INFRAWEBS AXIOM EDITOR – A GRAPHICAL ONTOLOGY-DRIVEN TOOL FOR CREATING COMPLEX LOGICAL EXPRESSIONS

Gennady Agre, Petar Kormushev, Ivan Dilov

Abstract: The current INFRAWEBS European research project aims at developing ICT framework enabling software and service providers to generate and establish open and extensible development platforms for Web Service applications. One of the concrete project objectives is developing a full-life-cycle software toolset for creating and maintaining Semantic Web Services (SWSs) supporting specific applications based on Web Service Modelling Ontology (WSMO) framework. According to WSMO, functional and behavioural descriptions of a SWS may be represented by means of complex logical expressions (axioms). The paper describes a specialized user-friendly tool for constructing and editing such axioms – INFRAWEBS Axiom Editor. After discussing the main design principles of the Editor, its functional architecture is briefly presented. The tool is implemented in Eclipse Graphical Environment Framework and Eclipse Rich Client Platform.

Keywords: Web services, Semantic Web Services, Web Service Modelling Ontology framework.

ACM Classification Keywords: H.5.2 User Interfaces: Graphical user interfaces (GUI)

Introduction

Current Web service technologies describe the syntactical aspects of a Web service providing only a set of rigid services that cannot be adapted to a changing environment without human intervention. Realization of the full potential of the Web services requires further technological advances in the areas of service interoperation, service discovery, service composition and orchestration. A possible solution to these problems is likely to be provided by application of Semantic Web technologies.

Semantic Web Services (SWSs) are self-contained, self-describing, semantically marked-up software resources that can be published, discovered, composed and executed across the Web in a task driven semi-automatic way. There are two major initiatives aiming at developing world-wide standard for the semantic description of Web services – the American OWL-S [OWL-S 2004] and the European WSMO [Roman et al. 2005]. The INFRAWEBS European research project is based on the WSMO framework for service modelling and proposes a next, more technology-oriented step in the process of semantic Web service development [Nern et al. 2004]. One of the concrete project objectives is developing a full-life-cycle software toolset for creating and maintaining SWSs supporting specific applications based on Web Service Modelling Ontology (WSMO) framework.

A main part of WSMO-based SWS is service capability – a declarative description of Web service functionality. A formal syntax and semantics for such a description is provided by Web Service Modelling Language (WSML), which is based on different logical formalisms, namely, Description Logics, First-Order Logic and Logic Programming [de Bruijn et al. 2005]. The conceptual syntax for WSML has a frame-like style. The information about a class and its attributes, a relation and its parameters and an instance and its attribute values is specified in one large syntactic construct, instead of being divided into a number of atomic chunks. It is possible to spread the information about a particular class, relation, instance or axiom over several construct. WSML allows using of variables that may occur in place of concepts, attributes, instances, relation arguments or attribute values. A variable may not, however, replace a WSML keyword. Furthermore, variables may only be used inside logical expressions. A WSML description of a Web service capability is represented as a set of complex logical expressions called axioms. Machines can easily handle these axioms. However, it is very difficult for humans to create and comprehend complex logical expressions. Therefore, the construction of axioms needs to be supported by some easy-to-use graphical tools. It should allow a non-specialist to create highly complex axioms in WSML language through simple graphical interaction.

This paper describes a specialized tool called INFRAWEBS Axiom Editor, which is aimed at constructing and editing WSMO-based SWS capabilities. The structure of the paper is as follows – the next section discusses the basic design principle of the Editor. Then the models for representing and constructing the axioms are presented. Next two sections are devoted for describing the main functionality of the Editor and its graphical user interface. In conclusion some implementation details and future trends are discussed.

Basic Design Principle of INFRAWEBS Capability Editor

INFRAWEBS Axiom Editor is a specialized user-friendly tool for constructing and editing complex WSML logical expressions based on available set of WSML ontologies. It is a core part of a more complex tool – INFRAWENS Semantic Service Designer, which is aimed at converting existing Web services to WSMO-based semantic Web Services [Agre et al. 2005].

