

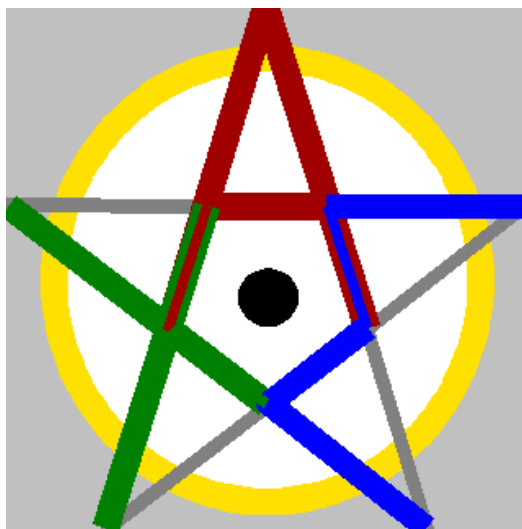


FP6-511723

INFRAWEBBS

Intelligent Framework for Generating Open (Adaptable)
Development Platforms for Web-Service Enabled Applications Using
Semantic Web Technologies, Distributed Decision Support Units
and Multi-Agent Systems

Specific Targeted Research Project
Priority 2 "Information Society Technologies"



INFRAWEBBS Axiom Editor User's Guide

Version 1.0.5

Gennady Agre
Petar Kormushev
Ivan Dilov

TABLE OF CONTENTS

1 INTRODUCTION.....	3
2 THE WORKSPACE ELEMENTS	3
2.1 Ontology View.....	3
2.2 Diagram Area.....	3
2.3 Properties View.....	3
2.4 Outline View.....	3
2.5 Thumbnail + zoom controls	3
2.6 Text View	3
2.7 The main menu.....	3
3 AN INFORMAL MODEL OF THE AXIOM CONSTRUCTION PROCESS	3
3.1 Definition Step	3
3.2 Refinement Step	3
3.3 Logical Development Step.....	3
4 USING AXIOM EDITOR	3
4.1 Loading ontologies	3
4.2 Imported ontologies	3
4.3 On-demand loading	3
4.4 Menu for operations available at the axiom definition step	3
4.5 Two modes for axiom construction.....	3
4.6 The first variable	3
4.7 Refining attributes by selection.....	3
4.8 Refining attributes by default.....	3
4.9 Refining attributes by manual connection	3
4.10 Multiple value attributes.....	3
4.11 Use of relations.....	3
4.12 Use of logical operators.....	3
4.13 Renaming Variables	3
4.14 Operations for editing connections.....	3
4.15 Manipulating large axiom models.....	3
4.16 Hiding unused attributes.....	3
4.17 Saving and Loading axioms	3
4.18 Saving and Loading axioms as wsml files.....	3
5 CONCLUSION AND FUTURE TRENDS	3

REFERENCES3

APPENDIX A. TABLE OF OPERATIONS.....3

1 INTRODUCTION

INFRAWEBs Axiom Editor is an ontology-driven user-friendly tool for graphical construction of complex WSML logical expressions. This guide provides information about how to use the tool.

It is not required to know WSML in order to use Axiom Editor as an axiom construction tool. Therefore, this guide does not contain an introduction to neither WSML, nor WSMO. More experienced users are advised to get familiar with the WSML language (Web Service Modeling Language) [Roman et al., 2005], which provides a formal syntax and semantics for the Web Service Modeling Ontology (WSMO) [Bruijn et al. 2005].

The main objective of the INFRAWEBs Axiom Editor is to construct valid WSML logical expressions to be used for describing a capability of WSMO-based Semantic Web Services (SWS). Up to now all known approaches to this problem are reduced to some (enhanced) text editors creating such logical expressions as simple strings without any support to the process of axiom construction. Axiom Editor, on the other hand, is an entirely graphical tool for construction. It was developed to be as much easy to use as possible.

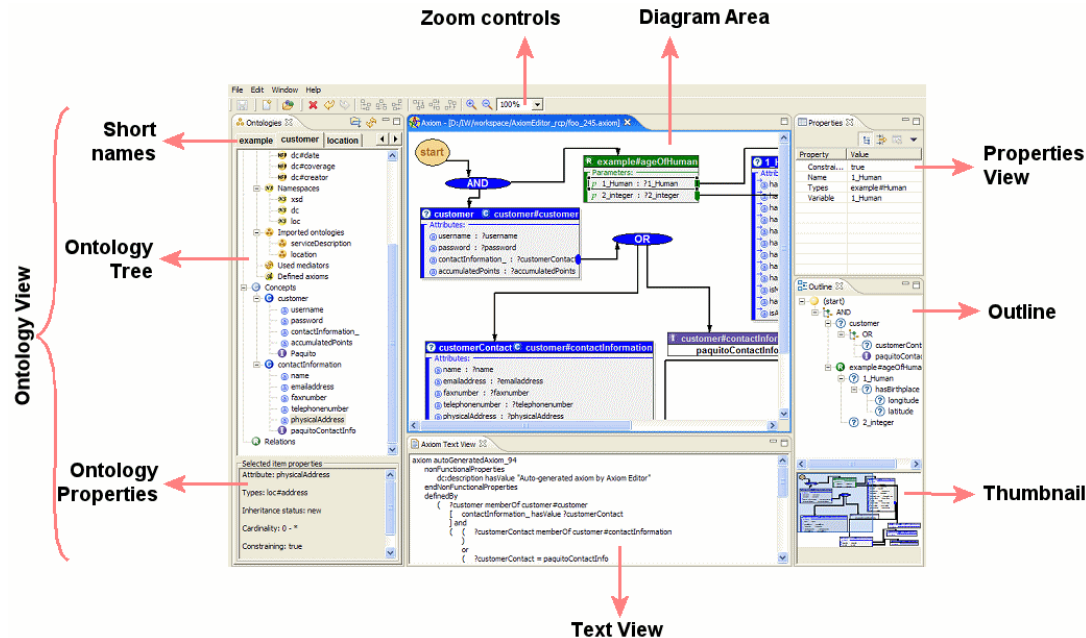
The INFRAWEBs Axiom Editor runs as an Eclipse plug-in. Eclipse is a free, integrated development environment (IDE) which can host different third-party applications, providing a unified visual outlook and better integration between them. Some other SWS projects are also developed as Eclipse plug-ins. More information about Eclipse can be found in [Des Rivieres and Wiegand 2004].

The INFRAWEBs 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 – Axiom Editor).

The INFRAWEBs Axiom Editor is developed by the Institute of Information Technologies – Bulgarian Academy of Sciences in the frame of FP6-511723 IST Project INFRAWEBs.

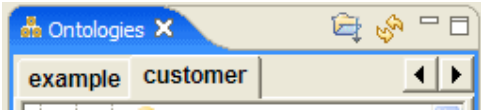
2 THE WORKSPACE ELEMENTS

The screen is divided in several major areas: Ontology View, Diagram Area, Properties View, Outline View, Thumbnail + zoom controls and Text View.



2.1 Ontology View

The Ontology View contains all loaded WSMO ontologies. The Ontology View is read-only – the user cannot modify the ontologies in any way.

- At the top of the view there is a list of tabs used to switch between different ontologies. When several ontologies have been loaded, the list can be scrolled sideways using the arrows that appear to the right. All ontologies are automatically given **unique short names** which are displayed in the tabs and are used throughout the whole workspace. The short name is based on the last part of the ontology IRI.
 
- The center part of the Ontology View is the **Ontology Tree** which contains 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.


The user can browse the elements and select concepts, instances and relations for the creation of axioms.


The structure of the sub-tree of  **Concepts** reflects:

- the hierarchy of inheritance between the concepts – sub-concepts are displayed as children of all their super-concepts;
- the memberships of instances to their concepts – all instances of a concept are displayed as children of that concept;
- the membership and inheritance of attributes – all attributes of a concept are displayed as children of that concept and the icon specifies the origin of the attribute.




There are three possible types of attributes with respect to their origin:

-  **new** attributes – defined for the first time by the concept they are children of;
-  **inherited** attributes – defined by a super concept and left unchanged in the sub-concept;
-  **overridden** attributes - defined by a super concept and changed in the sub-concept. For example, the sub-concept may have changed the attribute range or may have imposed additional constraints on its values.

The same holds for the sub-tree of  **Relations** – sub-relations, parameters and relation instances are aligned in the same way as sub-concepts, attributes and instances respectively.

The sub-tree called  **More information** contains all ontology metadata and elements which are not used for the creation of axioms. These are non-functional properties, namespaces, imported ontologies, used mediators and axioms defined in the ontology. Such information can only be browsed by the user to get a more comprehensive view of the ontology.

The bottom part of the Ontology View is the **Ontology Properties** section. It contains some textual details about the selected element in the Ontology Tree such as the non-functional properties of a concept, the definition of ontology axioms etc.

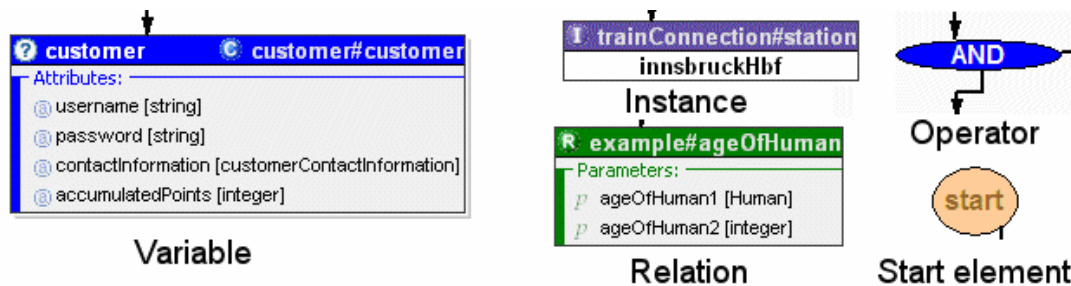
The most important ontology elements in the INFRAWEBs Axiom Editor are  **concepts**,  **instances** and  **relations** since they are the building elements of axioms. All concepts, instances and relations can be drag-and-dropped from the Ontology Tree into the Diagram Area thus creating axiom elements. Double-clicking on Ontology Tree elements has the same effect as drag-and-drop operation except that their positions in the diagram are selected at random.

2.2 Diagram Area

The **Diagram Area** contains the graphical representation of the axiom (axiom model). The model is displayed as a directed acyclic graph, reflecting the tree structure of the logical expression. The Diagram Area is a place where the user creates axiom

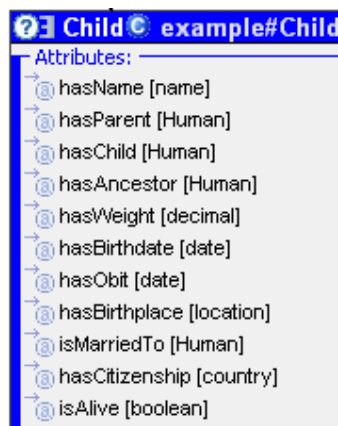
elements from selected ontology elements and adds dependencies between them by using semantically consistent operations.

Axiom elements are:



Variables are created from ontology concepts. They are displayed as blue colored rectangles. The word following the symbol is the variable name, and the word following the symbol defines the variable type. A variable may contain attributes that are shown as: <name> [<type>], where the name and the type come from the definition of the corresponding concept in the ontology.

If the variable has its property “Exists” set to *True*, then a small sign appears beside its icon, like this:



Instances are created from ontology instances. They are displayed as violet colored rectangles. There are 2 types on instances: instances of WSML build-in types (e.g. “string”) and normal instances presented in ontologies (e.g. “Paul”).



Relations are created from ontology relations. They are displayed as green colored rectangles.

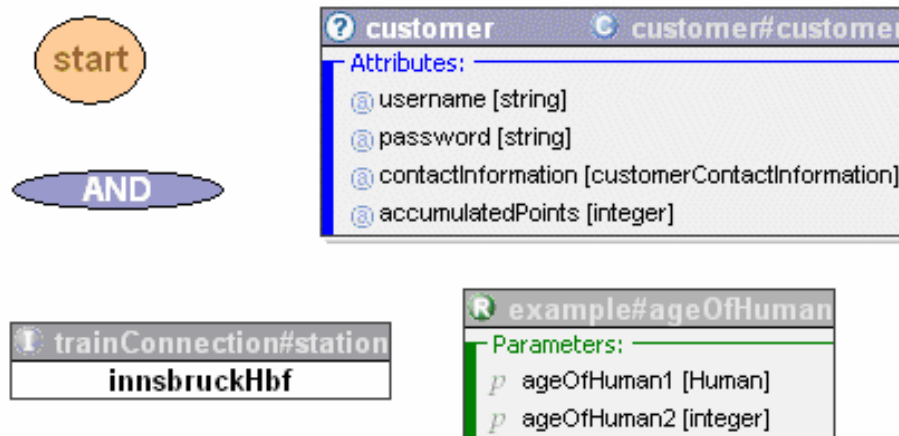
Logical Operators are displayed as ellipses. At present only AND, OR and NOT are supported.

● The **start** element. It is a single yellow ellipse with the word "start" in it, pointing to the root element of the logical expression.

Dependencies between these elements are introduced through the use of **connections** displayed as directed arrows. The INFRAWEBs Axiom Editor forces many restrictions on these connections in order to preserve the semantic consistence of the axioms. The connections are only allowed:

- from the start element to exactly one variable, instance, relation or operator. The target is the root element of the logical expression.
- from variable attributes to variables, instances or operators. The target is the root of a sub-expression refining the value of the attribute.
- from relation parameters to variables, instances or operators. The target is the root of a sub-expression refining the value of the parameter.
- from operators to variables, instances, relations or operators. The connections reflect the logical structure of the expression tree.

