source code of AA for this purpose. Z/EVES is developed in Allegro Common Lisp and it presents a more complicated challenge for integration with SESeW.

4. Performance Evaluation

To evaluate the performance of SESeW, experimental performance monitors are included to find out the memory and computational time used for creation of ontologies.

Figure 5 shows how the increase in the number of ontology resources and relations affects the time and memory usage of the tool, assuming that in an ontology building process, each resource/relation costs same amount of time and consumes same amount of memory. Approximately, the time and memory usage increases linearly as the number of resources/relations increases. The average time needed for creating one resource/relation decreases slowly.

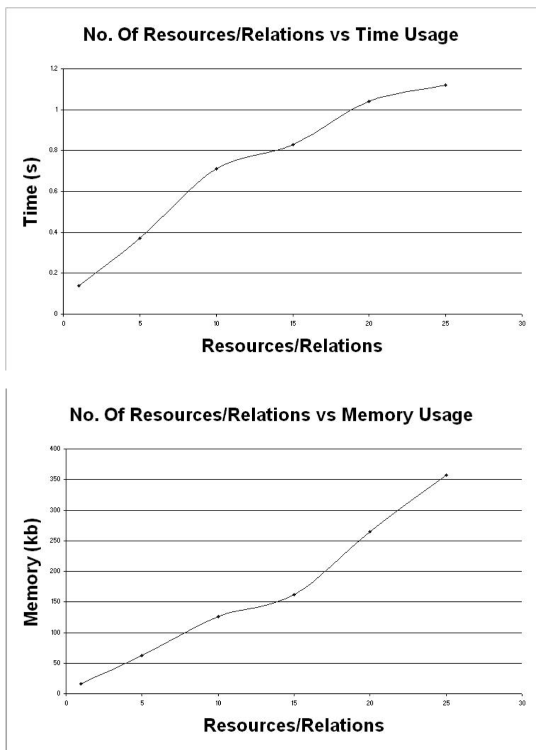


Figure 5. Performance of Creating Ontologies

The transformation from ontologies to formal specification language is also instant for DAML+OIL ontologies and takes a few seconds maximum for OWL ontologies (both linear time). The key point of using a combined approach is that our tool is able to identify (and locate) inconsistencies in a large ontology in a reasonable amount of time. Comparison on effectiveness and efficiency between our combined approach with others are available in [8]. However, because our approach involves interactive proving, e.g. Z/EVES, PVS or user-decided parameters which seriously affect both memory and computational time usage, e.g. AA, and moreover there are not yet benchmarks for evaluating performance on reasoning about ontologies, we skip the performance evaluation for reasoning about the ontologies.

5. Conclusion

As we have shown before, formal methods can be successfully applied to the Semantic Web domain to improve the quality of ontologies. To advocate this application, we developed an integrated tools environment so that different tools from both the SW and formal methods communities can be grouped together so that they can be used in combination more efficiently. In a nutshell, SESeW allows systematic creation as well as effective querying, transformation, verification and reasoning of DAML+OIL/OWL/RDF ontologies.

There are quite a number of ways that we can further improve SESeW. The very first is to offer tighter integration with external tools/programs, as discussed in Section 3.4. We may as well offer more configurable ways of performing queries. As SESeW has been developed with a military application in mind, the query part is currently dedicated to the military planning ontologies case study. For instance, the query templates are solely designed for military planning ontologies. However, our experience strongly suggests that query templates for general ontologies are useful as well. Hence, we are exploring general query patterns and templates as ways of speeding up the query search. Moreover, we plan to make query templates and namespaces pluggable so that more experienced users can design their own query templates, manage namespaces used in query, etc. Another important extension is to support ontology merging. Ontology merging is an important task in ontology development and integration as ontologies from different sources sometimes need to be merged or aligned to take advantage of the semantics of the ontologies. There is not much tool support for performing ontology merging and alignment currently. Lastly, because ontology languages are actively developing, it is necessary for SESeW to be extended to support the latest ontologies languages in the future. One of them at the moment is OWL-S [29], for Semantic Web Services. Besides reasoning about the static properties of an ontology, it is our belief that formal methods such as Z or CSP [15] can play an important role in ensuring the dynamic part of the service ontologies.

It is our goal in this paper to harness the strength of existing state-of-the-art formal methods and tools so that SW community can benefit. Although our tool is still in a prototype stage, we believe that it can serve as a good starting point of promoting a wider adoption of formal methods in novel application domains such as the Semantic Web.

Acknowledgement

This paper is partially supported by the Defense Innovative Research Project (DIRP) “Formal Design Methods and DAML”. The third author would like to thank Singapore Millennium Foundation (`http://www.smf-scholar.org/`) for financial support.