The main design principles of the Axiom Editor are:

1. *Specialization:* the tool is intended to be used mainly for constructing logical expressions representing capabilities of WSMO-based semantic Web services rather than axioms in ontologies. Our analysis has shown that the logical structure of such expressions is rather simple and in most cases does not require using of such complex WSML logical operators (connectives) as **Implies**, **ImpliedBy** or **Equivalent.**
2. *User-friendliness*: it is assumed that the users of our tool will be semantic Web service providers as well as customers of such services. In both cases the users will not be specialists in first-order logic, so using of some (even rather advanced) text editor for constructing logical expressions seems for us an inappropriate solution. That is why we propose a graphical way for constricting and editing the axioms abstracting away as much as possible from a concrete syntax of logical language used for implementing them.
3. *Intensive use of ontologies*: it is well known that the core concept of the Semantic Web is ontologies – "formal, explicit specification of a shared conceptualization" [Gruber 1993]. In our opinion, creating such formal and consensual specifications for different application domains requires very intensive cooperation of highly qualified domain knowledge engineers and logicians. Both categories of the users do not belong to the range of potential customers of our tool; for such users are more appropriate such general ontology editors like Protégé2000 [Protégé 2005] or Ontology Management Suit which is currently under development in the frame of WSMO project. So we assume that our customer will be mainly *a user* of already created ontologies rather then creator of new ontologies. However, we foresee that in some cases the service providers need to be able to create some *specialized versions* of (general) existing ontologies containing specific instances or subconcepts of general ontology concepts. Means for creating such (restricted) ontologies are going to be included in our Editor.
4. *Semantic consistency:* our analysis has shown that the main difficulties of the process of constructing complex logical expressions are associated with use of correct names of concepts, attributes, relations and parameters as well as their types rather than with expressing logic itself. That is why the process of constructing logical expression in INFRAWEBS Axiom Editor is *ontology-driven*, which means that in each step of this process the user may select only such elements of existing ontologies that are consistent with already constructed part of the axiom. From this point of view the created axiom is always semantically consistent with ontologies used for it construction.

Representation of Axioms

According to the formulated above requirements the Axiom Editor should allow automatic generating correct WSML logical expressions from some graphical representation (model) of such expressions. As a graphical model of WSML axiom we have selected a direct acyclic graph (DAG). Such a graph can contain four kinds of nodes:

- A single node called *Root*, which may have only outgoing arcs. This node corresponds to WSML statement **defineBy**. Graphically the root node is represented as a circle named "Start".
- Intermediate nodes called *variables*. Such nodes have one or more incoming arcs and can have several outgoing arcs. Each variable has a unique name and a frame-like structure consisting of slots represented by pairs attribute – attribute value (WSML variable). Such a variable corresponds to a notion of compound molecule in WSML [de Bruijn 2005] consisting of an a-molecule of type Var_i **memberOf** Γ and conjunction of

b-molecules of type Var_i [p_1 hasValue Var_{j1}] and Var_i [p_k hasValue Var_{kl}] respectively, where $Var_i, Var_{j1}, Var_{kl}$ are WSML variables and Γ is a concept from a given WSML ontology. Graphically each variable is represented as a rectangle with a header containing variable name and type (i.e. the name of concept, which has been used for crating the variable), and a row of named slots.

- Intermediate nodes called *relations*. Such a node corresponds to a WSML statment $r(Par_1, \ldots, Par_n)$, where r is a relation from a given ontology, and $Par_1, \ldots, Par_n$ are WSML variables – relation parameters. Graphically each relation node is represented as a rectangle with a header containing relation name and a row of relation parameters.
- Intermediate nodes called *operators* that correspond to WSML logical operators *AND*, *OR* and *NOT*. Each node can have only one incoming arcs and can have one (for *NOT*) or several (two or more – for *AND* and *OR*) outgoing arcs. Graphically each operator is represented as an oval, containing the name of the corresponding operation.
- Terminal nodes (leaves) that can not have any out going arcs. Such terminal nodes are called *instances*. Each instance corresponds to the WSML statement *Var* hasValue *Instance*, where *Var* is a WSML variable and *Instance* is an instance of a concept from a given ontology. Graphically an instance is represented by a rectangle with header containing the name of concept, an instance of which the Instance is, and the concrete name of the instance.