The axiom elements, which are not connected to the axiom tree, are shown with a half-tone color, like this:

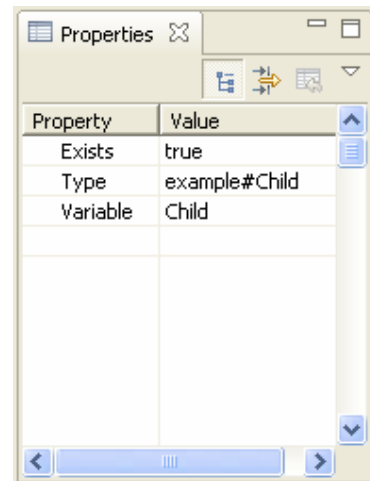


Several axioms can be simultaneously opened for editing. They appear as tabs in the Diagram Area. They all share the ontologies loaded into the Ontology View.

2.3 Properties View

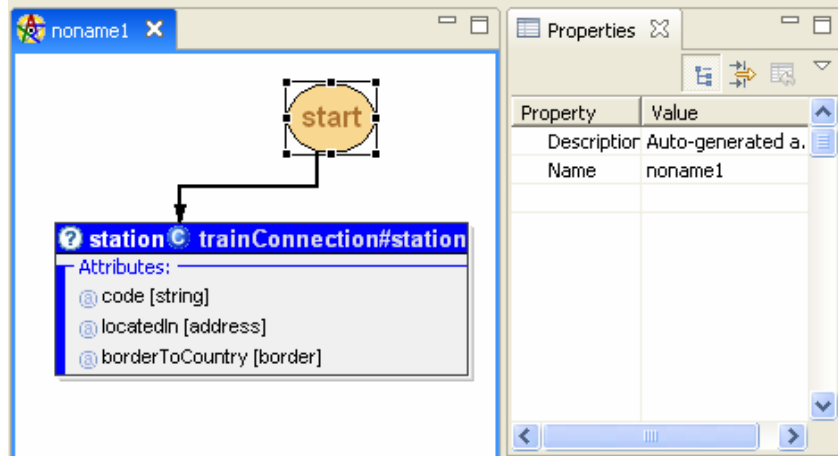
The **Properties View** displays the properties of the selected element in the Diagram Area. Different kinds of elements have different sets of properties – some of them read-only, others - editable. For example, a variable has only three properties:

1. A read-only **type** which is the qualified name of a concept.
2. An **“Exists”** Boolean flag with values True or False. This is used to set whether the variable should be included in the “Exists” clause in the axiom text or not.
3. An editable **variable name** which is a string that must be:
 - unique within the axiom model, and
 - valid with respect to the WSMML syntax for variable names. This means it must contain only alphanumeric characters and a few other characters like ‘_’ for example. Detailed description of the WSMML syntax can be found in [Bruijn et al. 2005].

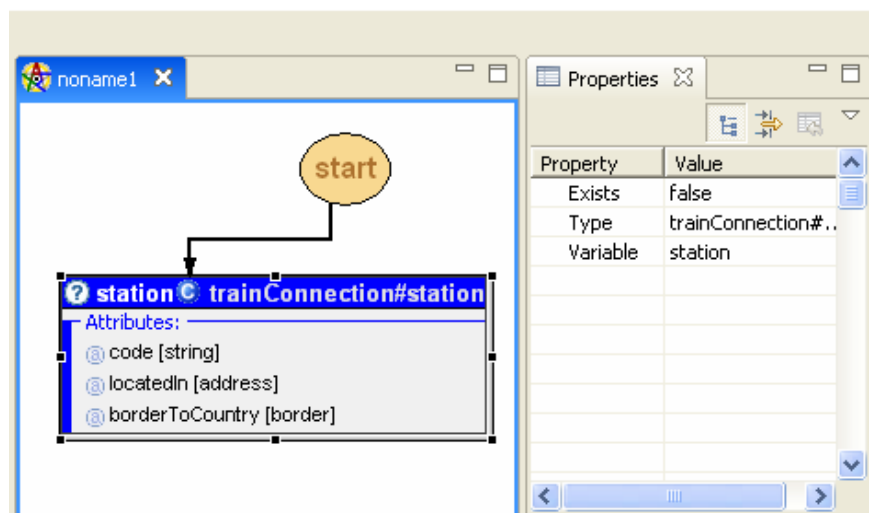


Here are three examples of different property views when different elements from the axiom are selected:

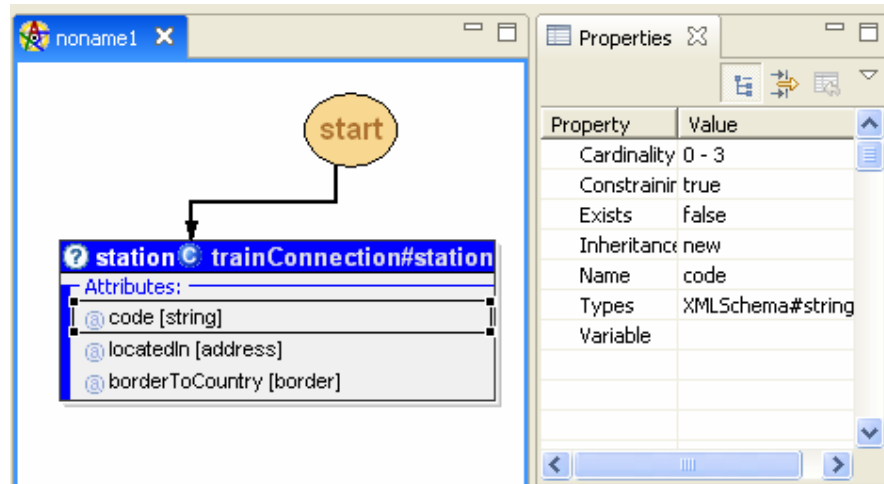
If the “start” element is selected:



If a variable is selected:



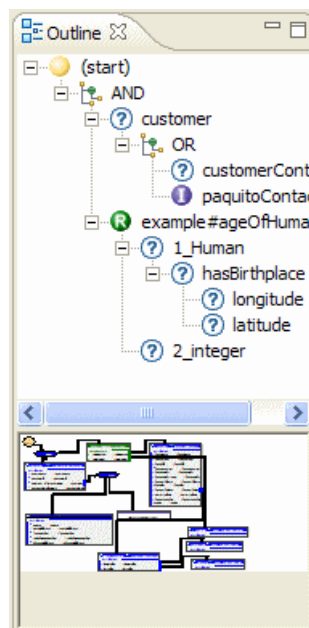
If an attribute is selected:



The name of a variable can be easily changed by editing the “Variable” property in Properties View or right at the variable rectangle header.

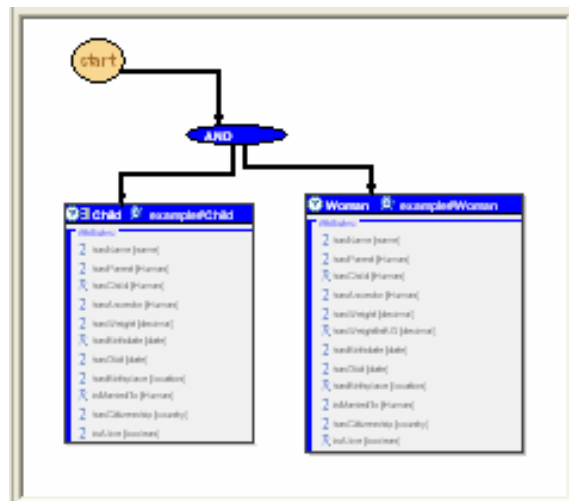
2.4 Outline View

The **Outline View** displays a classical tree representation of the logical expression. The branches of the tree, at all levels, can be expanded or collapsed to help the viewer better perceive the high-level structure of the expression. It also allows easier navigation among the elements. If an element is selected in the Outline View, it becomes selected in the Diagram Area as well and its properties are displayed in the Properties View.

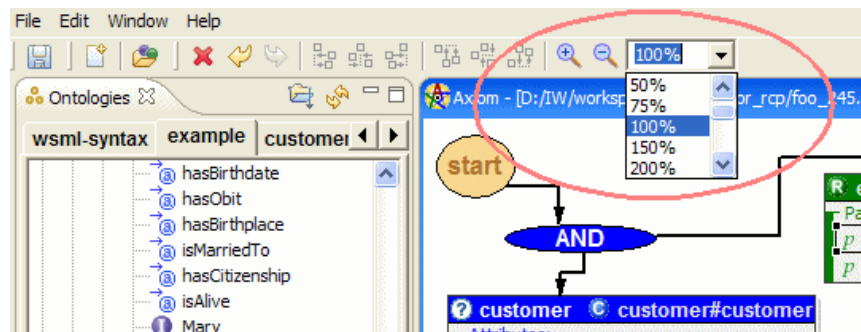


2.5 Thumbnail + zoom controls

The **Thumbnail** is a mini-map of the whole diagram. For large diagrams it helps the user to not lose the whole picture, makes navigation easier and always highlights the part of the diagram being displayed in the Diagram Area.



The **Zoom Controls** provide a way of getting a larger part of the diagram into view by selecting zoom-factor less than 100%. On the contrary – if the user selects a zoom-factor above 100% details can be clearly seen and elements can be more precisely aligned in the Diagram Area.



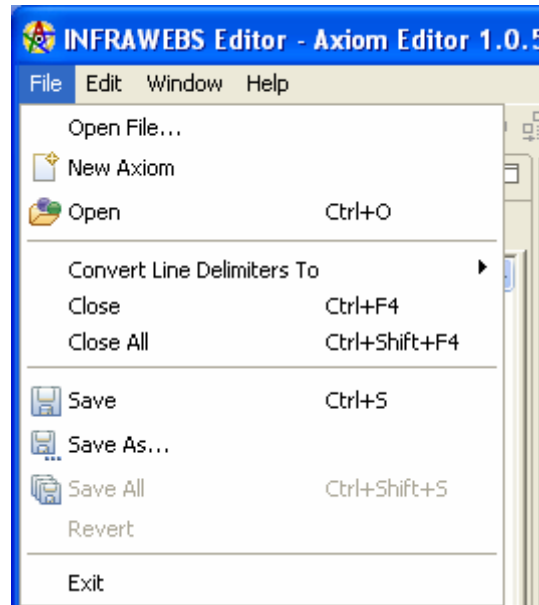
2.6 Text View

The **Text View** contains the WSMML representation of the axiom. It is automatically refreshed whenever something is changed in the diagram to reflect the current state of the expression. It is useful for advanced users who want to know the exact impact of their actions on the final description of the axiom.

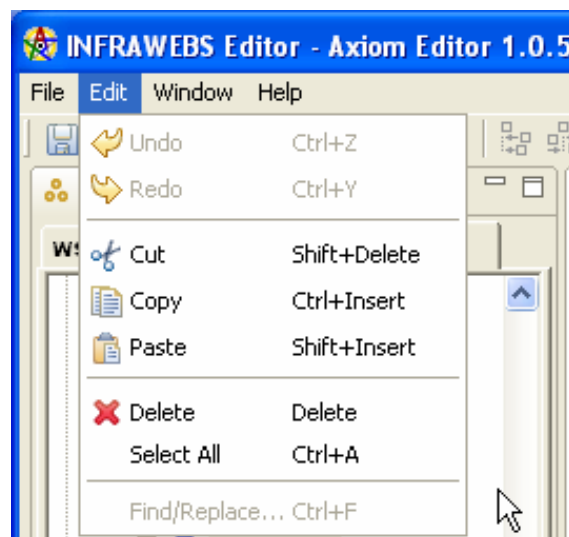
2.7 The main menu

The main menu contains the following top-level menu items: File, Edit, Window, Help.

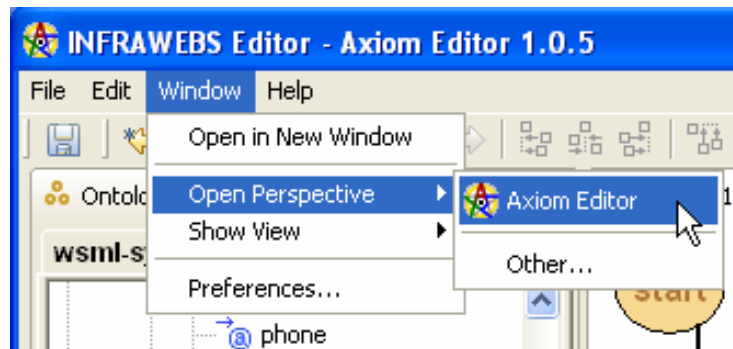
The *File* menu contains operations to Open/Save/Close axioms created with this editor. The user can also create new axiom with the “New Axiom” menu item:



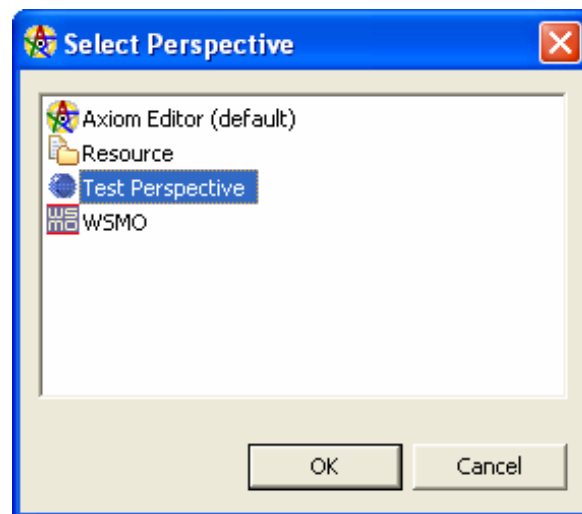
The *Edit* menu contains standard editing operations like Cut/Copy/Paste/Delete. It also provides Undo/Redo functionality to the user:



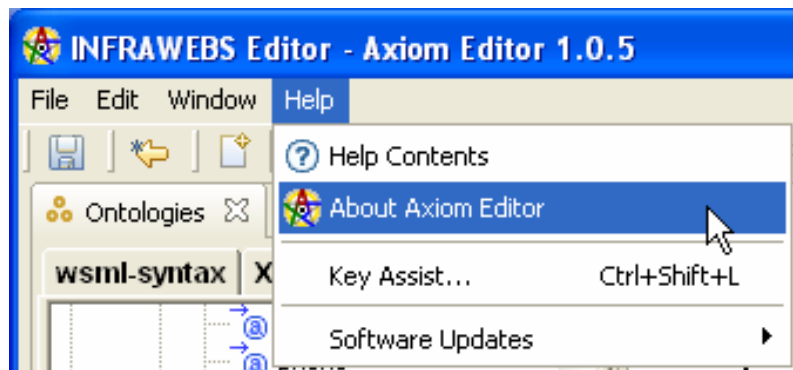
The *Window* menu can be used to change the Perspective or the Preferences of the Editor. There is one pre-defined perspective called “Axiom Editor” which is the default one.



