SWIDE: Semantic Web Integrated Development Environment

Islam Hany Harb[1], Abdurrahman A. Nasr, Salah Abdel-Magid, Hany Harb
1 Computers and Systems Engineering Department
Azhar University
Cairo, Egypt
1 islam.hany@hotmail.com

Abstract: - Ontology is a specification of conceptualization. This paper introduces an environment to develop semantic web applications. This environment integrates a lot of tools such as an editing capability, logic reasoner and semantic search engine. Design and implementation of a generalized ontology editor is presented in this paper through which the user may create, edit, validate, open, search (local and global), or visualize an ontology or an instance file. The user may edit an instance to be stored either in a RDF/XML file, OWL/XML, xml knowledge base or other formats. The user may present the ontology hierarchy and the knowledge base in a tabular form. The environment provides an interface through which the user may consult the knowledge base by SQL-like statements. It also allows the user to map ontology to another. It also introduces the virtualization concept providing a mechanism to categorize the ontology instances based on given ontology features. It also provides logic reasoner so we may check the truth of an instance against a specific knowledge base. A semantic search engine is also available either locally or globally.

Key-Words: - Ontology Editor, RDF, OWL, XML, Semantic Web, SPARQL, Jena

1 Introduction

The Semantic Web aims at representing the semantics of resources. It aims at providing a promising foundation for enriching resources with well defined meanings. The ontology defines the vocabularies that describe the domain and it also defines the relationships between these vocabularies. The ontology is required to unify the meaning of the vocabularies used in a domain. Different words can be used to describe the same meaning. The ontology can put the axioms that relate these terms and vocabularies with each other. This way makes the search more efficient and easier. It expands the space of words that the user can use and it can also be used to handle the abbreviations to be matched with the corresponding vocabularies.

A semantic web integrated development environment (SWIDE) has been designed and implemented in this paper. The user may stick with this environment to do all the semantic web related activities. The SWIDE makes it easier to work simultaneously with both classes and instances. Thus, a singular instance can be used on the level of a class definition, and a class can be stored as an instance.
The knowledge-based system is usually very expensive to build and maintain and its development is a team effort, including both developers and domain experts who may have less familiarity with computer software. The SWIDE is designed to guide developers and domain experts through the system development process of knowledge-based systems. The SWIDE is designed to allow developers to reuse domain ontologies and problem-solving methods, thereby shortening the time needed for development and maintenance. Several applications can use the same domain ontology to solve different problems, and the same problem-solving method can be used with different ontologies. This paper is organized as follows. Section 2 presents the planning of the SWIDE project, while section 3 describes the environment architecture.

2 Planning a SWIDE project

The SWIDE is an integrated software tool used by system developers and domain experts to develop knowledge-based system and its problem-solving and decision-making applications in a particular domain. The SWIDE is designed to support iterative development, where there are cycles of revision to the ontologies and other components of the knowledge-based system. Therefore developers should not expect to "complete" ontology development without considering other aspects of the process. For the development of a successful SWIDE project, we would recommend the following steps [3]:

1. Plan for the application and expected uses of the knowledge base. This usually means working with domain experts that have a set of problems that could be solved with knowledge-base technology.
2. Build an initial small ontology of classes and slots.
3. When we have built this ontology (and later when we have extended it or opened it from file), we can directly view forms for entering instance knowledge into the ontology.
4. We may use these forms for acquiring slot values of test instances. The user may judge the ontology and the filled-out instance forms. This inevitably leads to a set of revisions, both to the ontology and to the forms. Note that ontology modifications can be expensive, since some sorts of change could force rebuilding some or all of the knowledge base.
5. Customize the forms to a refined knowledge-acquisition tool. While constructing this customized version of the KB-sub tool, further design problems in the original ontology may become apparent.
6. Build a somewhat larger knowledge-base that can be tested with the application or problem-solving method.
7. Test the full application. This step can lead to further revisions to the ontology and the KB-sub tool.

Figure 1 below shows this typical pattern use of the SWIDE subsystems. The thick arrows indicate the forward progression through the process, while the thin arrows show places where revisions are usually necessary (either to the ontology or the knowledge-acquisition tool).

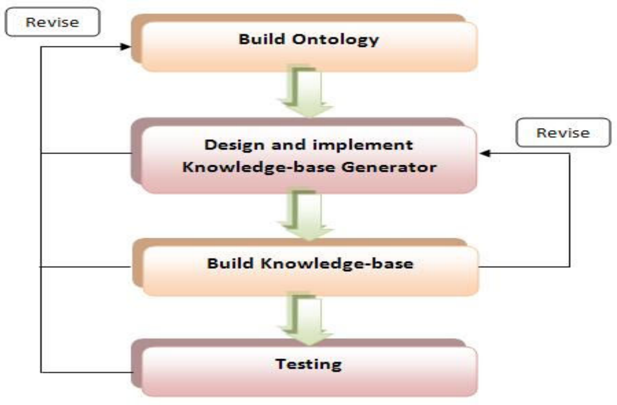


Fig.1: Typical pattern use of the SWIDE subsystems

At the heart of a successful SWIDE project is the design of the class and slot structure of the ontology. In particular, the model we use in building our ontology must balance the needs of the domain expert when building a knowledge base (at knowledge-acquisition time) against the requirements of our problem-solving method or application (at run-time). Hopefully, these are not too contradictory! Ontology developers should therefore both [3]:

- Model the domain with a set of problems and a problem-solving method in mind.
- Design the ontology so that it can be used to generate and customize an appropriate KB-tool for a specific set of users.