Directed arcs of a graph are called *connections*. A connection outgoing from a variable or relation has the meaning of refining the variable (or relation parameter) value and corresponds to WSML logical operator *AND*. A connection outgoing from an operator has the meaning of a pointer to the operator operand.

The proposed model allows to consider the process of axiom creation as a formal process of DAG expanding (and editing) and to formulate formal rules for checking syntactic and semantic (in relation to given ontologies) correctness of constructed axioms.

An advantage of the proposed model is ability to separate logical *AND* (represented as the model *AND* operator) used by the axiom creator for describing logical conjunction at a high level of abstraction from a "hidden", "technical" *AND* (represented by the model connection) used for specifying more concrete values of variable attributes. As a result, the explicit logic conjunction may be used in the model only as a part of a path starting from the axiom root and ending in an intermediate variable node or in a terminal node. This has a very important consequence for the semantic service discovery process. First, if a represented in such a way axiom is interpreted as a user goal (i.e. a request for desired service functionality), the proposed mechanism gives a very simple method for splitting the goal to sub-goals. And second, if such an axiom is interpreted for example as a service post-condition, the proposed mechanism allows easily determining if the service offers a single functionality of a set of different functionalities.

An Informal Model of the Axiom Construction Process

A process of axiom creation may be considered as a repetitive process consisting of combination of three main logical steps – definition, refinement (or specialization) and logical development (or elaboration). The *definition* step is used for defining some general concepts needed for describing the meaning of axioms. The *refinement* step is used for more concrete specification of desired properties of such concepts. Such a step may be seen as specialization of too general concepts introduced earlier. The *logical development* step consists of elaborating logical structure of the axioms, which may be achieved by combination of general concepts by means of logical operators *AND*, *OR* and *NOT*.

Syntactic and semantic checks applied during the all phases of axiom creation process are based on the following properties:

- Subsumption relation between different elements of ontologies: such a relation determines compatibility between axiom variables;
- Acyclic property of the selected model (DAG) for representing an axiom;
- Uniqueness of the names of variables used for constructing an axiom (if contrary is not explicitly specified);
- Arity of logical operators used for constructing an axiom.

Definition Step

During the definition step the nature of a main variable defining the axiom is specified. Such a step is equivalent to creating a WSML statement *?Concept* memberOf *Concept*, which means that the WSML variable *?Concept* copying the structure of the *Concept* from a given WSML ontology is created. Attributes of the concept, which are "inherited" by the axiom model variable, are named *variable attributes.* By default the values of such attributes are set to free WSML variables with type defined by the definition of such attributes in the corresponding ontology.

It should be mentioned that in the definition step every concept, instance or relation from an arbitrary WSML ontology may be used as a template for creating the corresponding axiom variable.

Refinement Step

The refinement step is a recursive procedure of refining values of some attributes (relation parameters) defined in previous step(s). In terms of our model each cycle in such a step means an expansion of an existing non-terminal node – variable (or relation). More precisely that means a selection of an attribute from a list of available attributes of an existing axiom variable, and binding its value (which in this moment is a free WSML variable) to another (new or existing) node of the axiom model. The main problem is to ensure semantic correctness of the resulted (extended) logical expression. Such correctness is achieved by applying explicit rules determining permitted expansion of a given node.

An attribute value[1] of an axiom variable may be refined by binding it to:

A. A new variable produced from the ontology concept specified by ofType or impliesType WSML statement for the corresponding attribute (default binding);
B. A new variable produced from a subconcept of the ontology concept specified by ofType or impliesType WSML statement for the corresponding attribute;
C. A new terminal node – instance produced from an instance of the corresponding concept or of its subconcepts;
D. A relation which parameters are compatible with the type of the selected attribute;
E. An existing axiom variable, which are compatible with the type of the selected attribute and which does not lead to creation of cycles in the model.
F. A shared variable with compatible type.
G. A complex logical expression composed from all mentioned above items by logical operators OR and NOT.

Logical Development Step

This step of the axiom construction process consists in adding logical operations (*AND*, *OR* and *NOT*) to the current logical expression. Such operators may be added to connect two independently constructed logical expressions or be inserted directly into already constructed expressions. In both cases it leads to creating more complex logical expressions.