The user can choose another perspective from the “Other...” menu item with the following perspective selection dialog:



And finally, the *Help* menu contains the About dialog of the Axiom Editor:



3 AN INFORMAL MODEL OF THE AXIOM CONSTRUCTION PROCESS

A process of axiom creation may be considered as a repetitive process combining of three main conceptual 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.

3.1 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 WSMML statement *?Concept memberOf Concept*, which means that the WSMML variable *?Concept* copying the structure of the *Concept* from a given WSMML 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 WSMML variables with type defined by the definition of such attributes in the corresponding ontology.

For example, let us assume that there is a concept of *reservationRequest* in a WSMML ontology defined as follows:

```
concept reservationRequest
nonFunctionalProperties
  dc#description hasValue "This concept represents a
  reservation request for some trip for a particular person"
endNonFunctionalProperties
reservationItem impliesType wsml#true
reservationHolder impliesType prs#person
```

The result of the definition step using this concept as “a template” will be creation of the following axiom model variable (WSMML logical expression):

```
?reservationRequest memberOf tr#reservationRequest
and reservationItem hasValue ?reservationItem
and reservationHolder hasValue ?reservationHolder
```

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.

3.2 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 value¹ 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 sub-concept 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 sub-concepts;
- 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.

3.3 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 two 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

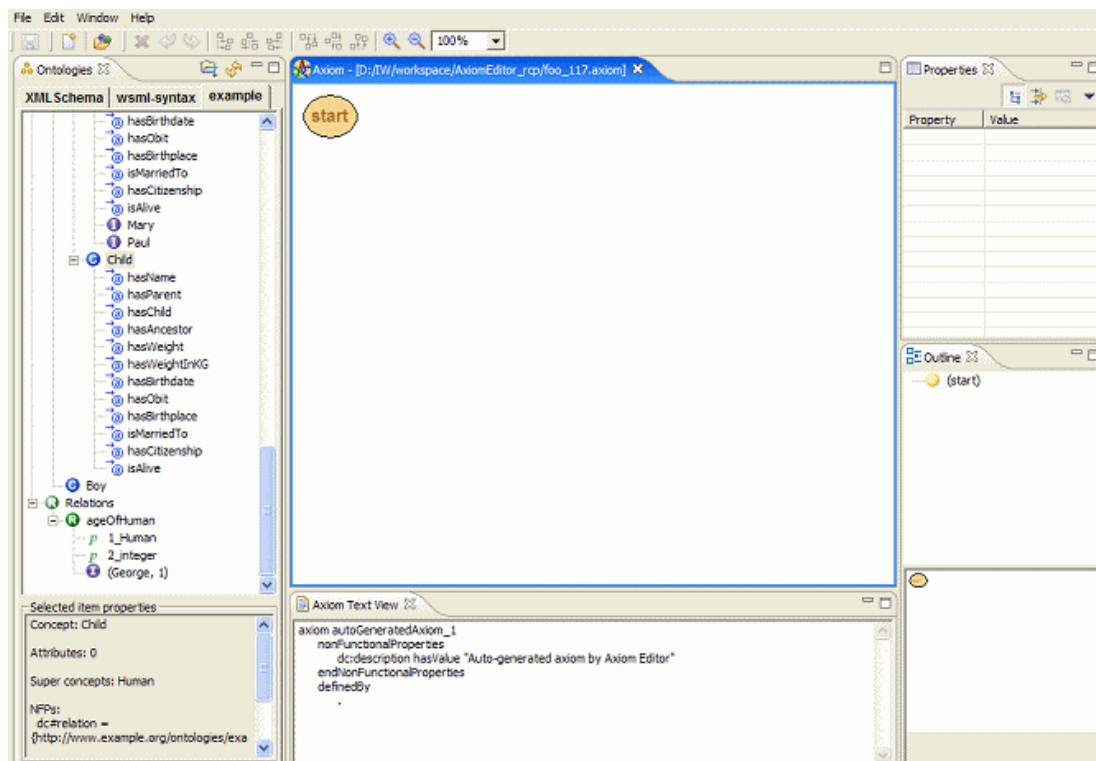
¹ The same rules are applicable to every unbound relation parameter.

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.

4 USING AXIOM EDITOR

In this section we will introduce the tasks a user can perform in Axiom Editor. After the application has been started, the workspace is initialized with an empty axiom. The diagram contains only the **start** element which is not a real element of the axiom but only a pointer to the root of the expression which the user is going to build. The Text View contains the declaration of an axiom with in empty definition.

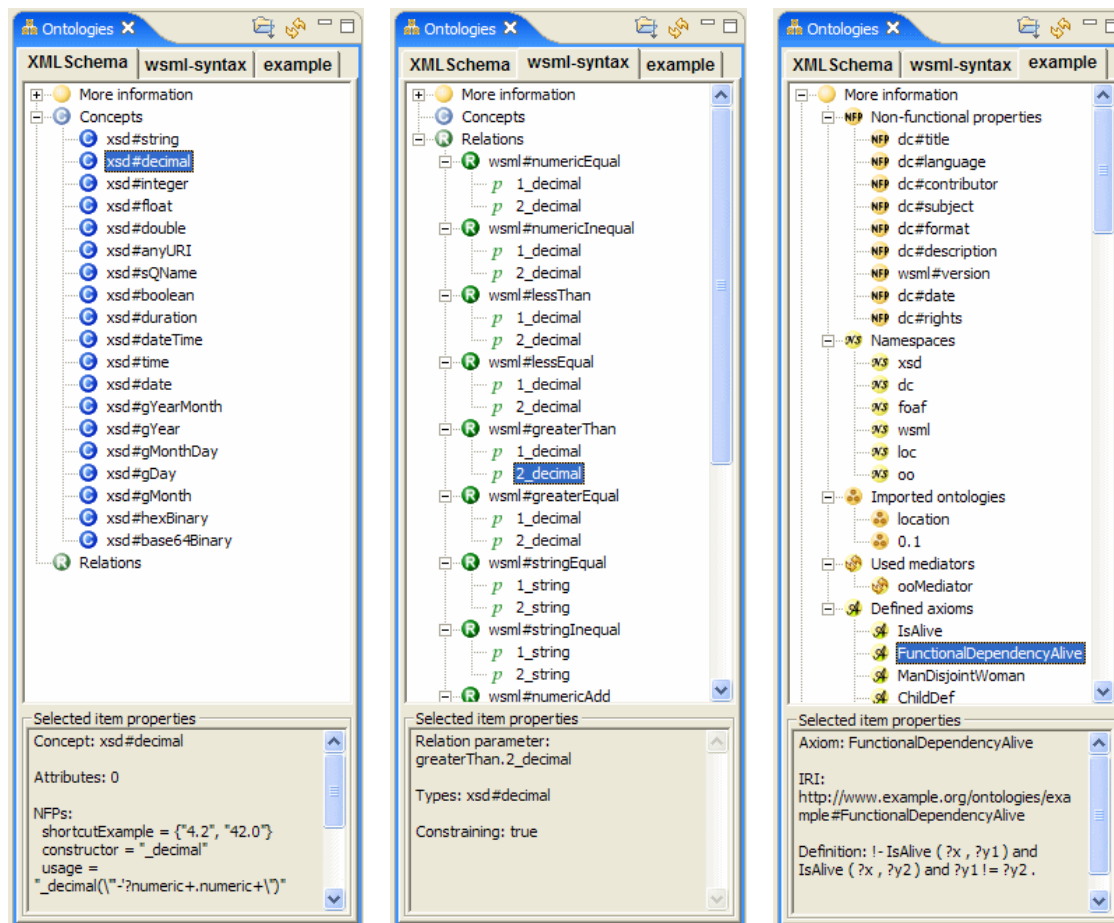


4.1 Loading ontologies

Ontologies are the main input for Axiom Editor. Usually the creation of an axiom involves using concepts from several ontologies as well as extensive usage of built-in WSM data types and predicates (XML data types are directly mapped to WSM data types). To unify the access to concepts, relations and WSM built-in constructs, the INFRAWEBS Axiom Editor treats the latter as regular concepts and relations. Two pseudo-ontologies are automatically loaded upon startup – one containing the

XML Schema data types defined at <http://www.w3.org/2001/XMLSchema> and the second containing the rest of the constructs defined at <http://www.wsmo.org/wsmo-syntax>. Thus the user can use basic data types (strings, numeric values etc.) as ordinary ontology concepts built-in predicates (such as numeric comparisons) as ordinary ontology relations.

The user can view an additional, non-logical ontology content by expanding the **More information** node at the top of the Ontology Tree and browsing the elements. Details are displayed in the Ontology Properties section.



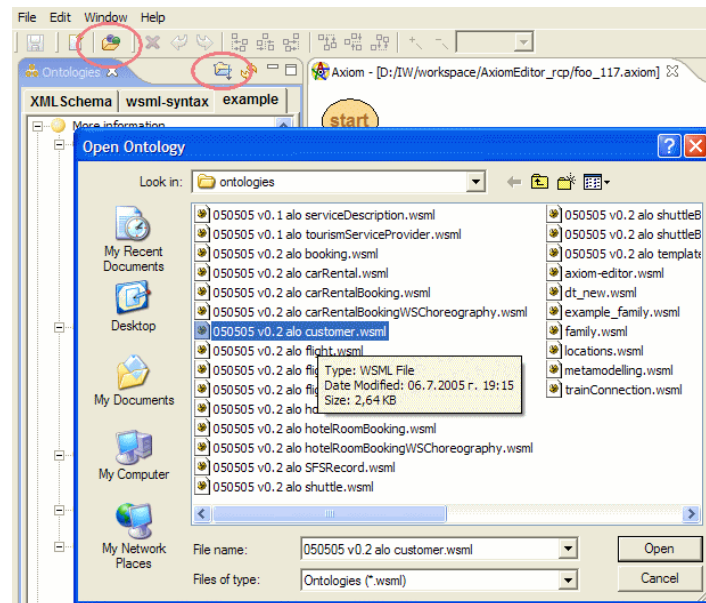
Built-in data types

Built-in relations

Additional content

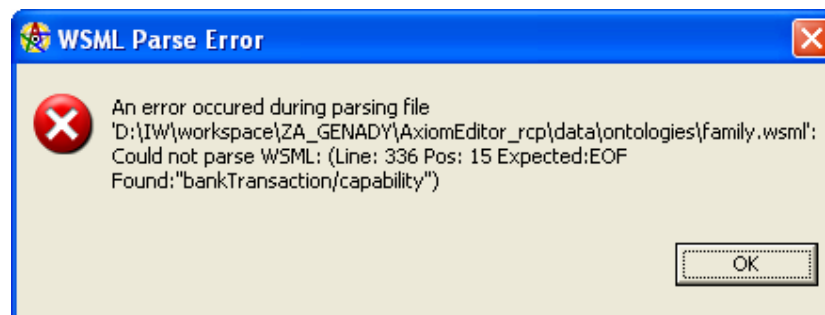
In order to load an ontology the user can press one of the Open buttons. The upper one is global and is used to load any file – a previously stored *.axiom file to be edited in the Diagram Area, a *.wsmo file containing an ontology to be loaded into the Ontology View or just a text file which will be opened for editing in a plain text editor.

The lower button is a part of the Ontology View and is intended to load ontologies only.

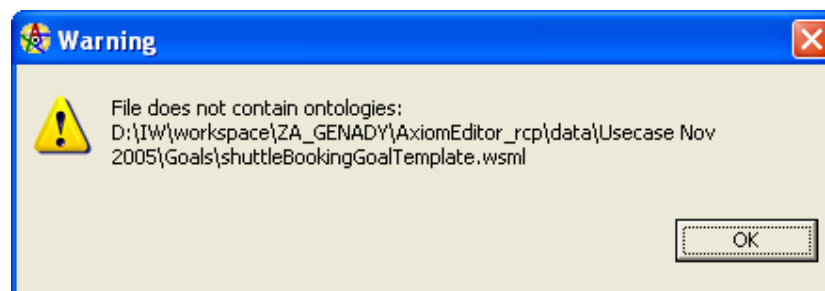


The user selects a wsml file. All ontologies described in that file (may be more than one) are extracted and added to the Ontology View. If the file is not a valid WSMML file a corresponding parse error is displayed.

