# Chapter 6

# The Tools Environment: SESeW

The combined approach presented in Chapter 4 is an effective way of verifying correctness of the Semantic Web ontologies. However, it was also pointed out in Chapter 4 that there are a number rather involved steps in this approach. Moreover, there are some other functionalities, such as ontology querying, that the users may desire but not covered in the combined approach. An implementation of the ontology development methodology, the Methontology [29], is incorporated to facilitate systematic ontology creation.

For these reasons, we have developed a prototype of an integrated tools environment, the Software Engineering for the Semantic Web (SESeW), that facilitates the application of the combined approach and supports a number of other functionalities.

This chapter is devoted to an introduction of our integrated tools environment for developing and reasoning DAML+OIL and OWL ontologies. It is divided into the following parts. In Section 6.1, we present SESeW in brief. In subsequent Sections, we present the ontology creation process, querying, transformation and connection with external tools. Finally, Section 6.6 summarizes the chapter.

## 6.1 Overview of SESeW

Figure 6.1 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 [78] [1] specifications respectively. Transformed Z, Alloy and PVS specifications are displayed in the respective text areas.

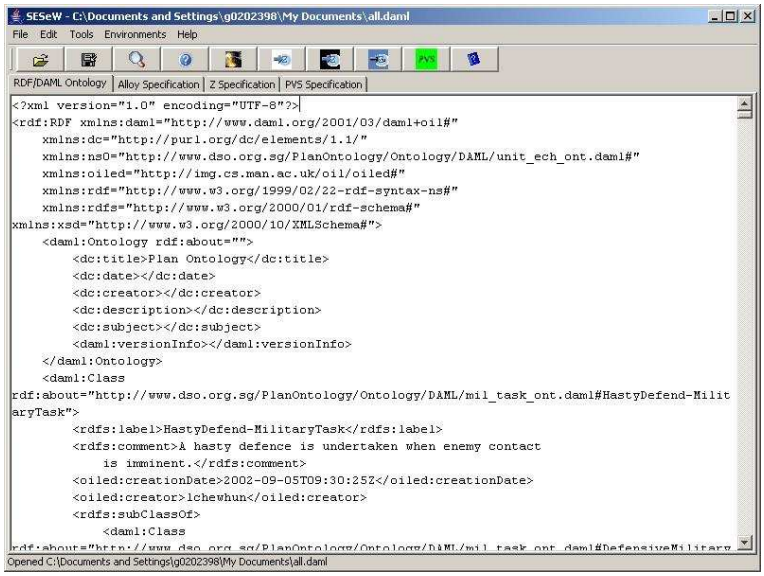


Figure 6.1: Main Window of SESeW

A user may load an existing ontology created using other editors like Protégé [30], 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.

[1] The PVS text area is for research work [21].

## 6.2 Ontology Creation

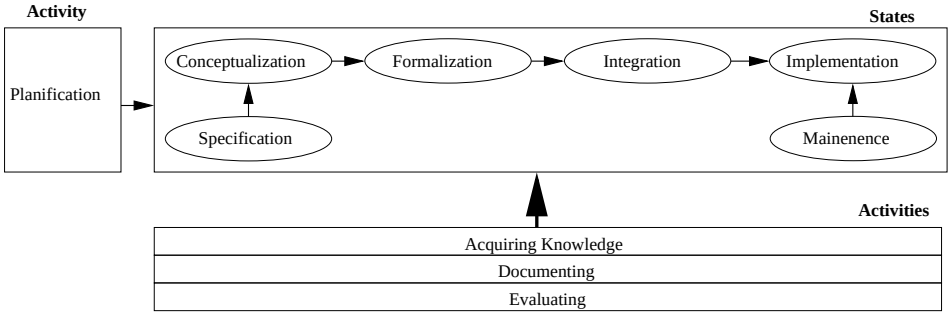


Figure 6.2: Flow of Ontology Creation

We implemented a systematic methodology for creating ontologies, namely Methontology [29]. 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 6.2, borrowed from [29]). The specification phase is to produce either an informal, semi-formal or formal ontology specification document in natural language, 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 in 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 6.3 shows the window for introducing new datatype properties. A datatype property can be a *FunctionalProperty*.

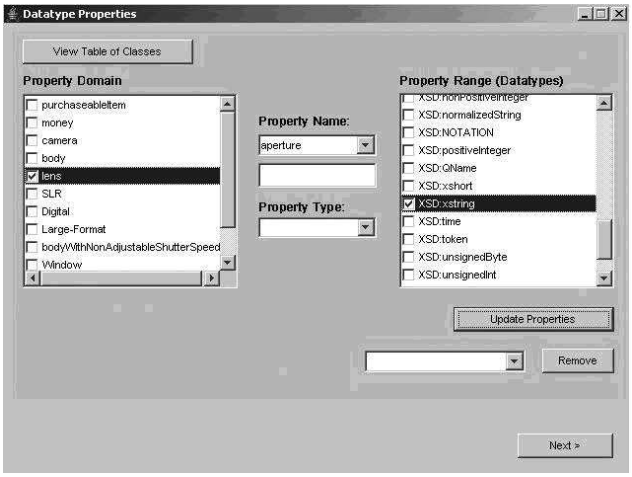


Figure 6.3: Creating Datatype Property

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*.

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 least one value of the property is a member of the class indicated by the *someValueFrom* clause.
- *cardinality*: which requires the specification of 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.

After the user has fully specified the ontology, it is automatically generated, making use of the Jena Framework [51] (bundled with SESeW), shown in its text area and saved into the file designated.