A logical operator can be inserted only into a connection that has been already created as a part of the axiom model. Such an insertion "splits" the connection on two parts, which are linked by newly inserted logical operation. Since operators *AND* and *OR* should have at least to operands, the addition of such logical operators requires creating the second operand, which can be either a new or an existing axiom element. The operation is controlled by context-dependent semantics and syntactic checks so different logical operators can be inserted only in some allowed places in the axiom. Such checks analyze the whole context of the axiom, which in some cases leads for necessity to verify the path from the edited element till the starting axiom element – the axiom *Root*.

It should be underlined that during this step the user is constructing the axiom by logical combination of main axiom objects defined in the previous steps. In other words, the logical operators are used not for refining or clarifying the meaning of some parameters of already defined objects, but for complicating the axiom by specifying the logical connections between some axiom parts which are independent in their meaning.

1 The same rules are applicable to every unbound relation parameter.

Functional Architecture

The functional architecture of the Axiom Editor provides a complete set of functions (operations) needed for graphical constructing WSML logical expressions. The top-level functional components of the Editor are:

- *Ontology Store* – a set of operations for maintaining ontologies used for creating and editing axioms.
- *Axiom Model Generator* – a set of operations for graphical constructing and editing an axiom.
- *Axiom Text Generator* – the module providing automatic generation of the WSML text corresponding to the current graphical model of an axiom.
- *Axiom Persistence* – the module providing saving and retrieving axioms as well as all information needed for axiom creation.

Ontology Store

The Ontology Store is an *in-memory* set of ontologies providing the semantic elements for constructing axioms. These elements are concepts, attributes, instances, relations and parameters[1]. The Ontology Store is global to all axioms opened in the Editor.

In order to be used in the Axiom Editor ontologies should be defined in the WSML language. To start creating a new axiom at least one ontology is needed. The Axiom Editor reads ontologies from *.wsml files. The pasring of these files is done by a standard WSML parser which is a part of the WSMO4J API [WSMO4J 2005].

A *tree structure* is used for graphical representation of ontologies. Since *.wsml files are flat (they have no hierarchical structure), additional information is obtained from the WSMO4J API to construct a tree from the lists of concepts, relations etc. The API provides information about concept and relation inheritance by a special SuperConcepts property that every ontology element possesses. It should be noted that this property is a set, which means that one element can have more than one parent in the hierarchy. In tree-structured visualization every child element appears as many times in the tree as there are concepts in its SuperConcepts property. A visualized ontology may be browsed and all properties associated with each ontology element are shown in a special window.

Ontologies may be loaded manually by the user from the file system or loaded automatically on- demand. Ontologies describe inheritance between concepts. A concept usually has one or more super-concepts. Super-concepts may be defined in other ontologies. For example the concept “Person”, defined in the ontology “Sociology”, may have the concept “Human”, defined in the ontology “Biology” as its super-concept. In such a case, the “Sociology” ontology declares “Biology” as an *imported ontology*. The “load imported ontology” operation can be applied to such concepts displayed in the Ontology View which are defined in imported ontologies. Since an imported ontology is declared with its identifier, the URI is used to locate that ontology and load it to the Ontology Store. The concept is automatically located in the new tree, the concept’s attributes become available so variables of that type can be now created.

A concept inherits all its super-concepts’ attributes. If a super-concept is defined in an imported ontology, which is not currently loaded to the Ontology Store, then the super-concept’s attributes are unavailable. The mechanism for on-demand loading of imported ontologies provides automatic updating concepts’ attributes inherited from super-concepts belonging to such ontologies.

Axiom Model Generator

As it has been already mentioned, the main concern of the Axiom Editor is to *guarantee the semantic consistence* of the constructed logical expressions since the users of this tool are assumed to be non-specialists in the first-order logic. Such a consistence is achieved by a semantically-aware construction process, in each step of which the user is allowed to perform only such operations that are consistent with the already constructed part of the axiom.

1 Functions are not supported in the current implementation of the Capability Editor. Such elements of a WSML ontology as non-functional properties and ontology axioms are shown in the Editor but currently are not used in the process of axiom constructing.