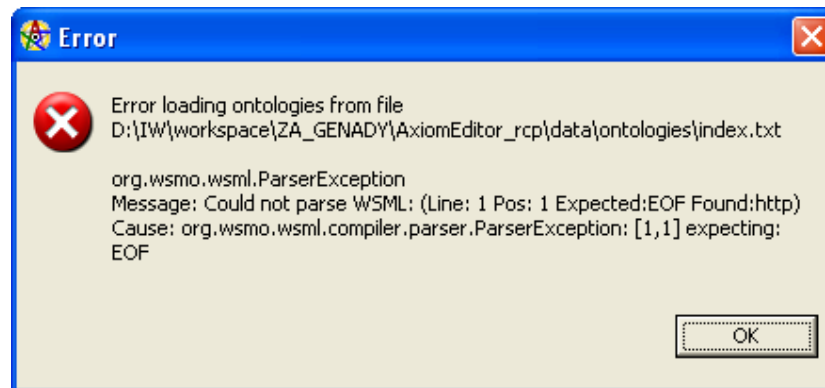
If an error occurs while loading the selected ontology, an appropriate error message appears with explanation of the error. For example, if an error was reported by the WSMML parser, the following error may appear:



If the selected WSMML file does not contain any ontologies, the following warning message appears:



If the selected file does not contain a proper WSMML format, another error message appears:



4.2 Imported ontologies

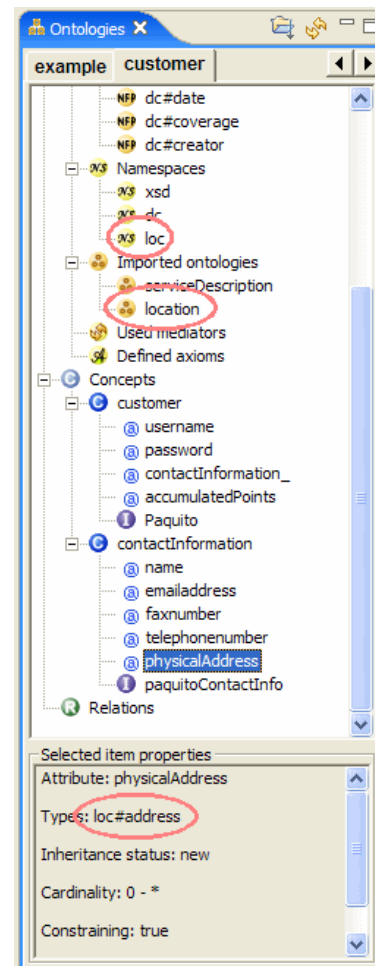
Suppose we want to refine the *physicalAddress* property of the *customerContact* variable. We could first locate it in the Ontology View to examine its details in the Ontology Properties section (using the "Locate in Ontology" operation from the context menu).

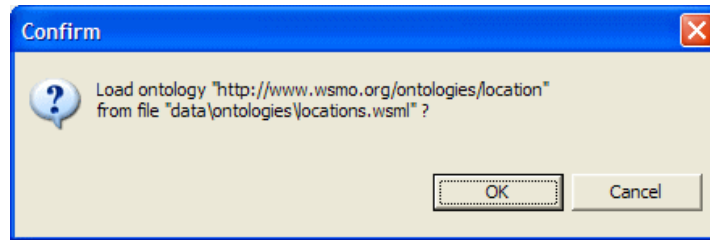
The type of the attribute is *loc#address*. The prefix *loc#* means that the concept *location* is not defined in the default namespace but in another namespace whose identifier is abbreviated to the word *loc*.

The abbreviated identifiers are called namespace prefixes. They can be found in the *Namespaces* branch of the *More information* section of the Ontology Tree. Selecting the node *loc* would reveal that the full identifier of the namespace is <http://www.wsmo.org/ontologies/location>.

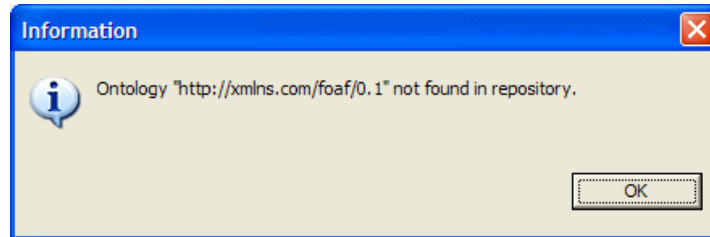
A *location* ontology is also listed in the **Imported ontologies** branch. Its identifier is the same as that of the namespace, which means that the *loc#address* concept is defined in the imported ontology <http://www.wsmo.org/ontologies/location>.

Double-clicking on an attribute navigates to its default type. Since the type of the *physicalAddress* attribute is not available, the ontology that defines it must be loaded. Axiom Editor uses an internal index to search its repository of available ontologies. When it finds the definition of an ontology, the following confirmation message appears.





If the ontology is not found, an error message is displayed and the construction of the axiom cannot continue in this direction:



After the user confirms, the *location* ontology is loaded and displayed in the Ontology View. The concept *address* is automatically highlighted.

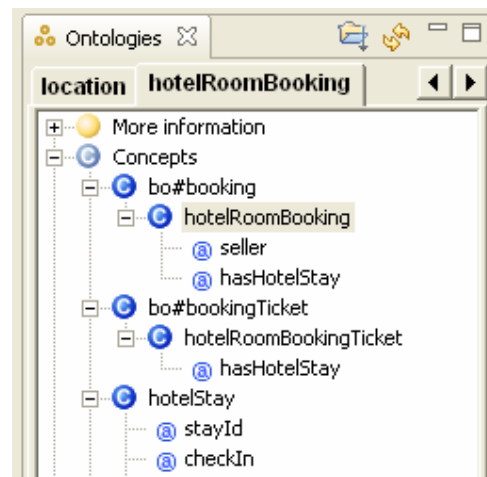
All elements from the *location* ontology can now be used for axiom construction. The *physicalAddress* of the *customerContact* variable can now be refined by a *loc#address* variable.

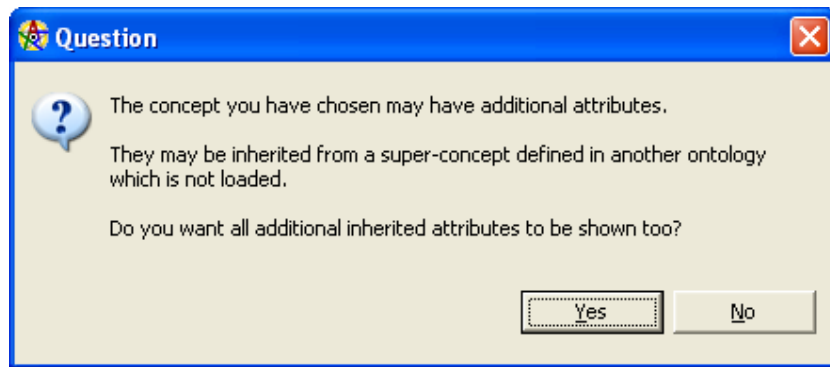
4.3 On-demand loading

The INFRAWEBS Axiom Editors has a built-in mechanism for on-demand ontology loading. This means the user does not necessarily have to load all required ontologies in the beginning. Instead, every required ontology will be loaded later at the time it is needed with the on-demand loading.

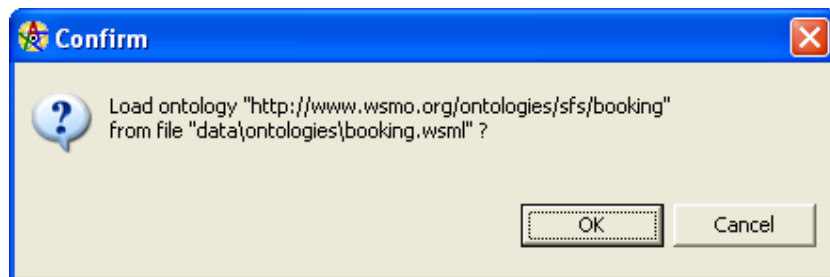
Let's illustrate this with an example. Let's say we have already opened the ontology "hotelRoomBooking" in Ontology View:

If we try to create a variable with type, equal to the concept "hotelRoomBooking", the following message appears:

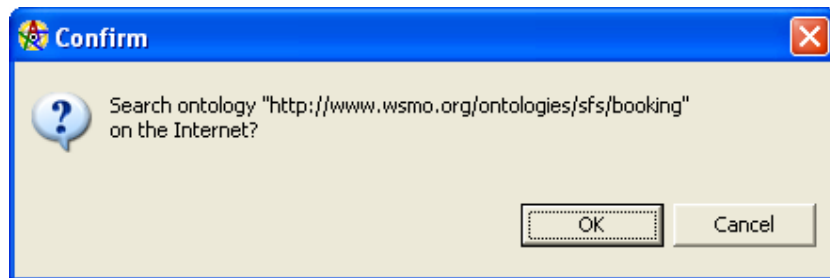




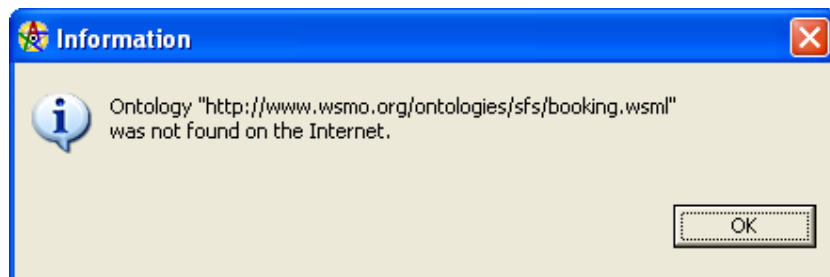
In this particular case this message is caused by the “bo#booking” super-concept of the concept “hotelRoomBooking” because it is defined in another ontology (booking) which is not loaded at the moment. The on-demand mechanism invokes and asks the user whether he would like to load the “booking” ontology:



Axiom Editor firstly tries to locate the ontology in its own ontology store. If it cannot find the ontology there it asks the user for permission to search for it on the Internet:



If it cannot find it in Internet, then the following message appears:

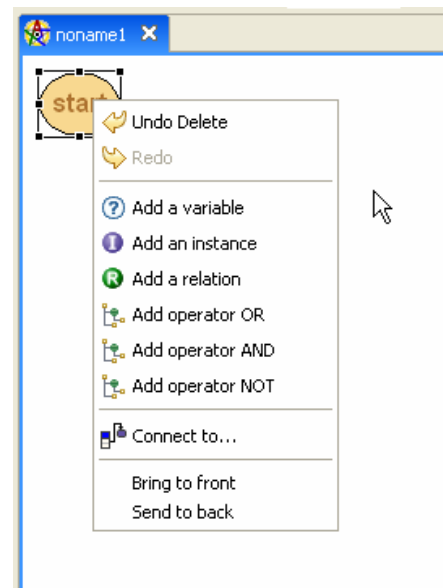


If the ontology is successfully loaded, then all additional attributes that might be inherited from the super-concepts will be loaded and added to the already present attributes of the selected concept.

4.4 Menu for operations available at the axiom definition step

At the first phase the axiom model contains only a “start” element. The menu for available operations allows the following:

- Adding a variable / instance / relation / operator to the model.
- Undo/Redo operations.
- Manual connecting elements with the “Connect to...” command.
- Changing the Z-order of the elements in the diagram in order to avoid overlapping of important information.



4.5 Two modes for axiom construction

The main concern of the INFRAWEBS 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.

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 modeling area, which can be later combined in various semantically correct ways. This allows advanced users to be more efficient.

The disconnected elements in the axiom model, the ability to reconnect elements and to use the “Connect to...” operation belong to the *Advanced mode*. In the current version of the INFRAWEBS Axiom Editor it is possible to work with the advanced mode operations at all times.

4.6 The first variable

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".

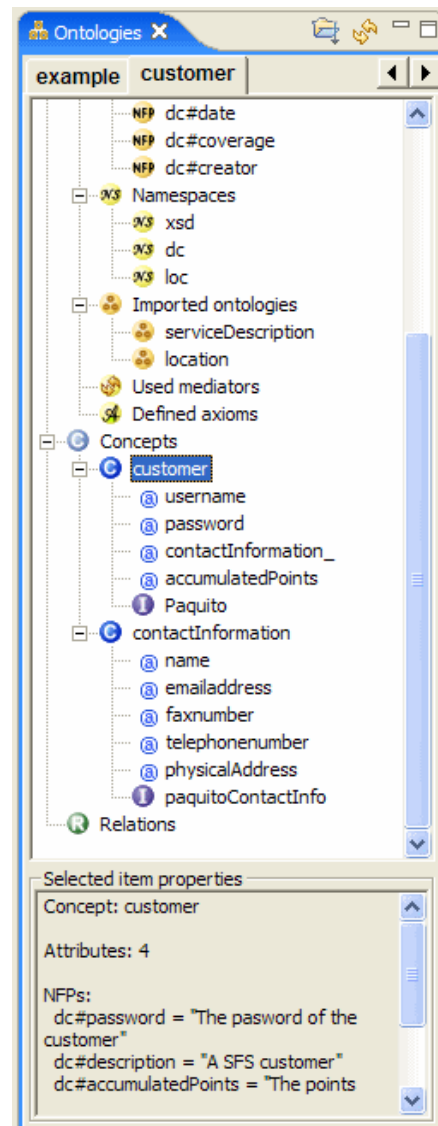
From this moment on, the construction process continues by performing semantically-correct operations on different elements in the axiom model which can be: variables, variable attributes, instances, connections, operators, relations and relation parameters.