We build on our experience using SWIDE (Protege 2000), Ontolingua (Ontolingua 1997), and Chimaera (Chimaera 2000) as ontology-editing environments. In this paper, we use SWIDE as a developer tool for our ontology examples. The e-learner and sensor examples that we use throughout in this paper are considered two case studies.The e-learner is mainly based on an ontology and knowledge base presented in a paper describing SEE-ONT as e-learner ontology for Social and Educational Environment [1].

3 The SWIDE System Architecture

The SWIDE architecture is considered as a three tier architecture as shown in the figure 2. There are three main layers and a database that holds any ontology with its knowledge base (instances). The first layer from the top is the User Interface (UI). The UI is used to facilitate and enable all the SWIDE users from dealing with it. The SWIDE targets users who are experts in specific domains. Meanwhile these users may not be developers or may not even know any programming language. The UI will take care of these programming technical issues instead of the user. The second layer is the Business Logic (BL) which handles and services all the events encountered by the users through the UI. It interprets the events and the actions of the users specifying the appropriate API functions to call. The third layer is the Ontology Management Engine (OME) which is the most important layer where all the operations that are done on the ontology are settled in this layer.

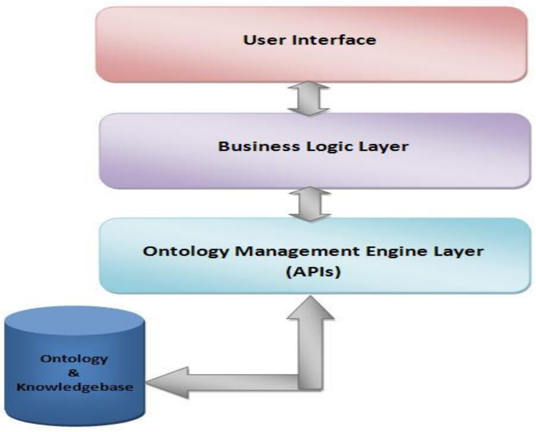


Fig.2: The SWIDE Architecture

3.1 The key components

The key components and the features that are provided by the SWIDE system are user interface component, ontology editor, semantic search engine, consistency checker, ontology export, Ontology Visualization, SPARQL, and Markup.

3.1.1 User Interface. It is the graphical forms that enable the non-developer users from dealing with the SWIDE system without going through programming issues.

3.1.2 Ontology Editor. It is the component that enables the domain experts from going through the life cycle of the Ontology development. Editing ontology is generalized and it includes the following features:

- Create a new ontology
- Open an existing ontology
- Define classes in the ontology.
- Arrange the classes in a taxonomic (sub class–super class) hierarchy.
- Define slots and describe the allowed values for these slots,
- Fill in the values for slots for instances.
- Display the ontology in a pane of the SWIDE either as XML DOM Tree view or as XML code text.

3.1.3 Semantic Search Engine. Sometimes, we do not want to create ontology from scratch. It is preferable to have a look and search for all the ontologies within the domain. So that we may find appropriate predefined ontology in the same domain and can be considered as suitable starting point. This feature enables the users from searching **locally** and **globally** for any ontology using instances or classes. Figure 3 shows the SWIDE system search engine interface that provides such feature. This interface appeared after we pressed on the “Search Ontologies” option. The “Text” field will hold the class or the instance that we want to search for its related ontology. There is “Online” check box on the right acting as a switch between local and global search.

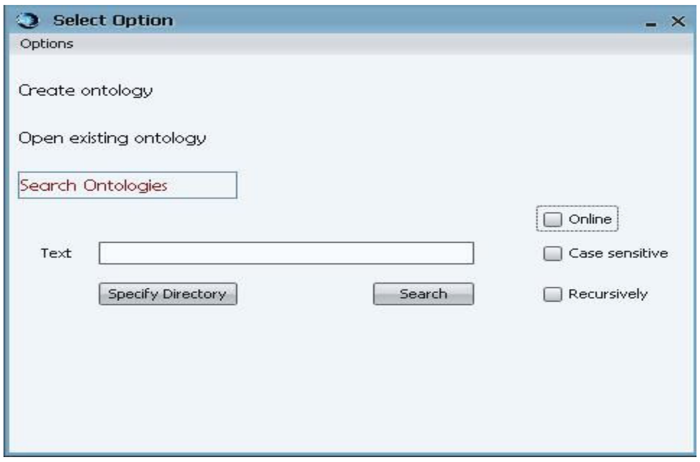


Fig.3: The local and global semantic search features

3.1.4 Consistency Checker. The SWIDE system enables the users to create ontology in a consistent methodology and so not an error prone process. Consistency check is mandatory first step before any other reasoning service, which verifies the input axioms do not contain contradictions and only contain satisfiable axioms. Meanwhile loading or importing ontology process may be campaigned with inconsistency problems. This component checks for the consistency of any loaded ontology. If ontology is not a consistent one, then it will be loaded with error messages showing all inconsistent states.