Two modes for axiom construction are available:

- *Standard mode* involves only extending an existing part of the axiom by selecting semantically compatible elements from context-sensitive menus. This method is construction-driven and is suitable for novice users.
- *Advanced mode* allows adding isolated elements to the modelling area, which can be later combined in various semantically correct ways. This allows advanced users to be more efficient.

The axiom construction process begins by selecting a concept from Ontology Store. This concept is used to create the first variable in the axiom model. The variable's type is equal to the selected concept. Automatically, just after adding the first variable to the model, it is connected to the Axiom root element "Start".

From this moment on, the construction process continues by performing semantically-correct operations on different elements in the axiom model which can be: variables, variable attributes, instances, connections, operators, relations and relation parameters. A summary of the most important semantically-correct operations in Axiom Model Generator are shown in Table 1.

Operations for creating elements of Axiom Model	
Create a variable	Creates a new variable in the graphical axiom modelling area (window). The type of the variable is selected by the user from Ontology Store. The name of the variable is automatically generated from the name of the selected concept guaranteeing the uniqueness of variable names across the axiom.
Create an operator	Creates a new logical operator of a specified type in the modelling area. The operator's type is selected from the menu – it can be *OR*, *AND* or *NOT*.
Create an instance	Adds an instance to the graphical modelling area. The user is given the opportunity to select the instance from Ontology Store.
Create a connection (advanced mode)	Creates a new connection between two elements placed on the modelling area. The user selects a source and a target element for the new connection. The selection is restricted only to semantically-compatible source and target elements.
Create a relation	Adds a relation to the modelling area. The user is given the opportunity to select an arbitrary relation from Ontology Store.
Operations on Variables	
Rename a variable	The user can change the automatically generated variable name as long as the uniqueness of names is not violated. The Axiom Editor takes care of changing the variable's name from the old one to the new at all its occurrences in the model.
Involve a variable in a relation (Advanced mode)	A variable that has been already placed at the axiom modelling area may be further involved in a relation also presented at this area. More exactly, such an operation creates a connection linking the variable with a parameter of the relation. Operation is possible only when the variable and the selected relation parameter have compatible types.
Delete variable	Deletion of a variable leads to deletion of all incoming and outgoing connections of the selected variable in the model, thus keeping the axiom consistent.
Operations on attributes of a variable	
Refine an attribute by a variable	Creates a new variable at the modelling area and links the selected attribute value to it with a connection. The meaning of the operation is that the value of the attribute is equal to this new variable. The name of the new variable is automatically set equal to the name of the selected attribute value being refined.
Refine an attribute by an instance	Adds to the current axiom a new instance selected from an ontology and links it to the attribute value to be refined by a connection. The user is given the opportunity to select such an instance from a special dialog window containing a subset of instances from the Ontology Store. More exactly, in order to preserve the semantic consistence of the axiom, the selection is limited only to those instances, whose concepts are equal to or are sub-concepts of the concept specified as the type of the chosen attribute.

Refine an attribute by involving into a relation	A value of an attribute of a variable from the current axiom may be further refined by specifying that it is involved in a relation defined either in the Ontology Store or already placed at the modelling area. Selecting the attribute to be refined restricts a set of relations that may be applied to the value of such an attribute – that are all relations, which have parameters with types compatible with the type of that attribute.