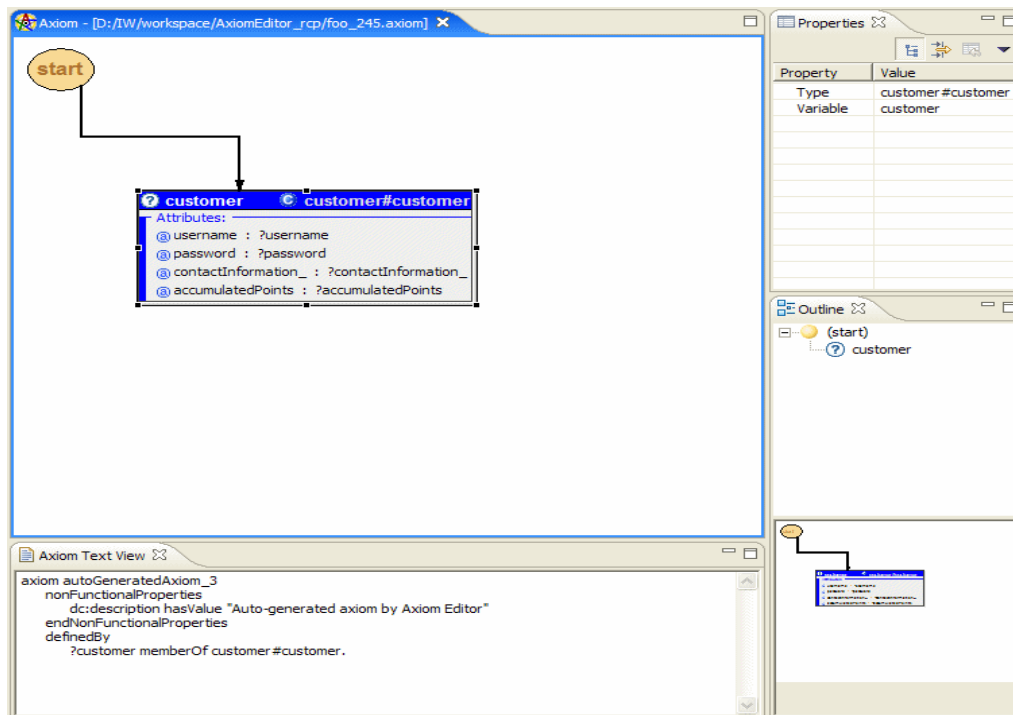
After the ontology "*customer*" has been loaded, the user selects a concept (in this case – the concept "*customer*") from the Ontology Tree. Details about the concept are displayed in the Ontology Properties section below.

The user can double-click the concept or drag-and-drop it to the Diagram Area. Thus a variable is created in the diagram which is of type *customer*. The *start* element is automatically connected to the variable making it the root of the expression. The Text View now contains a declaration of a *customer* variable.

The difference between double-clicking and drag-and-dropping is that with drag-and-dropping the user can specify the exact location of the new element in the diagram immediately. Of course, at any point in time the location of elements in the diagram can be changed at the will of the user.

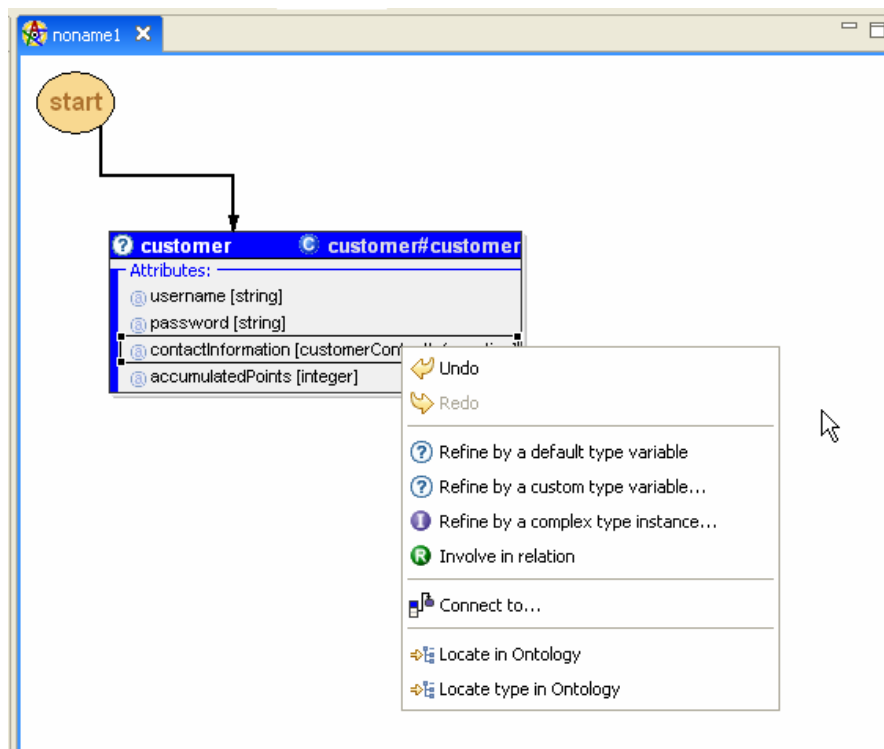
The concept's attributes are visible in the Diagram Area. They can be selected and refined. Here is how the axiom model would look like at this moment:





4.7 Refining attributes by selection

The most common task during the building of an axiom is the refinement of attribute values.

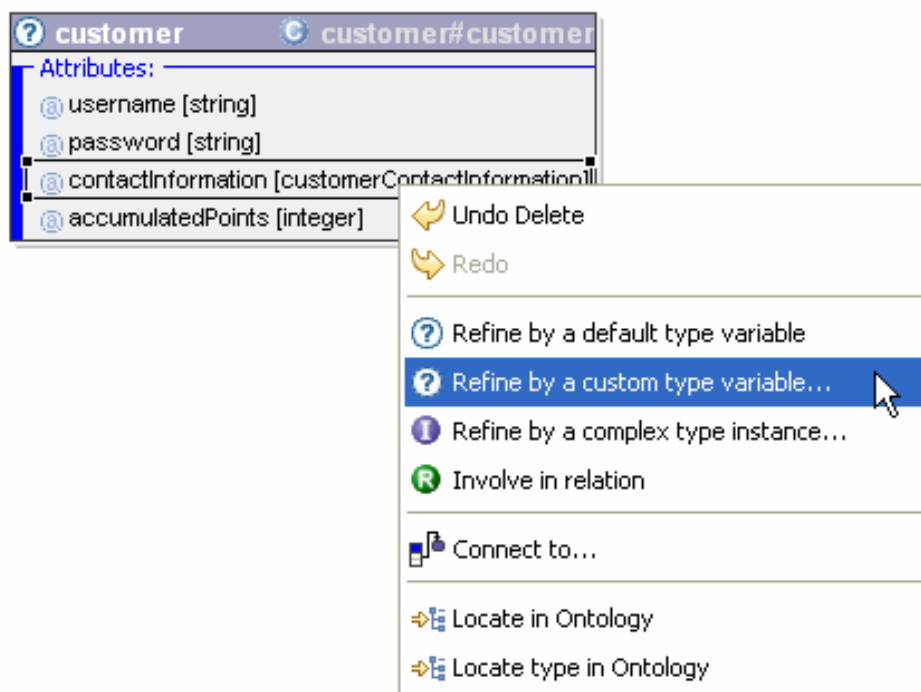
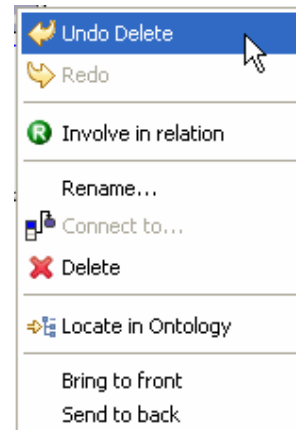


The user must select an attribute from a variable and press the right mouse button to invoke the context menu. The context menu gives access to all operations which can be applied to the selected axiom element as well as some global operations. For attributes, it contains the global Undo/Redo operations, the common Connect-to operation, a set of specific refinement operations, and a couple of ontology navigation operations. All of them will be discussed later.

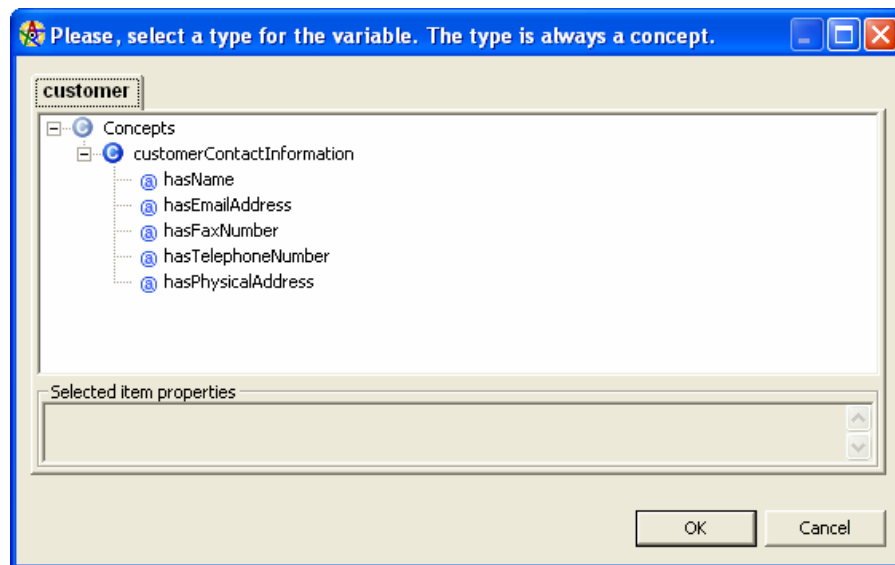
The INFRAWEBs Axiom Editor maintains a stack of all operations since the beginning of the creation process (or the moment the axiom was loaded). Operations can be undone by selecting the Undo operation from the context menu or from the toolbar at the top. The keyboard shortcut for this operation is Ctrl-Z.

If the user selected *Undo* at this moment, the customer variable would be deleted. The variable could be restored by selecting *Redo* in a similar way.

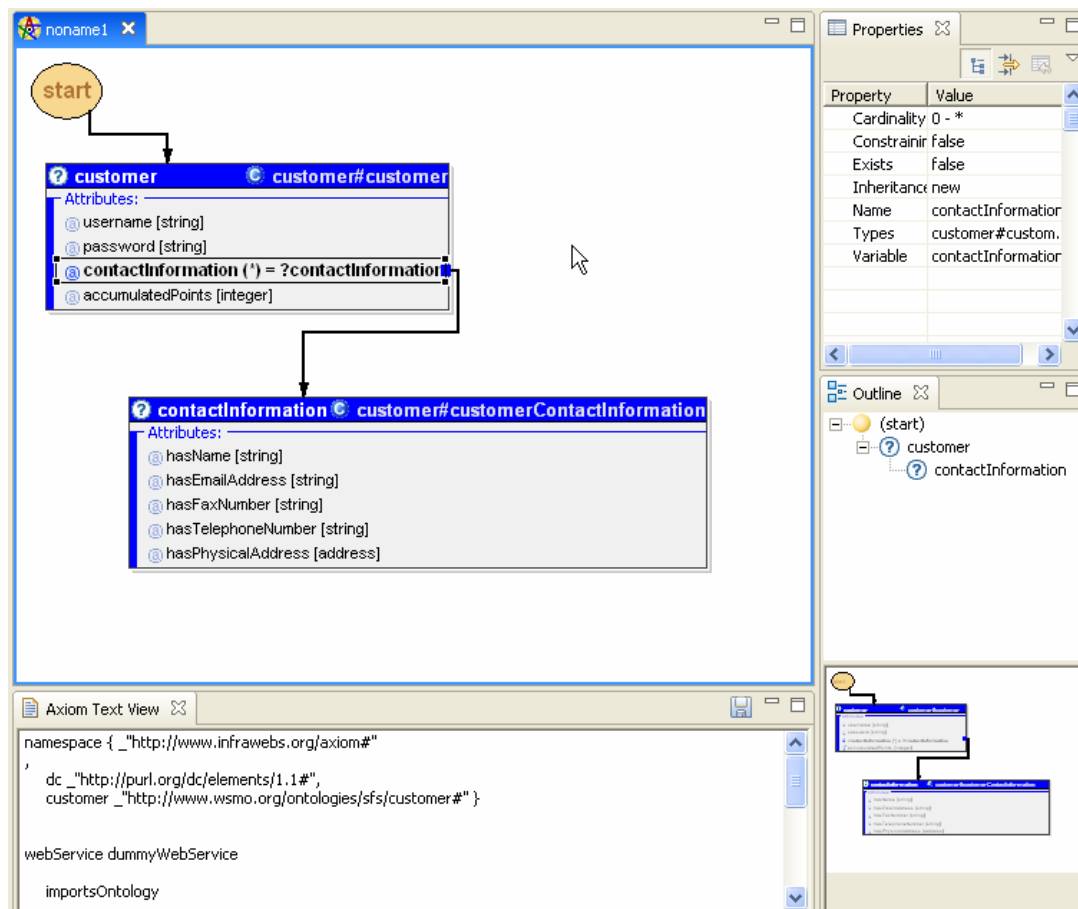
The goal is however to refine the value of the *contactInformation* attribute. A set of refinement options is available. The “**Refine by a custom type variable...**” allows the user to select the type for a new variable which will be the value of the attribute. A dialog appears, containing a subset of the ontology view. It allows the user to select only concepts. What's more, it only shows these concepts, which are compatible by type with the type of the attribute. This includes concepts that are sub-concepts of the concept defined as a default type of the attribute.