As an example of inconsistency, consider the following three statements

CivilServant disjointWith Contractor

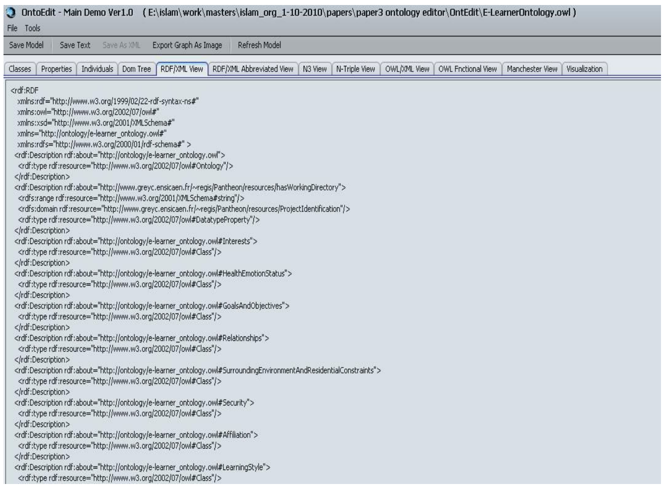
Person0853 type CivilServant, Contractor

CivilServantContractor subClassOf (CivilServant and Contractor)

Consistency check is very important in ontology building environments to ensure that no instances of unsatisfiable classes exist, since any consequence can be inferred from inconsistency. As an example of inconsistency, consider the following three statements

3.1.5 Ontology Export. This component is responsible for writing or exporting ontology into different formats. Figure 4 shows the available formats that are supported by the SWIDE where there is more than one tab and each tab is corresponding for a specific format to save and export the current ontology. The supported formats are RDF/XML, RDF/XML Abbreviated, N3, N-Triple, OWL/XML, OWL Functional and Manchester.

3.1.6 Ontology Visualization. The SWIDE provides the visualization feature for its users. The Visualization is a mechanism to represent the ontology as a graph. It describes all the classes, the instances and the relationships between them in a specific ontology. Figure 5 shows the visualization within the SWIDE system. This graph in the figure represents the high level taxonomy of all the concepts in **sensor** ontology as one of our case studies. Figure 6 shows another visualization graph (cluttered graph) for **sensor** ontology.



```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns="http://ontology/e-learner_ontology.owl#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
 <rdf:Description rdf:about="http://ontology/e-learner_ontology.owl">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Ontology"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://www.greyc.ensicaen.fr/~regis/Pantheon/resources/hasWorkingDirectory">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="http://www.greyc.ensicaen.fr/~regis/Pantheon/resources/ProjectIdentification"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://ontology/e-learner_ontology.owl#Interests">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://ontology/e-learner_ontology.owl#HealthEmotionStatus">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://ontology/e-learner_ontology.owl#GoalsAndObjectives">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://ontology/e-learner_ontology.owl#Relationships">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://ontology/e-learner_ontology.owl#SurroundingEnvironmentAndResidentialConstraints">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://ontology/e-learner_ontology.owl#Security">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://ontology/e-learner_ontology.owl#Affiliation">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://ontology/e-learner_ontology.owl#LearningStyle">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
```