Operations on relations and relation parameters	
Refine a relation parameter	A set of available operations on relation parameters is practically the same as the operations working on values of attribute variables (see “Operations on attribute of a variable”).
Delete relation	Deletion of a relation leads to deletion of all its incoming and outgoing connections in the model, thus keeping the axiom consistent.
Operations on operators	
Change operator type	Is used for changing the type of an operator selected from the modelling area.
Delete operator	Potentially leads to creating some orphaned axiom model elements. In order to preserve the semantic consistence of the axiom, such “orphaned elements” are not included in the axiom text generation.
Add operand	Adds a new operand to a selected operator placed at the modelling area and links them by a connection. The new operand can be either an existing model element from the modelling area (variable, relation, instance, etc.) or a new element that can be created by means of already described operations, which the user may select from right-click sub-menu.
Operations on instances	
Edit an instance of WSML built-in data types	Can be performed on such instances of the axiom model which have a WSML built-in data type or a subtype of such type. The value of these instances is entered by the user and can be edited later.
Delete an instance	Leads to deletion of the instance along with all connections incoming to it from the model, thus keeping the axiom consistent.
Operations on Connections	
Insert an alternative	The main meaning of a connection in the axiom model is that the target element of the connection is used as a refinement of its source element. It is natural to allow the user to define an alternative (or several alternatives) for such a refinement. In order to insure that such an operation will be meaningful, it is necessary to restrict its application domain.
Insert an AND operator	Aims at allowing the user to specify explicitly logical conjunction of two axiom model elements and is also used during the “Logical development” phase of the axiom model construction process operator as its second operand.
Insert a NOT operator	Inserts a NOT operator in the middle of any connection.
Reconnect a source/target element (Advanced mode)	Moves the starting/ending point of the connection to another element in the Axiom Model. In order to preserve the semantic consistence of the axiom, the operation can be performed only if the new source/target element is semantically-compatible with the type of the edited connection.

Table 1. The most important semantically-correct operations in the Axiom Model Generator

Axiom Text Generator

The Axiom Text Generator dynamically generates text representation of the graphical axiom model in the human-readable WSML-Core syntax. That allows to observe and control (for experience users) the result of each operation accomplished on the axiom model. It should be mentioned that only elements of the modelling area having connections with the root elements of the axiom model (*Start* element) are considered as parts of the current axiom and, hence, are mapped to its WSML text representation.

Axiom Persistence

Creating a semantic Web service is a rather complex process, which may need a lot of time, so it is necessary to have a module for storing all intermediate results and supplemented data structures facilitating such a process. The Axiom Persistence is such a module that is used for storing and retrieving axioms created by the Axiom Editor. Since an axiom has no meaning without the ontologies used for its creation, loading an axiom leads to automatic loading of all ontologies associated with it.

Axioms are persisted in binary files which can only be opened by the Axiom Editor. Besides the semantic content, all elements store their graphical coordinates so that the graphical model of an axiom can be fully restored. During the loading process different validations are made. If any of them fails, an error message is displayed and the axiom file is not loaded. For implementation of these operations Java serialization is extensively used.

Currently the Axiom Editor uses a predefined directory called the Ontology File Store in the file system to store *.wsml files containing ontologies. Every ontology has a unique identifier, which is a URI written in the *.wsml file defining the ontology. When an ontology, whose identifier is known, must be loaded, the Axiom Editor searches the Ontology File Store for that identifier and loads the respective ontology to the Ontology Store.