In our example, there is only one concept compatible with the attribute's type and it is the *contactInformation* concept from the same ontology. Its attributes are displayed only for convenience. They cannot be selected as refining elements.



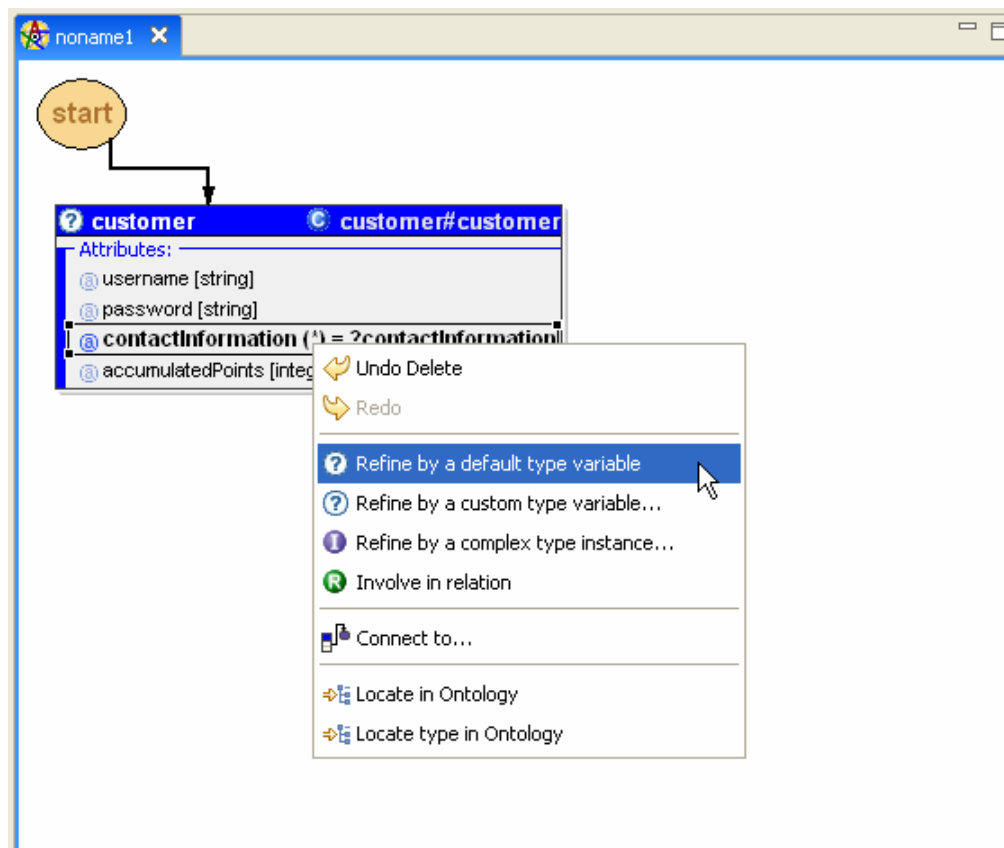
After the user has selected this concept and pressed the Ok button the result is as follows:



A new variable of type *contactInformation* is added to all views – the Diagram Area, the Outline View, the Thumbnail and the Text View. The attribute is connected to the new variable showing the refinement dependency between them. The name of the new variable is the same as the name of the associated attribute extended by sign “?”.

4.8 Refining attributes by default

There are two more ways to achieve the same effect. The first is to select option **“Refine by a variable with default type...”** from the context menu:

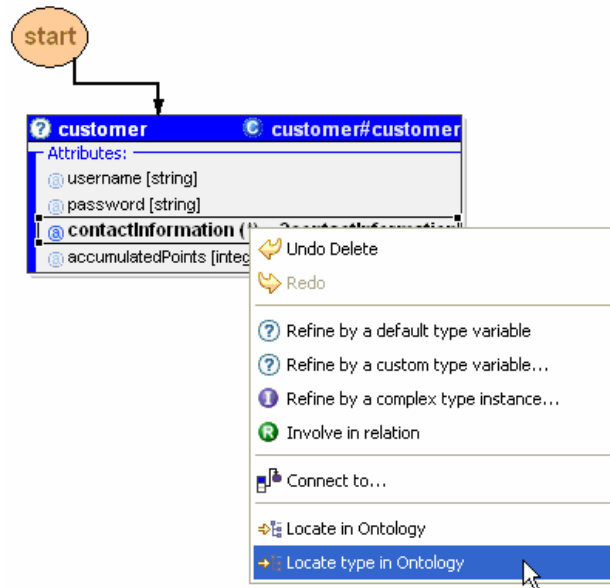


The selection dialog would have been skipped and a variable having attribute's default type would have been added and connected to the axiom model. This approach is quicker but the user wouldn't be able to select a sub-concept.

4.9 Refining attributes by manual connection

This approach involves advanced editing. The user might want to examine the attribute's type more closely before selecting an appropriate concept.

The navigation from an element to its type is done by selecting “**Locate type in Ontology**” option from the context menu.



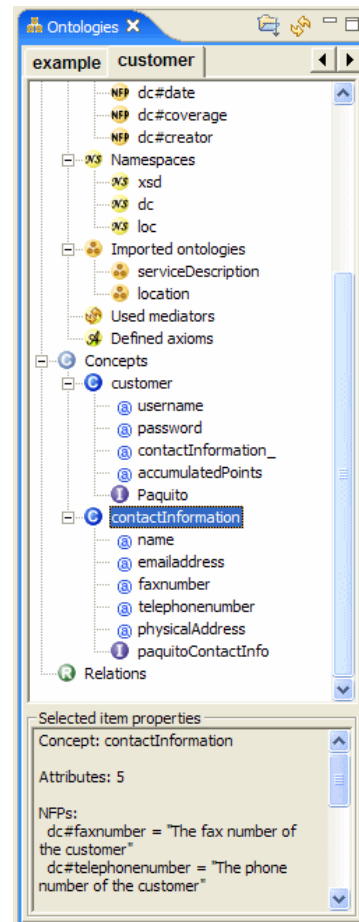
The *contactInformation* concept is highlighted in the Ontology Tree. The user can then select a nearby sub-concept or instance which will also be compatible.

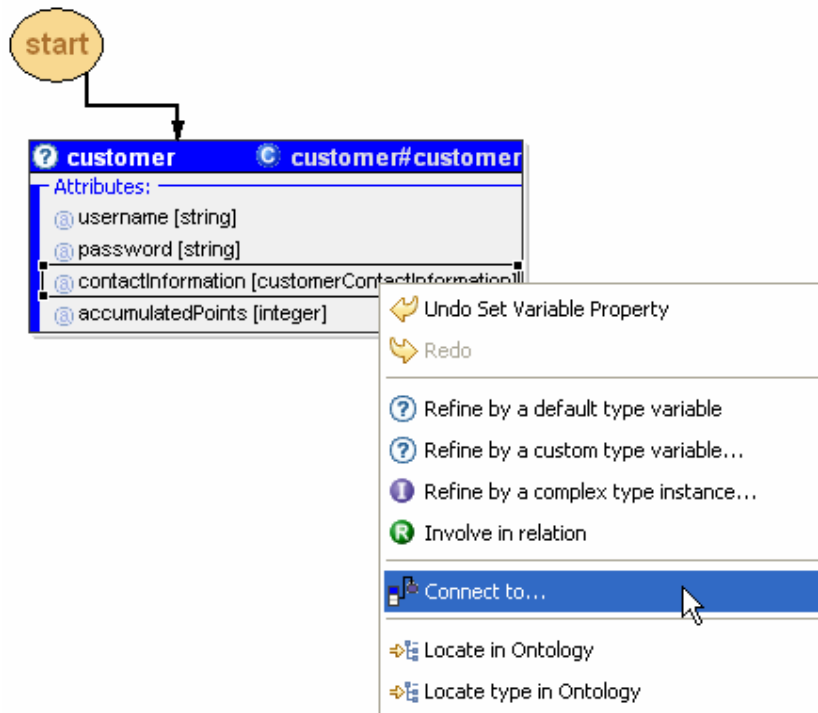
Similarly, the “**Locate in Ontology**” operation would highlight the attribute itself rather than its type.

Double-clicking on the concept adds a new variable of type *contactInformation* to the Diagram Area which is not connected to any part of the axiom. It is not included in the Outline View and is not declared in the Text View.

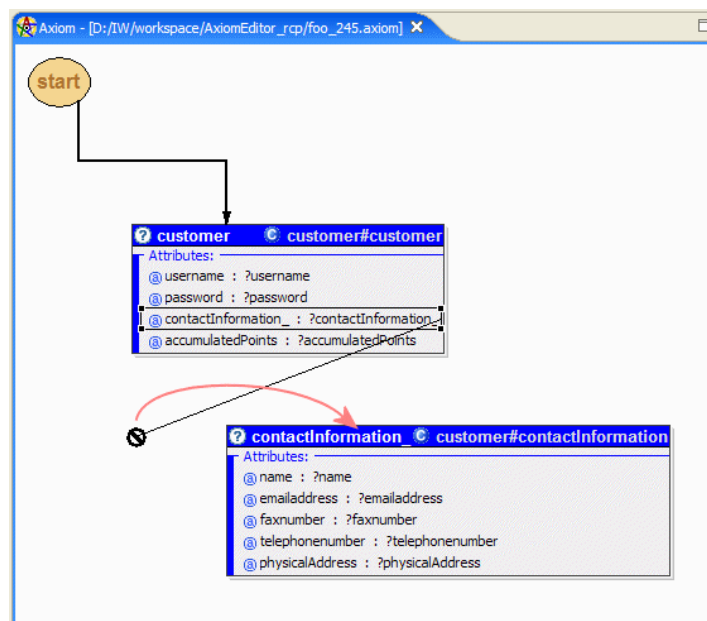
This time, the connection must be established manually. The user has to right-click on the *contactInformation* attribute and to select “**Connect to...**” from the context menu.

The first end of the connection is attached to the attribute while the second end is left free. The user has to point at a compatible variable or instance from the Diagram Area which is not yet connected to the axiom. Once again, the target element must be compatible with the type of the attribute.



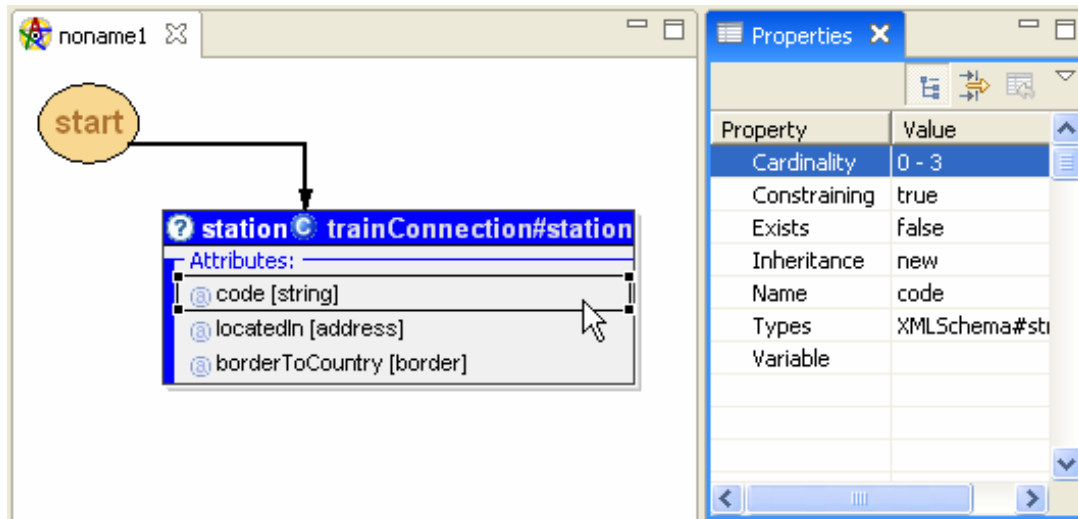


The mouse cursor changes its shape to indicate which elements are compatible and which are not. Once the target has been selected the connection looks the same as in the previous example.

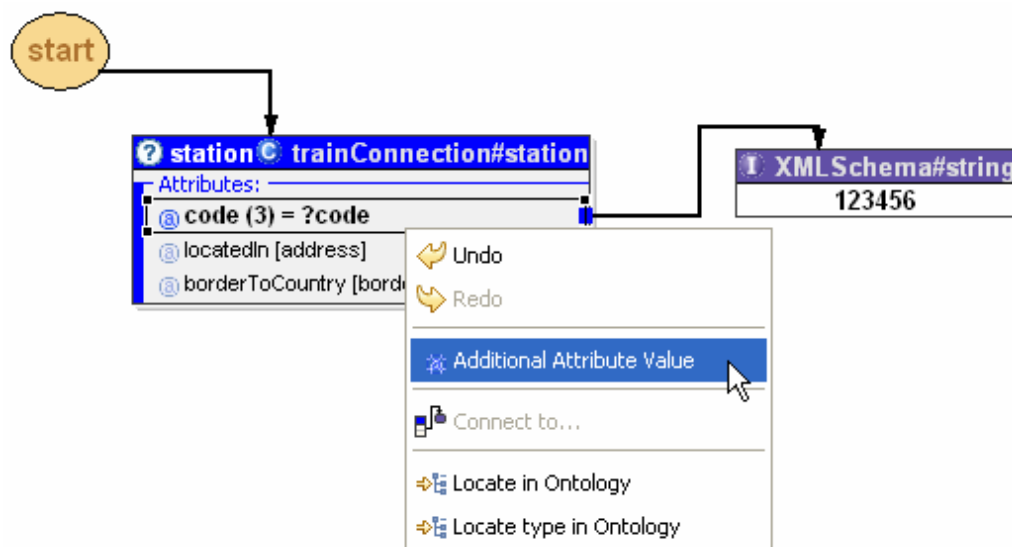


4.10 Multiple value attributes

The *Cardinality* property of an attribute shows the number of different values that can be assigned to this attribute. An attribute having a maximum cardinality greater than one is called a “multiple value attribute”. For example, the code attribute of the variable station has a maximum cardinality of 3.