Fig.4: Taps shows the different formats that are supported by SWIDE

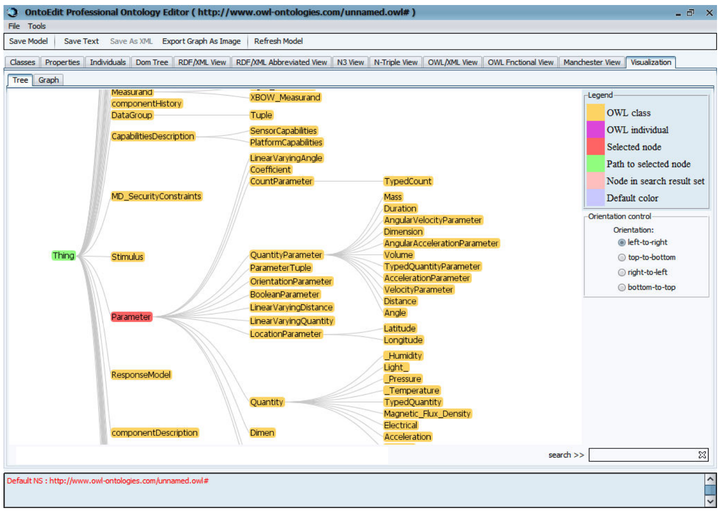


Fig.5: Ontology tree visualization for sensor ontology using the SWIDE

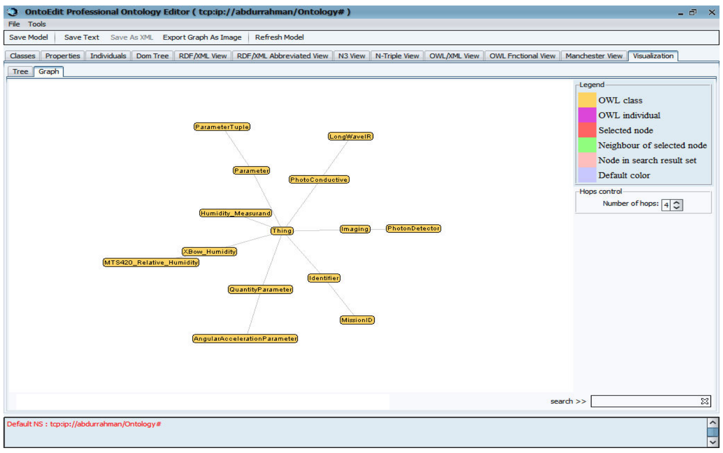


Figure 6: Cluttered graph for sensor ontology using the SWIDE

3.1.7 Consult the knowledge base by SPARQL. SPARQL is a W3C Candidate Recommendation towards a standard query language for the Semantic Web. Its focus is on querying RDF graphs at the triple level. SPARQL can be used to query an RDF Schema or OWL model to filter out criteria with specific characteristics. The SWIDE provides this feature to query and retrieve any information from ontology written with any of the triples format. Figure 7 shows the steps to be followed to run any query on the ontology (Tools -> SPARQL). After selecting this feature, a new form that enables the user to construct any query will appear. This Query form is shown in figure 8.

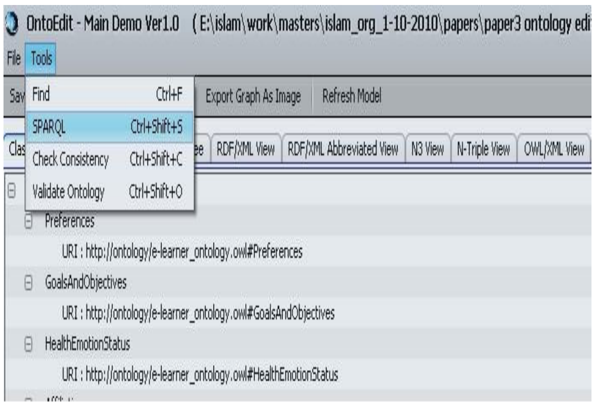
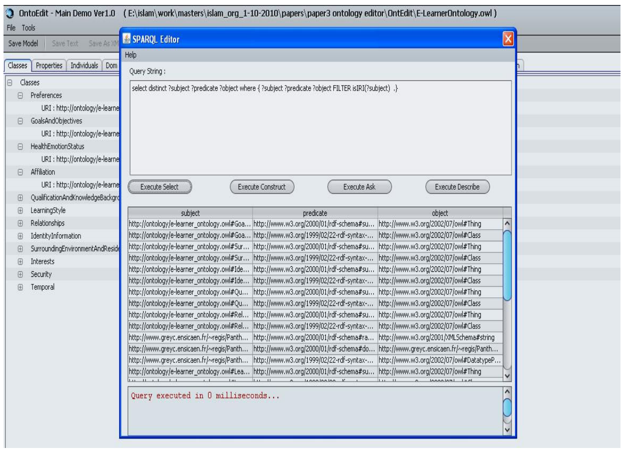

OntoEdit - Main Demo Ver1.0 (E:\islam\work\masters\islam_org_1-10-2010\papers\paper3 ontology edi
File Tools
Find Ctrl+F
SPARQL Ctrl+Shift+S
Check Consistency Ctrl+Shift+C
Validate Ontology Ctrl+Shift+O
Export Graph As Image
Refresh Model
RDF/XML View
RDF/XML Abbreviated View
N3 View
N-Triple View
OWL/XML View
Preferences
URI : http://ontology/e-learner_ontology.owl#Preferences
GoalsAndObjectives
URI : http://ontology/e-learner_ontology.owl#GoalsAndObjectives
HealthEmotionStatus
URI : http://ontology/e-learner_ontology.owl#HealthEmotionStatus