Graphical User Interface

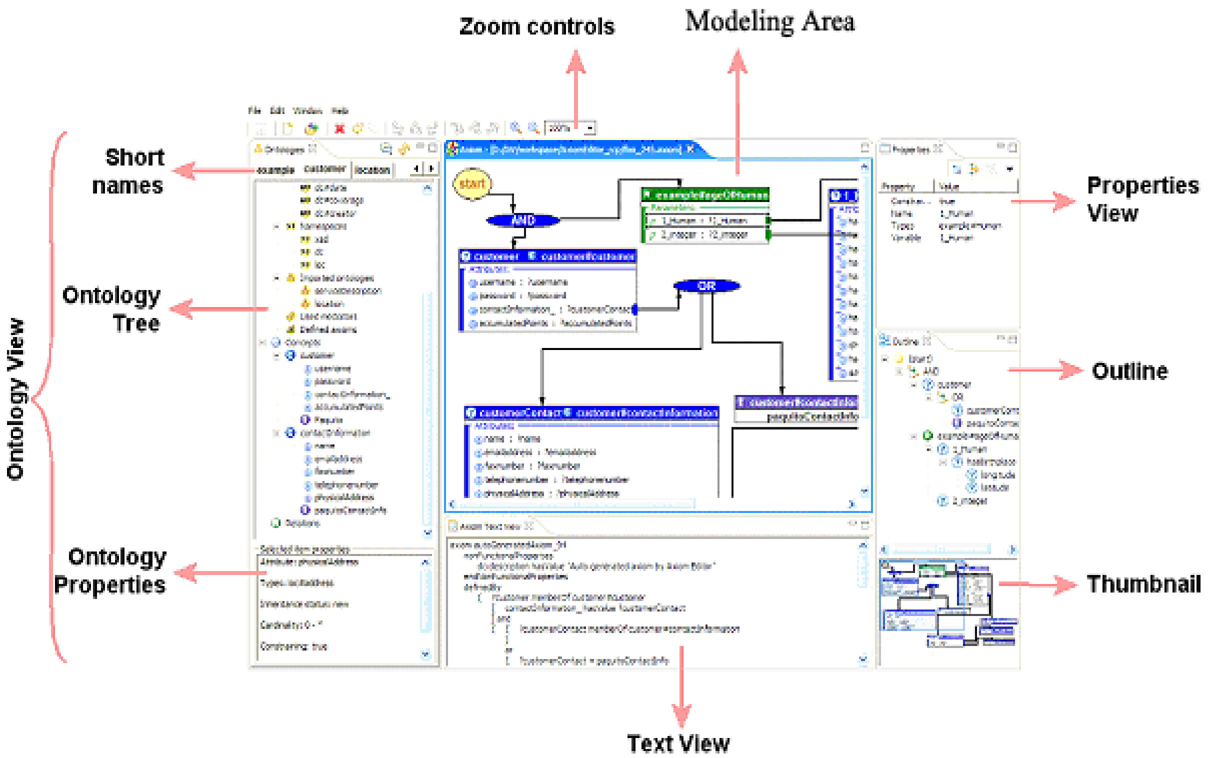

Figure 1. An overview of the Axiom Editor workspace

The Axiom Editor runs as an Eclipse plug-in. Eclipse [Des Rivieres and Wiegand 2004] is a free, integrated development environment (IDE) which can host different third-party applications, providing a unified visual outlook and better integration between them.

The Axiom Editor is bundled as a standalone application on top of the Rich Client Platform (RCP). The RCP is a compact Eclipse core which can also host plug-ins. It provides a startup executable which runs a lightweight version of the IDE and automatically loads the appropriate plug-in (in this case – the Axiom Editor).

An overview of the Axiom Editor workspace is shown on Figure 1. The screen is divided in several major areas: Ontology View, Modelling Area, Properties View, Outline View, Thumbnail + zoom controls and Text View.

You can find a more detailed description of the workspace areas in Table 2.

Workspace area	Description
Ontology View	Contains all loaded WSMO ontologies (Ontology Store). At the top of the view there is a list of tabs used to switch between different ontologies.
Ontology Tree	The centre part of the ontology view containing all ontology elements, structured in a hierarchy. The nodes represent: concepts, attributes, instances; relations, parameters, relation-instances; non-functional properties, namespaces; imported ontologies, used mediators; defined ontology axioms.
Ontology Properties	The bottom part of the Ontology View section. It contains details in plain text about the selected element in the Ontology Tree such as the non-functional properties of a concept, the definition of an ontology axiom etc.
Modelling Area	Contains the graphical representation of the axiom model. The model is displayed as a directed acyclic graph, reflecting the tree structure of the logical expression. This is where the user creates axiom elements out of ontology elements and adds dependencies between them through the use of semantically consistent operations. Axiom elements are: Variables, Instances, Relations, Logical Operators, The start element. Dependencies between these elements are introduced through the use of connections displayed as directed arrows.
Properties View	Displays the properties of the selected element in the Modelling Area. Different kinds of elements have different sets of properties – some of them read-only, others - editable.
Outline View	Displays a classical tree representation of the logical expression. The branches of the tree can be expanded or collapsed to help the viewer better perceive the high-level structure of the expression. It also allows for easier navigation among the elements. If an element is selected in the Outline View, it becomes also selected in the Modelling Area and its properties are displayed in the Properties View.
Thumbnail	A mini-map of the whole modelling area. On large models it helps the user to not lose the whole picture, makes navigation easier and always highlights the part of the model being displayed in the Modelling Area.
Zoom Controls	Provide a way of getting a larger part of the model into view by selecting zoom-factor less than 100%. If the user selects a zoom-factor above 100% details can be clearly seen and elements can be more precisely aligned in the Modelling Area.
Text View	Contains the WSML representation of the axiom. It is automatically refreshed whenever something changes in the graphical axiom model to reflect the current state of the expression. It is useful for advanced users who want to always know the exact impact of their actions on the capabilities of the web-service they are designing.