References

[1] RDQL - A Query Language for RDF. `http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/`, 2004.

[2] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a Reason-able Ontology Editor for the Semantic Web. In *Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence*, number 2174, pages 396–408, Vienna, September 2001. Springer-Verlag.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):35–43, 2001.

[4] A. Borgida. On the Relative Expressiveness of Description Logics and Predicate Logics. *Artificial Intelligence*, 82:353–367, 1996.

[5] O. Caprotti, J. H. Davenport, M. Dewar, and J. A. Padget. Mathematics on the (Semantic) NET. In *The Semantic Web: Research and Applications, First European Semantic Web Symposium, ESWS 2004*, pages 213–224, 2004.

[6] D. Brickley and R.V. Guha (editors). Resource Description Framework (RDF) Schema Specification 1.0. `http://www.w3.org/TR/rdf-schema/`, Feb. 2004.

[7] J. S. Dong, Y. Feng, and Y. F. Li. Reasoning Support for the OWL Rules Language. In *Proceedings of First International Colloquium on Theoretical Aspects of Computing (ICTAC'04)*, pages 265–279, Guiyang, China, Sept. 2004.

[8] J. S. Dong, C. H. Lee, Y. F. Li, and H. Wang. A Combined Approach to Checking Web Ontologies. In *Proceedings of 13th World Wide Web Conference (WWW'04)*, pages 714–722, New York, USA, May 2004.

[9] J. S. Dong, C. H. Lee, Y. F. Li, and H. Wang. Verifying DAML+OIL and beyond in Z/EVES. In *Proceedings of 26th International Conference on Software Engineering (ICSE'04)*, pages 201–210, May 2004.

[10] J. S. Dong, J. Sun, and H. Wang. 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.

[11] M. Fernandez, A. Gomez-Perez, and N. Juristo. METHONTOLOGY: from Ontological Art towards Ontological Engineering. In *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering*, pages 33–40, Stanford, USA, March 1997.

[12] J. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, and S. W. Tu. The evolution of protégé: An environment for knowledge-based systems development. Technical Report SMI-2002-0943, Stanford Medical Informatics, Stanford University, 2002.

[13] V. Haarslev and R. Möller. Practical Reasoning in Racer with a Concrete Domain for Linear Inequations. In I. Horrocks and S. Tessaris, editors, *Proceedings of the International Workshop on Description Logics (DL-2002)*, Toulouse, France, Apr. 2002. CEUR-WS.

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

[15] C. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice-Hall, 1985.

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

[17] I. Horrocks and P. F. Patel-Schneider. A Proposal for an OWL Rules Language. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 723–731, New York, USA, May 2004. ACM.

[18] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`, May 2004.

[19] HP Labs. Jena - A Semantic Web Framework for Java . `http://jena.sourceforge.net/`.

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

[21] D. Jackson, I. Schechter, and I. Shlyakhter. Alcoa: the Alloy Constraint Analyzer. In *The 22nd International Conference on Software Engineering (ICSE'00)*, pages 730–733, Limerick, Ireland, June 2000. ACM Press.

[22] M. K. Smith and C. Welty and D. L. McGuinness (editors). OWL Web Ontology Language Guide. `http://www.w3.org/TR/2004/REC-owl-guide-20040210/`, 2004.

[23] F. Manola and E. Miller. RDF Primer. `http://www.w3.org/TR/rdf-primer/`, Feb. 2004.

[24] M. Marchiori. The Mathematical Semantic Web. In *Mathematical Knowledge Management, Second International Conference, MKM 2003*, pages 216–224, 2003.

[25] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. `http://www.w3.org/TR/2003/PR-owl-features-20031215/`, Dec. 2003.

[26] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, pages 748–752, Saratoga, NY, 1992.

[27] A. Rezazadeh and M. Butler. Some Guidelines for Formal Development of Web-Based Applications in B-Method. In *ZB 2005: Formal Specification and Development in Z and B*, pages 472–492, 2005.

[28] M. Saaltink. The Z/EVES system. In J. P. Bowen, M. G. Hinchey, and D. Till, editors, *ZUM'97: Z Formal Specification Notation*, pages 72–85. Springer-Verlag, 1997.

[29] The OWL Services Coalition. OWL-S: Semantic Markup for Web Services. `http://www.daml.org/services/owl-s/`, 2004.

[30] 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, et. al., March, 2001.

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

[32] H. Zhu, B. Zhou, X. Mao, L. Shan, and D. A. Duce. Agent-Oriented Formal Specification of Web Services. In *Grid and Cooperative Computing - GCC 2004 Workshops*, pages 633–641, 2004.