Fig.7: Steps to run the SPARQL feature


OntoEdit - Main Demo Ver1.0 (E:\islam\work\masters\islam_org_1-10-2010\papers\paper3 ontology editor\OntEdit\E-LearnerOntology.owl)
SPARQL Editor
Help
Query String :
Execute Select
Execute Construct
Execute Ask
Execute Describe

```
select distinct ?subject ?predicate ?object where { ?subject ?predicate ?object FILTER isIRI(?subject) .}
```

subject	predicate	object
http://ontology/e-learner_ontology.owl#Goa...	http://www.w3.org/2000/01/rdf-schema#su...	http://www.w3.org/2002/07/owl#Thing
http://ontology/e-learner_ontology.owl#Goa...	http://www.w3.org/1999/02/22-rdf-syntax-...	http://www.w3.org/2002/07/owl#Class
http://ontology/e-learner_ontology.owl#Sur...	http://www.w3.org/2000/01/rdf-schema#su...	http://www.w3.org/2002/07/owl#Thing
http://ontology/e-learner_ontology.owl#Sur...	http://www.w3.org/1999/02/22-rdf-syntax-...	http://www.w3.org/2002/07/owl#Class
http://ontology/e-learner_ontology.owl#Ide...	http://www.w3.org/2000/01/rdf-schema#su...	http://www.w3.org/2002/07/owl#Thing
http://ontology/e-learner_ontology.owl#Ide...	http://www.w3.org/1999/02/22-rdf-syntax-...	http://www.w3.org/2002/07/owl#Class
http://ontology/e-learner_ontology.owl#Qu...	http://www.w3.org/2000/01/rdf-schema#su...	http://www.w3.org/2002/07/owl#Thing
http://ontology/e-learner_ontology.owl#Qu...	http://www.w3.org/1999/02/22-rdf-syntax-...	http://www.w3.org/2002/07/owl#Class
http://ontology/e-learner_ontology.owl#Rel...	http://www.w3.org/2000/01/rdf-schema#su...	http://www.w3.org/2002/07/owl#Thing
http://ontology/e-learner_ontology.owl#Rel...	http://www.w3.org/1999/02/22-rdf-syntax-...	http://www.w3.org/2002/07/owl#Class
http://www.greyc.ensicaen.fr/~regis/Panth...	http://www.w3.org/2000/01/rdf-schema#ra...	http://www.w3.org/2001/XMLSchema#string
http://www.greyc.ensicaen.fr/~regis/Panth...	http://www.w3.org/2000/01/rdf-schema#do...	http://www.greyc.ensicaen.fr/~regis/Panth...
http://www.greyc.ensicaen.fr/~regis/Panth...	http://www.w3.org/1999/02/22-rdf-syntax-...	http://www.w3.org/2002/07/owl#DatatypeP...
http://ontology/e-learner_ontology.owl#Lea...	http://www.w3.org/2000/01/rdf-schema#su...	http://www.w3.org/2002/07/owl#Thing

```
Query executed in 0 milliseconds...
```

Fig.8: The Query Form that enables the user to construct any of SPARQL statements

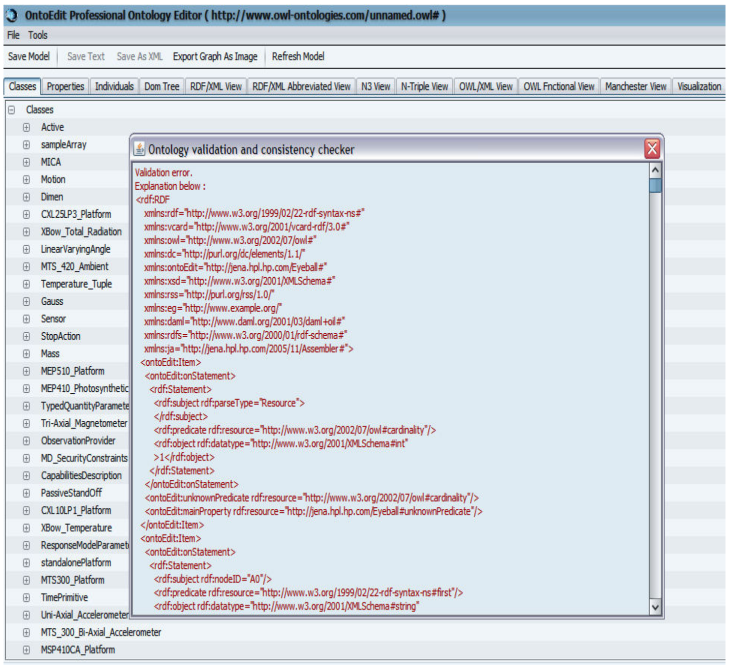
3.1.8 Ontology Validator

Ontology validation is the process of checking RDF and OWL models for various common problems. These problems often result in technically correct but implausible RDF. SWIDE validator checks against user-provided schema files and makes various closed-world assumptions.