Table 2. Description of Axiom Editor Workspace areas

Conclusion

The Axiom Editor is implemented in J2SDK 1.4.2 runtime environment and uses basic platform components, plug-in infrastructure, graphical user interface components (menus, buttons, tree views, event handling) from Eclipse RCP (Rich Client Platform). For development of visual designers the Eclipse GEF (Graphical Environment Framework) is used. Access to WSMO-based ontologies is accomplished via *WSMO4J (WSMO API)*.

Main directions or future development of the Editor are as follows:

- Transformation of the Axiom Editor to an integrated Service Capability Editor by extending it with some customized modules of WSMO Studio [[WSMO Studio 2005] and integrating with the Axiom Case-base Memory.
- Extending application domain of the Axiom Editor by expanding the range of logical operations used (e.g. including *implies, impliedBy, :-* and *!* operators). As a result the Editor could be used not only for creating the SWS capabilities but for constructing axioms in WSML ontologies as well.

Acknowledgement

This work is carried out under EU Project INFRAWEBS - IST FP62003/IST/2.3.2.3 Research Project No. 511723.

Bibliography

[Agre et al. 2005] G. Agre, T. Atanasova, J. Nern. Migrating from Web Services to Semantic Web Services: INFRAWEBS Approach. In: *Proceeding of Eleventh Annual Scientific Conference on Web Technology, New Media, Communications and Telematics Theory, Methods, Tools and Applications, EUROMEDIS'2005*, April 11-13, 2005, Toulouse, France, 221-225.

[de Bruijn et al. 2005] J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer and D. Fensel. *D16.1v0.2 The Web Service Modeling Language WSML, WSML Final Draft 20 March 2005,* available at: http://www.wsmo.org/TR/d16/d16.1/v0.2/20050320/.

[Des Rivieres and Wiegand 2004] J. Des Rivieres and J. Wiegand. Eclipse: A platform for integrating development tools. *IBM Systems Journal, 43(2)*, 2004.

[Enderton, 2002] H. B. Enderton. *A Mathematical Introduction to Logic (2nd edition)*. Academic Press, 2002.

[Gruber 1993] T. Gruber: A translation approach to portable ontology specifications, *Knowledge Acquisition, 5:*199-220, 1993.

[Kifer et al. 1995] M. Kifer, G. Lausen, and J. Wu: Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741-843, July 1995.

[Nern et al. 2004] H.-Joachim Nern, G. Agre, T. Atanansova, J. Saarela. System Framework for Generating Open Development Platforms for Web-Service Applications Using Semantic Web Technologies, Distributed Decision Support Units and Multi-Agent-Systems - INFRAWEBS II. *WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS*, *ISSN 1790-0832, Issue 1, Volume 1*, July 2004, 286-291.

[OWL-S 2004] *The OWL Services Coalition: OWL-S: Semantic Markup for Web Services, version 1.0*; available at http://www.daml.org/services/owl-s/1.0/owl-s.pdf

[Protégé 2005] http://protege.stanford.edu/index.html

[Roman et al. 2005] D. Roman, H. Lausen, U. Keller, J. de Bruijn, Ch. Bussler, J. Domingue, D. Fensel, M. Kifer, J. Kopecky, R. Lara, E.l Oren, A.l Polleres, M. Stollberg. *D2v1.1. Web Service Modeling Ontology (WSMO) - WSMO Final Draft 10 February 2005*, http://www.wsmo.org/TR/d2/v1.1/

[WSMO4J 2005] http://wsmo4j.sourceforge.net/.

[WSMO Studio 2005] http://www. wsmostudio.org.

Authors' Information

Gennady Agre – Institute of Information Technologies – Bulgarian Academy of Sciences, Acad. G. Bonchev St., block 29A, Sofia 1113, Bulgaria;

Petar Kormushev – Sofia University St. Kliment Ohridski;

Ivan Dilov – Sofia University St. Kliment Ohridski;