### 6.2.1 Performance Evaluation

To evaluate the performance of ontology generation in SESeW, experimental performance monitors are included to find out the memory and computational time used

for creation of ontologies.

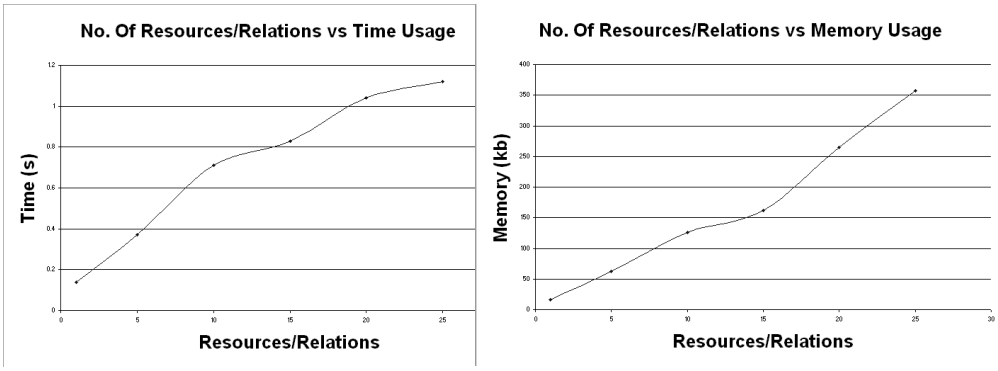


Figure 6.4: Performance of Ontology Creation

Figure 6.4 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.

## 6.3 Ontology Querying

A friendly user interface, shown in Figure 6.5, is provided for querying a given ontology. A user may input queries in an SQL-like language RDQL [86]. The query engine is a part of the ontology toolkit, the Jena Framework. 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 has the structure "`SELECT...WHERE...USING...`",

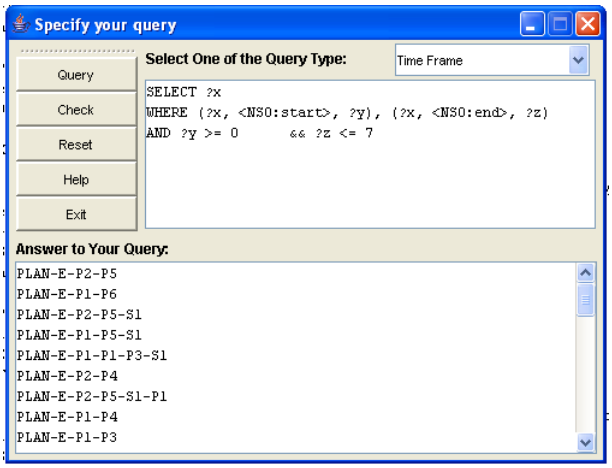


Figure 6.5: The Query Interface

where ontology entities of interest are specified after the keyword `SELECT` along with constraints after the keyword `WHERE` in the namespaces given after the keyword `USING`.

As SESeW 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 may not have the required expertise to identify the namespaces of a given ontology, the namespaces in an ontology are automatically recognized and extracted for user's convenience.

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 querying and understanding of the ontology.

## 6.4 Ontology Transformation

The main purpose of the SESeW is to realize our approach of using software engineering 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 DAML+OIL/OWL ontologies to Z specifications was discussed in detail in 3.4. Originally the transformation from DAML+OIL to Alloy was accomplished with an XSLT [109] stylesheet. To be integrated into the SESeW framework, the transformation program has been re-written using Java language. The transformation is based on the semantics library for DAML+OIL built in Alloy and Z. The semantical libraries are 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*. Similarly, the Alloy semantics is contained in a module *owl*.

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 Alloy Analyzer 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 more automated. In Z/EVES, a name must be declared

before it is used. Hence, the transformation program extracts all the names of 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.

In addition, SESeW also includes the fully automated transformation from ontology languages to PVS so as to verify both OWL and SWRL ontologies. In order to use PVS to verify and reason about ontologies with SWRL axioms, it is necessary to define the PVS semantics for OWL and SWRL. 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 SWRL 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 SWRL and beyond. For instance, OWL and SWRL 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]cf. http://nt-appn.comp.nus.edu.sg/fm/ORL2PVS/OWL2PVS.pvs.txt

## 6.5 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. The following tools are bundled with, or connected to SESeW with shortcuts:

- Alloy Analyzer: It is bundled with SESeW and can be invoked directly
- Z/EVES: A shortcut to Z/EVES previously installed in a machine is provided in SESeW to invoke it.
- RACER: Acting as a background reasoner, RACER is bundled with SESeW so that its reasoning functionality can be directly tapped whenever required. Moreover, RACER also acts as a background reasoner for ontology editors.
- OilEd: Being an ontology editor for DAML+OIL, OilEd is bundled with SESeW so that ontologies can be developed, visualized and reasoned about.

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

## 6.6 Chapter Summary

As we have shown in previous chapters, 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, the SESeW, so that different tools from both the SW and formal methods communities can be grouped together and used in combination more efficiently. In a nutshell, SESeW allows systematic creation as well as effective querying, transformation, verification and reasoning about DAML+OIL/OWL/RDF ontologies.

The SESeW includes functionalities such as ontology creation, querying, transformation, etc. It also links with a number of external tools to visualize and reason about ontologies.

By implementing a systematic approach of ontology creation, the Methontology, and supporting ontology querying and the combined approach of verifying ontology correctness, the SESeW supports a complete ontology life cycle.

So far, the chapters are only focused on transforming and verifying static Web resources. The dynamic aspect of the Web, the Web services, will take stage in the next chapter.