SWIDE ontology validator can check for:

- unknown [with respect to the schemas] properties and classes
- bad prefix namespaces
- ill-formed URIs, with user-specifiable constraints
- ill-formed language tags on literals
- data typed literals with illegal lexical forms
- unexpected local names in schema namespaces
- un typed resources and literals
- individuals having consistent types, assuming complete typing
- likely cardinality violations
- broken RDF list structures
- suspected broken use of the typed list idiom
- obviously broken OWL restrictions
- user-specified constraints written in SPARQL

SWIDE ontology validator works by inspecting your model against a set of schemas. The default set of schemas includes RDF, RDFS, the XSD data types, and any models your model imports. Figure 7 shows the ontology validation menu (Tools -> Validate Ontology). The validation screen is shown in Figure 9



```
Validation error.
Explanation below :
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:ontoEdit="http://jena.hpl.hp.com/Eyeball#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rss="http://purl.org/rss/1.0/"
  xmlns:eg="http://www.example.org/"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:ja="http://jena.hpl.hp.com/2005/11/Assembler#">
 <ontoEdit:Item>
  <ontoEdit:onStatement>
   <rdf:Statement>
    <rdf:subject rdf:parseType="Resource">
    </rdf:subject>
    <rdf:predicate rdf:resource="http://www.w3.org/2002/07/owl#cardinality"/>
    <rdf:object rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</rdf:object>
   </rdf:Statement>
  </ontoEdit:onStatement>
  <ontoEdit:unknownPredicate rdf:resource="http://www.w3.org/2002/07/owl#cardinality"/>
  <ontoEdit:mainProperty rdf:resource="http://jena.hpl.hp.com/Eyeball#unknownPredicate"/>
 </ontoEdit:Item>
 <ontoEdit:Item>
  <ontoEdit:onStatement>
   <rdf:Statement>
    <rdf:subject rdf:nodeID="A0"/>
    <rdf:predicate rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#first"/>
    <rdf:object rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

Figure 9: The ontology validation screen which list validation errors and explanation

3.1.9 Mark up

The Mark up is the process of adding annotations and semantics to the resources of the current web to make them machine readable. Ontologies are the best solution that can be used to annotate the web resources. The vocabularies that are defined within ontology allow a shared understanding of the context for specific domain. These vocabularies may be used to mark up and describe the web pages and the other web resources. To make Web pages machine-readable, they must be connected to the appropriate ontologies corresponding to the domain of the page need to be annotated. Semantic mark up can be achieved by using RDF document that contains some RDF statements describing and corresponding to some interesting entities and contents of web resource. Most common web resource is a web page. These entities and contents of the web resource represent other entities in the real world. Ontology represents these real world entities by classes and instances of these classes. The connection between the web resources and that ontology occurs using this RDF document. The RDF statements connect the entities in the web resources to their corresponding and the appropriate classes and instances within the suitable ontology. It is important to realize that a markup document is mainly for the agents to read and it is not for human. This attached RDF document makes the agent to be smarter and understand the content's semantics of the web page. The next subsection provides the procedures and steps to mark up any web page and discusses the Mark Up feature in the SWIDE system.

Semantic Mark up Steps

Marking up a web page is not a standard process. There are different methodologies that can be used to add annotations for any web resources. But, there are several steps we need to follow when semantically marking up a Web page [4] as it follows.

Step 1: Decide which ontology or ontologies to use for semantic markup. In this step, it is very important to determine the most suitable ontology to use for the annotations. The Ontology should be the nearest and much related to the domain of web resource's field and entities. This involves searching for all the existing ontologies within this domain, then reading and understanding them carefully. After that we may decide whether it fits our need and whether it is enough to express the semantics of all the interesting entities in the web resource. It is possible that we may have to come up with our own ontology, in which case, we need to remember the rule of always trying to reuse existing ontologies, or simply constructing our new ontology by extending an existing ontology [4].

Step 2: Markup the Web page. In this step, we should decide which entities and parts in the web page will be annotated. Then loop through all these entities and search for the corresponding descriptions and concepts within our ontology. These appropriate concepts from the ontology should represent the semantics that we would like the agent to understand from the annotation. The RDF document will contain all the important information about the interesting parts of our web page. Be sure that this RDF document does not have any syntax errors. If it has any syntax errors then the agent will not be able to read and understand its contents. Sometimes, it is useful to use a RDF validator to make sure that this RDF document is correct. W3C provides such validator in its official website. Also SWIDE can be used as a validator for this RDF document. If the SWIDE fails to open a document, then there are syntax errors.

Step 3: Let the world know that our page has a markup document. Once we have finished successfully constructed the RDF document, we have to link it with the web page and make it accessible by the whole world. We need to indicate explicitly to the agent that our web page has a markup RDF document. At the time of this writing, there is no standard way of accomplishing this. A popular method is to add a link in the HTML header of the Web page [4].

SWIDE's Mark Up

SWIDE provides mark up feature for the web contents and resources. The mark up is simply done by creating a Mark Up file and link it with the web's resource that is needed to be annotated. This Mark up file includes RDF statements that describes partial or whole of the web's resource. These RDF statements use concepts and properties defined within one or more ontologies. Now and after adding this mark up file, this web's resource will be understood and its data can be processed by agent or machine. The problem is that who will do such mark up and annotation for them and what will be the motivation. A lot of work is required just to markup even a simple page. Also, the owner of the page should know even basic information about ontologies and OWL. So it is not that easy task for the owners to markup their pages or other pages if possible. This is quite a dilemma [4]: without a killer Semantic Web application to demonstrate some significant benefit to the world, there will be no motivation for the page owners to markup their pages; on the other hand, without the link between the current Web and the machine-readable semantics, the killer application simply cannot be created. There is one solution for such dilemma is to automate this markup process. There is no till now a standard known mechanism to automate the markup process. The difficulties to automate the markup process are:

1. The hypertext content of the web page. The Internet contains highly heterogeneous text types that are mainly made up of natural languages.

2. The decision whether to markup all the web page or only part of it. Sometimes it will be difficult or even not possible for the agent to decide what to markup within web page.

There is another problem. There are a lot of web resources and contents that already exist without any mark up. If we assumed that we have the volunteers that will markup these documents and web resources, we still have where to save these RDF markup documents. It may happen that we don't have the write access to the web server where the web page exists. One solution is to establish a centralized server. We can upload any markup document to this centralized server. As mentioned above in the step3 of markup steps in the same section, we need to link this markup with its web page.

4 SWIDE Packages and APIs

This section introduces all the API's packages that we use in the SWIDE system. All the used Packages in the SWIDE are java APIs. We have imported different packages to support all the features and components of the SWIDE. For example, searching online through the World Wide Web feature, we have used the "Watson" package APIs [5,6].