Let's say that we have already refined this attribute with the value instance 123456. Now we would like to assign additional value to this attribute. This is done with the operation “Additional Attribute Value” from the context menu:

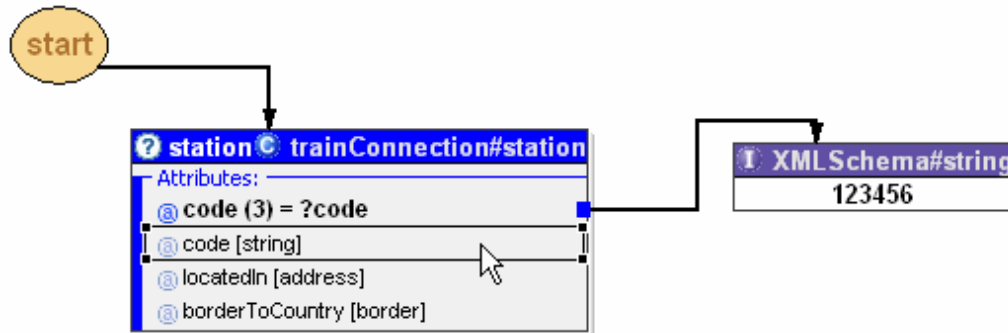


This operation is only available if the following requirements are met:

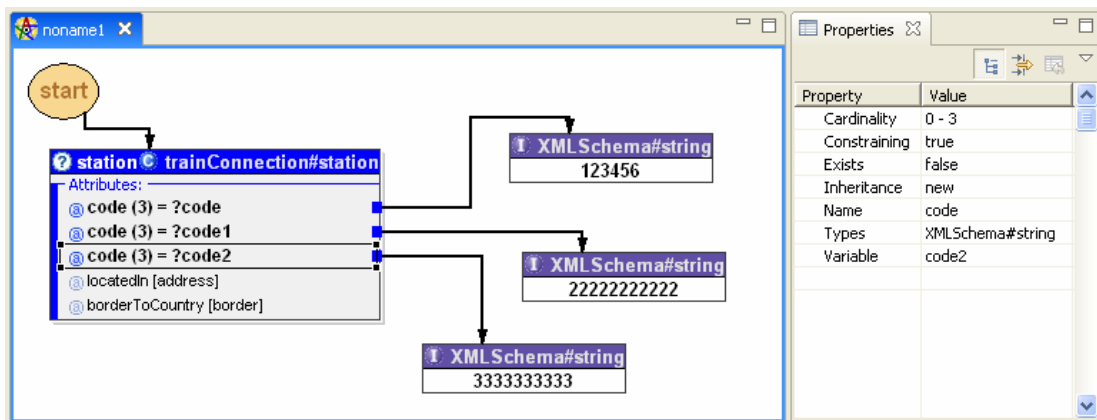
- the attribute is a multi value attribute

- the attribute is already refined
- the maximum cardinality for this attribute has not yet been reached

After executing this operation we have additional copy of the attribute which can be refined in all possible ways just like the first attribute:

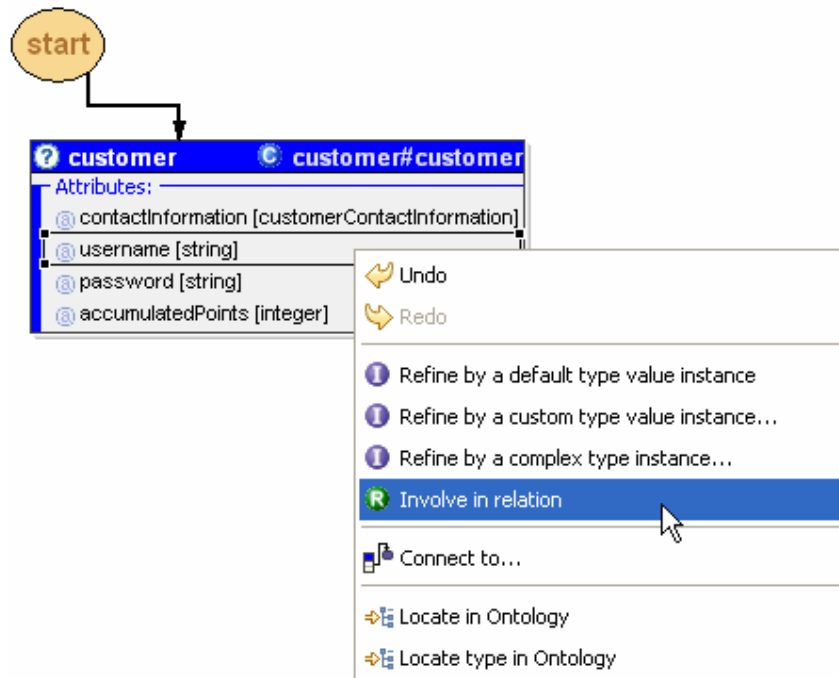


The number in brackets (3) shows the maximum cardinality of the attribute and it only appears in multi value attributes. Here is an example, showing how we could assign 3 different values to the multiple value attribute *code*.

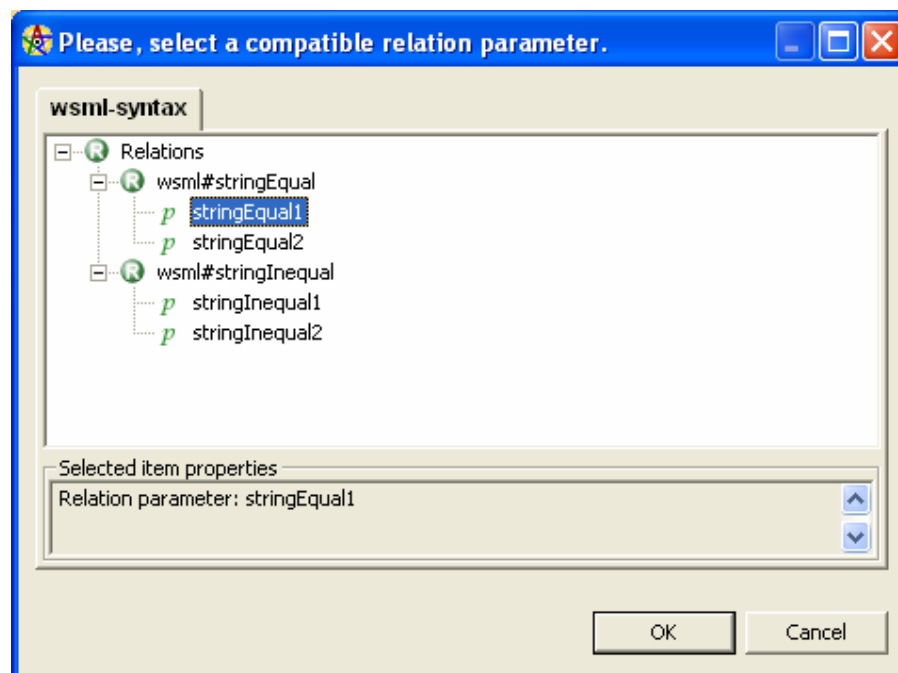



4.11 Use of relations

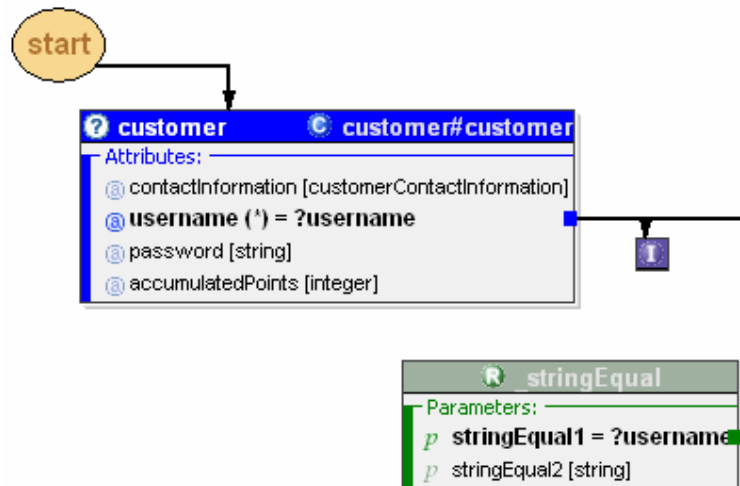
Another way to refine a variable is to involve it in relation. For instance, we can refine the “username” attribute by involving it in a “stringEqual” relation. This is done by selecting the “**Involve in relation**” option:



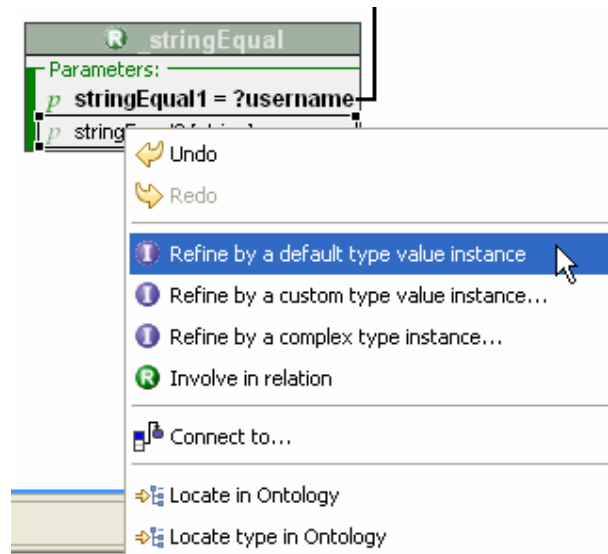
A relation selection dialog appears. The user must select the concrete parameter of the chosen relation to which the selected attribute should correspond:



The new relation appears in the model. Please note the small box , in the middle of the connection between the argument of the variable and the parameter of the relation. This is actually a value instance of type *_string*.



The parameters of a relation can be refined in the very same way as in the case of refining attributes of variables. Here is how the second parameter can be refined by a default type value instance:



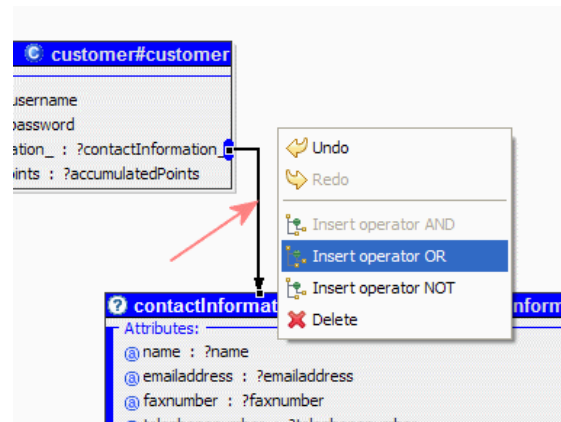
The manipulation of relations and their parameters is analogous to the manipulation of variables and their attributes respectively. Relation parameters also have types and assigned variable names. They can be refined in the same ways as attributes and the same consistency checks apply for them. The only difference between variables and relations is that a relation has no variable name and cannot refine an attribute by itself.

4.12 Use of logical operators

The current version of the INFRAWEBs Axiom Editor allows using the following logical operators: OR, AND, NOT and EXISTS. The EXISTS operator is performed by setting the “Exists” Boolean flag of a variable, which was described earlier. In this section we will describe the use of OR, AND and NOT operators.

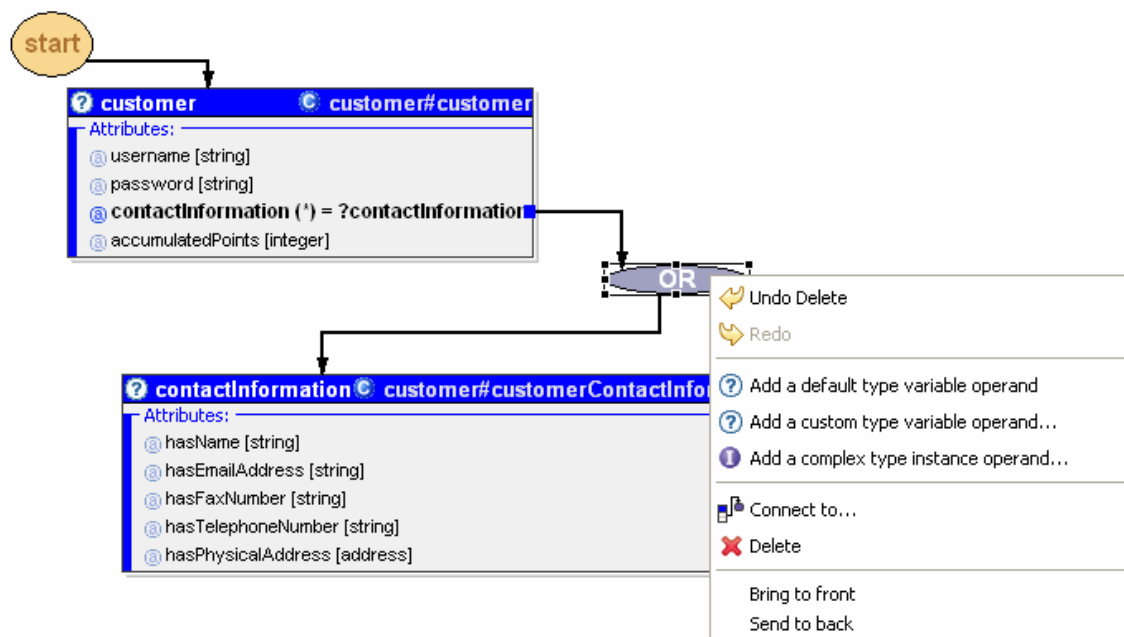
Operators can be inserted in the middle of existing connection to branch the expression logically.