```
    // packages imported from the WATSON

import
uk.ac.open.kmi.watson.clientapi.OntologySearch;

import
uk.ac.open.kmi.watson.clientapi.OntologySearchS
erviceLocator;

import
uk.ac.open.kmi.watson.clientapi.WatsonService;

        // excerpt from the actual code…

public OntologySearchEngine(){



OntologySearchServiceLocator locator =

new OntologySearchServiceLocator();

    try{

            os = locator.getUrnOntologySearch();

    }

    catch (Exception e) {

            log.error("Error : ",e)

    }

}
```

This API returns back the list of all the ontologies in any domain that is related to our search key. The used search key most probably represents an instance or class within the ontology. The SWIDE also provides other APIs and functionalities within this Watson package but they are deprecated. Some of the deprecated APIs in Watson package:

- List all the classes within any retrieved Ontology.
- List all the instances within any retrieved ontology.
- List all the properties within any retrieved ontology
- List all the languages used within any retrieved ontology.
- Determine the number of reviews for any of the retrieved ontology.

Jena is a Java API for semantic web applications [7]. The API has been defined in terms of interfaces so that application code can work with different implementations without change. Packages in Jena are used within our system (SWIDE) so that application developers can create/Edit ontologies in RDF/OWL formats. This package contains interfaces for representing models, resources, properties, literals, statements and all the other key concepts of RDF, and a ModelFactory for creating models [7, 8].

OWL API is a Java API and reference implementation for creating, manipulating and serializing OWL Ontologies [8]. SWIDE uses OWL API for representing ontology with diverse formats

```
import
com.hp.hpl.jena.datatypes.xsd.XSDDatatype;

import
com.hp.hpl.jena.ontology.AnnotationProperty;

import com.hp.hpl.jena.ontology.DatatypeProperty;

import com.hp.hpl.jena.ontology.Individual;

import com.hp.hpl.jena.ontology.ObjectProperty;

import com.hp.hpl.jena.ontology.OntClass;

import com.hp.hpl.jena.ontology.OntModel;

import com.hp.hpl.jena.ontology.OntProperty;

import com.hp.hpl.jena.ontology.OntResource;

import com.hp.hpl.jena.ontology.Ontology;

import com.hp.hpl.jena.rdf.model.Resource;

import com.hp.hpl.jena.rdf.model.Statement;

import com.hp.hpl.jena.rdf.model.StmtIterator;

import
com.hp.hpl.jena.util.iterator.ExtendedIterator;

import com.hp.hpl.jena.vocabulary.DC;

import com.hp.hpl.jena.vocabulary.OWL;

import com.hp.hpl.jena.vocabulary.OWL2;

import com.hp.hpl.jena.vocabulary.RDFS;
```

Packages/APIs Used for SPQRQL

ARQ is a query engine for **Jena** that supports the SPARQL RDF Query language [9]. SPARQL is the query language developed by the W3C RDF Data Access Working Group.

```
        // packages imported from ARQ API

import com.hp.hpl.jena.query.Query;

import com.hp.hpl.jena.query.QueryExecution;

import
com.hp.hpl.jena.query.QueryExecutionFactory;

import com.hp.hpl.jena.query.QueryFactory;

import com.hp.hpl.jena.query.QuerySolution;

import com.hp.hpl.jena.query.ResultSet;

          // excerpt from actual code

public List<QuerySolution>

 sparqlSelect(String queryString, Model model,
JTextArea logger) {

String queryString = "select distinct ?subject
?predicate ?object where

 {  ?subject  ?predicate  ?object  FILTER
isIRI(?subject)  .}";

Query query = QueryFactory.create(queryString);

QueryExecution            qe            =
QueryExecutionFactory.create(query, model);

ResultSet resultSet = qe.execSelect();

List<QuerySolution>        solutions        =
ResultSetFormatter.toList(resultSet);

qe.close();

 return solutions;



   }
```

Packages/APIs Used for Visualization

1. OWL2Prefuse is a Java package which creates Prefuse graphs and trees from OWL files. It takes care of converting the OWL data structure to the Prefuse data structure [10]. This makes it is easy for developers, to use the Prefuse graphs and trees into their Semantic Web applications.

```
// packages imported from OWL Prefuse

import prefuse.data.Graph;

import prefuse.data.Tree;

// excerpt from actual code

public void createGraphPanel() {

        OWLGraphConverter graphConverter =
new OWLGraphConverter(owlModel, false);

        Graph graph = graphConverter.getGraph();

        GraphDisplay graphDisp = new
GraphDisplay(graph, true);

        GraphPanel graphPanel = new
GraphPanel(graphDisp, true, true);

        graphPanel.setVisible(true);

  }
```

2. Packages/APIs Used for Mark Up
SMORE APIs will be imported in our SWIDE. SMORE is one of the projects developed by the researchers and developers in the University of Maryland at College Park. SWIDE through the SMORE APIs enables the users from creating their own mark up files easily by using the provided GUI.

5 Related Works
Swoop is based on the Model-View-Controller (MVC) paradigm [14]. The SwoopModel component stores all ontology-centric information pertaining to the Swoop Workspace (currently loaded ontologies, change-logs, checkpoints) and defines key parameters used by the Swoop UI objects (such as selected OWL entity, view settings for imports, QNames etc). Additionally, a SwoopModelListener class is used to reflect changes in the UI based on changes in the SwoopModel (using a suitably defined event-notification scheme). Control is handled through a plugin based system, which loads new Renderers and Reasoners dynamically. The obvious advantage of a plugin framework is to ensure modularity of the code, and encourage external developers to contribute to the Swoop project easily. Finally, we note that the entire Swoop code is written in Java, maintained in a subversion repository and makes use of numerous third party libraries, the most prominent being the WonderWeb OWL API [11] for parsing OWL ontologies.

6 Conclusion
The Design and implementation of a semantic web integrated development environment (SWIDE) was presented in this paper. The SWIDE user may create ontology or use an existing one. The user may edit an instance to be stored either in an xml file or in xml knowledge base. The SWIDE may present the ontology hierarchy and may present the knowledge base in a tabular form. The user may consult the knowledge base by SQL-like statement. It also introduces the virtualization concept providing a mechanism to categorize the ontology instances based on given ontology features.

References:
[1] Islam H. Harb, S. Abdel-Mageid, H. Farahat, Ahmed Abdel_Nabi, *SEE ONT: E-Learner Otology For Social and Educational Environment*, Al-Azhar Engineering Eleventh International Conference, ISSN 1110-6409, Volume 5, No.8, December 21 - 23, 2010
[2] Islam H. Harb, *Semantic Web Technologies*, National Workshop on Computational Intelligence and the Web, Egypt IEEE Computational Intelligence Chapter and Al-Azhar University Student Chapter, 4th December 2010
[3] http://protegewiki.stanford.edu/wiki/PrF_UG_intro_all
[4] Liyang Yu, Introduction to the Semantic Web and Semantic Web Services, Chapman and Hall/CRC 2007, Pages 173–185, Print ISBN: 978-1-58488-933-5, eBook ISBN: 978-1-58488-934-2, DOI: 10.1201/9781584889342.ch9
[5] http://www.w3.org/2001/sw/wiki/Watson
[6] http://watson.kmi.open.ac.uk/WS_and_API.html
[7] http://jena.sourceforge.net/
[8] http://owlapi.sourceforge.net/
[9] http://jena.sourceforge.net/ARQ/
[10] http://owl2prefuse.sourceforge.net/
[11] Bechhofer, S., Lord, P., Volz, R. *Cooking the Semantic Web with the OWL API*. Proceedings of the International Semantic Web Conference (2003)

[12] Cuenca-Grau, B., Parsia, B., Sirin, E.: Working with Multiple Ontologies on the Semantic Web. Proceedings of the Third International Semantic Web Conference (ISWC) (2004)
[13] Kahan, J., Koivunen, M.R., Prud'Hommeaux, E., Swick, R.: Annotea: An Open RDF Infrastructure for Shared Web Annotations. Proc. of the WWW10 International Conference (2001)
[14] Aditya Kalyanpur , Bijan Parsia , Evren Sirin , Bernardo Cuenca Grau , James Hendler, *Swoop: Design and Architecture of a Web Ontology Browser (/Editor),* Journal of Web Semantics, 2005, Scholarly Paper for Master's Degree in Computer Science with Non-Thesis Option, Fall 2004