Firstly, a connection must be selected. Secondly, the appropriate type of the operator must be selected from the context menu:

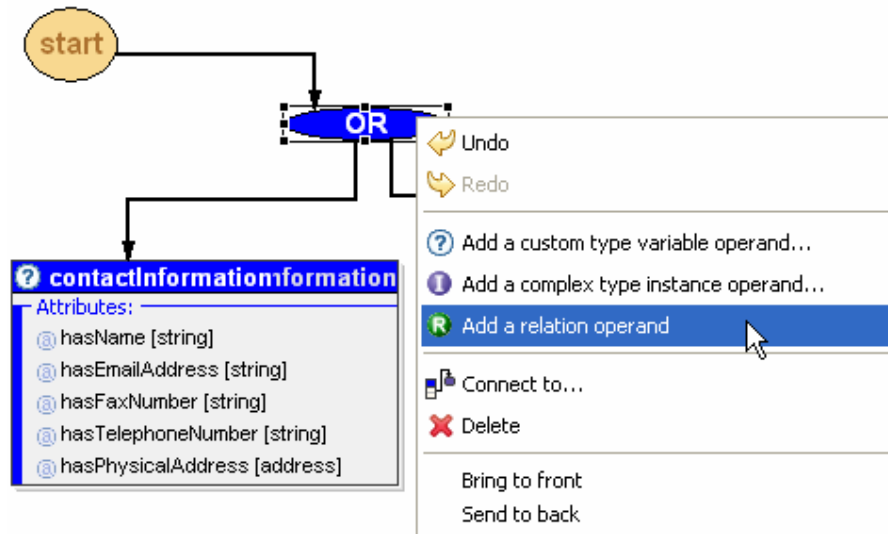


If it is an infix operator such as AND or OR, then a new variable or instance must be selected as its second operand.

If the operator used for refining a variable, then the following operations are available:

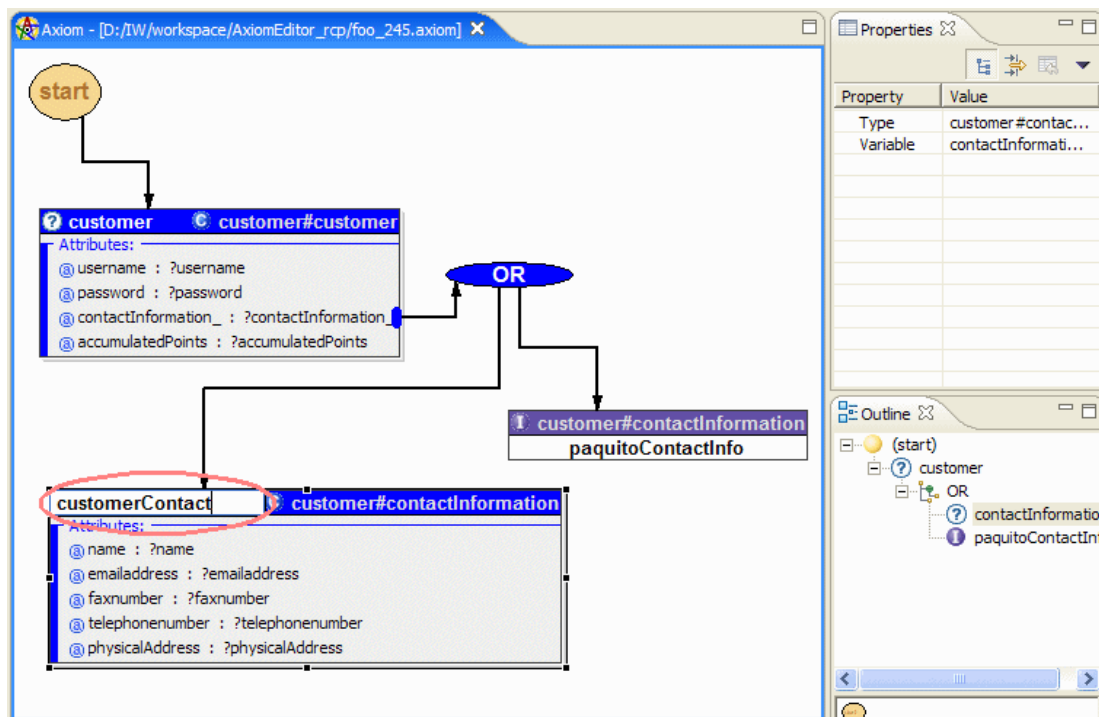


If the operator is not connected to a variable it is also possible to connect it to a relation, like this:



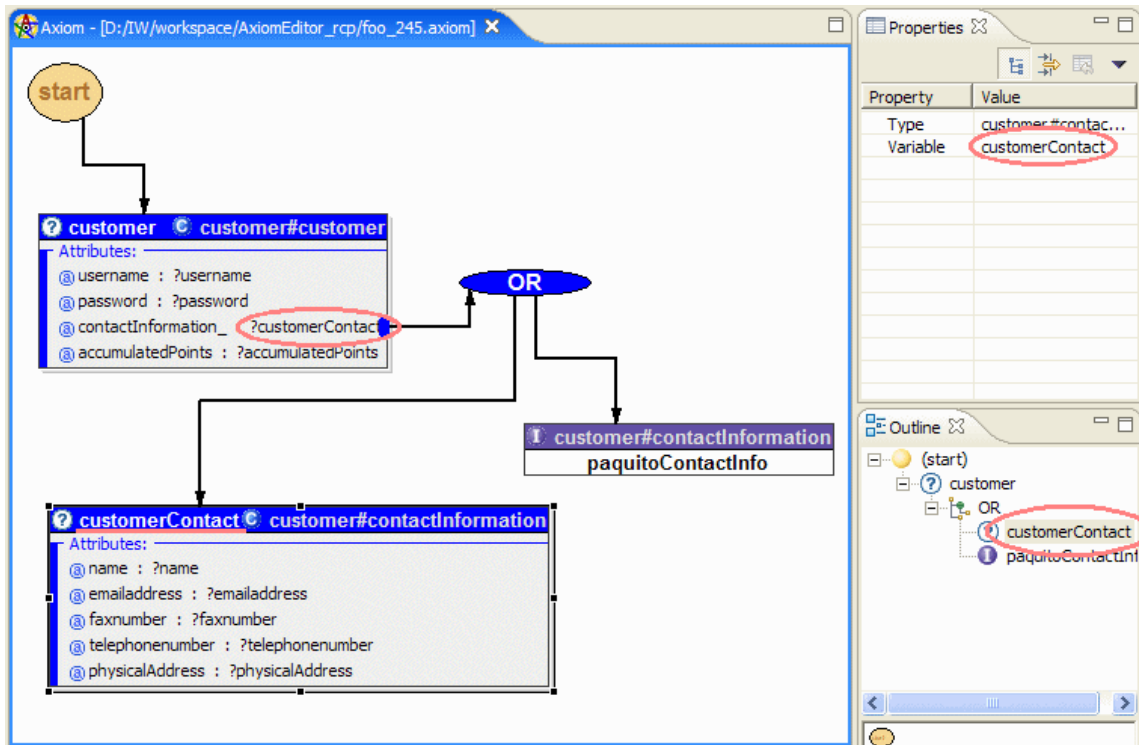
4.13 Renaming Variables

Variable names are automatically generated by the INFRAWEBS Axiom Editor and their uniqueness is observed during the whole construction process. However, advanced users might want to use their own, mnemonic variable names. The INFRAWEBS Axiom Editor allows this through the editable property called “Variable” in the Properties View. For convenience, a single click on a selected variable also opens a small editing area over the name of the variable displayed in the diagram. This is called “direct editing” as we know it from the Windows, Excel etc.



A single click on a selected variable enters direct edit mode.

After the variable is renamed from *contactInformation* to *customerContact* its new value is reflected everywhere in the workspace – all variables, instances and attributes from the Diagram Area bound to this variable name, the Properties View, the Outline View, and the Text View (which is not displayed here).



4.14 Operations for editing connections

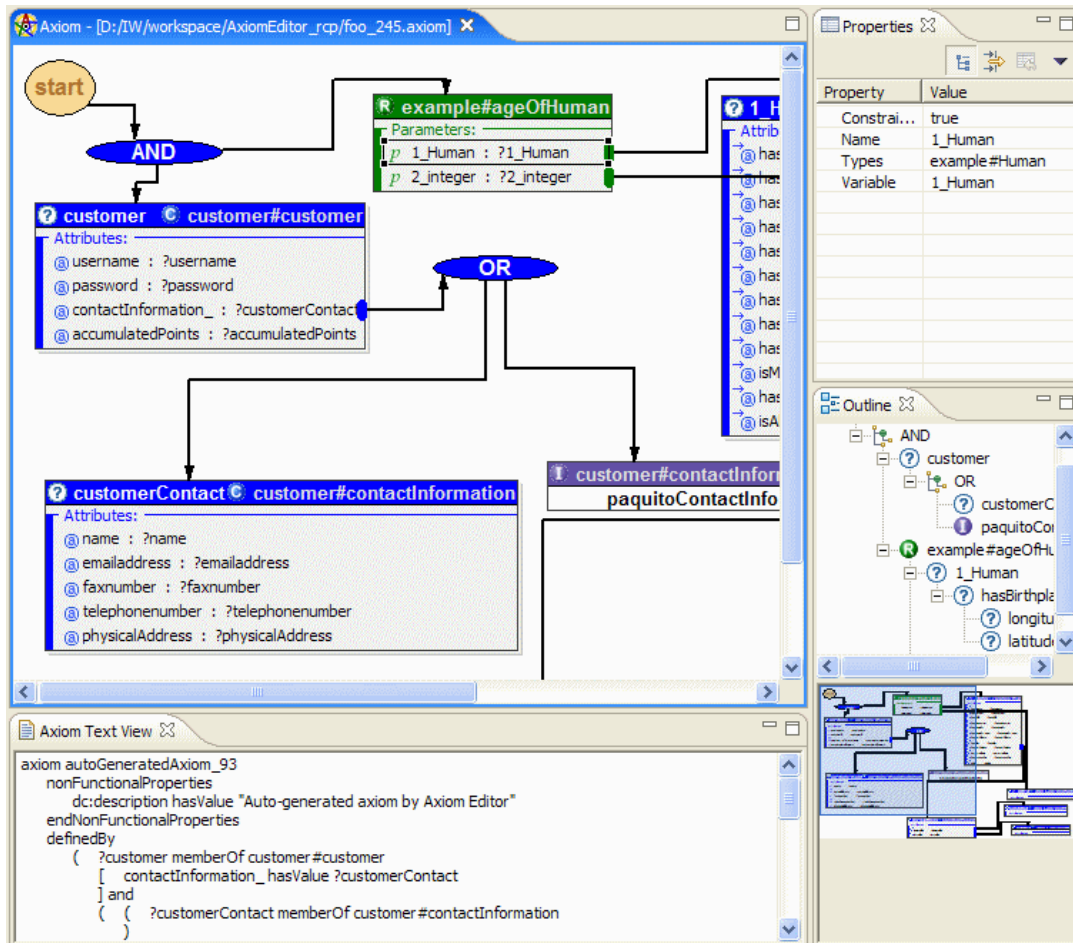
The advanced mode provides operations for editing connections. These operations include:

- reconnection of the source of an existing connection
- reconnection of the target of an existing connection

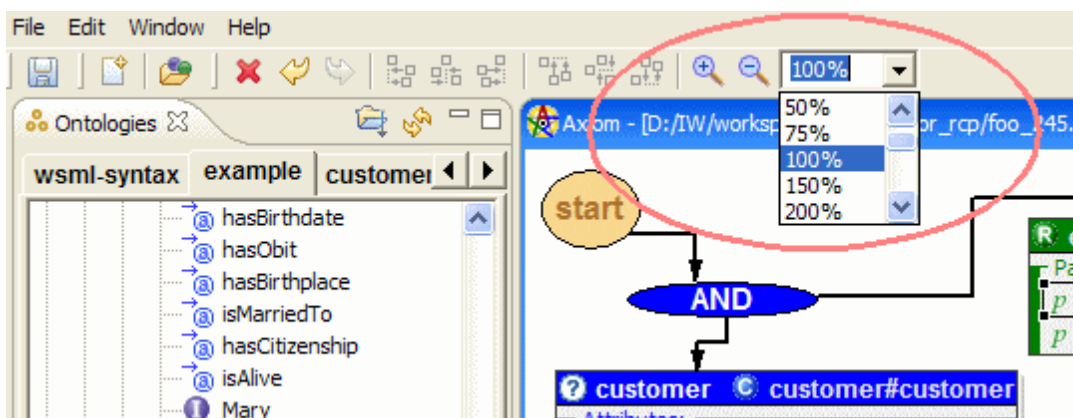
These operations are performed by drag-and-drop of the end-points of an existing connection.

4.15 Manipulating large axiom models

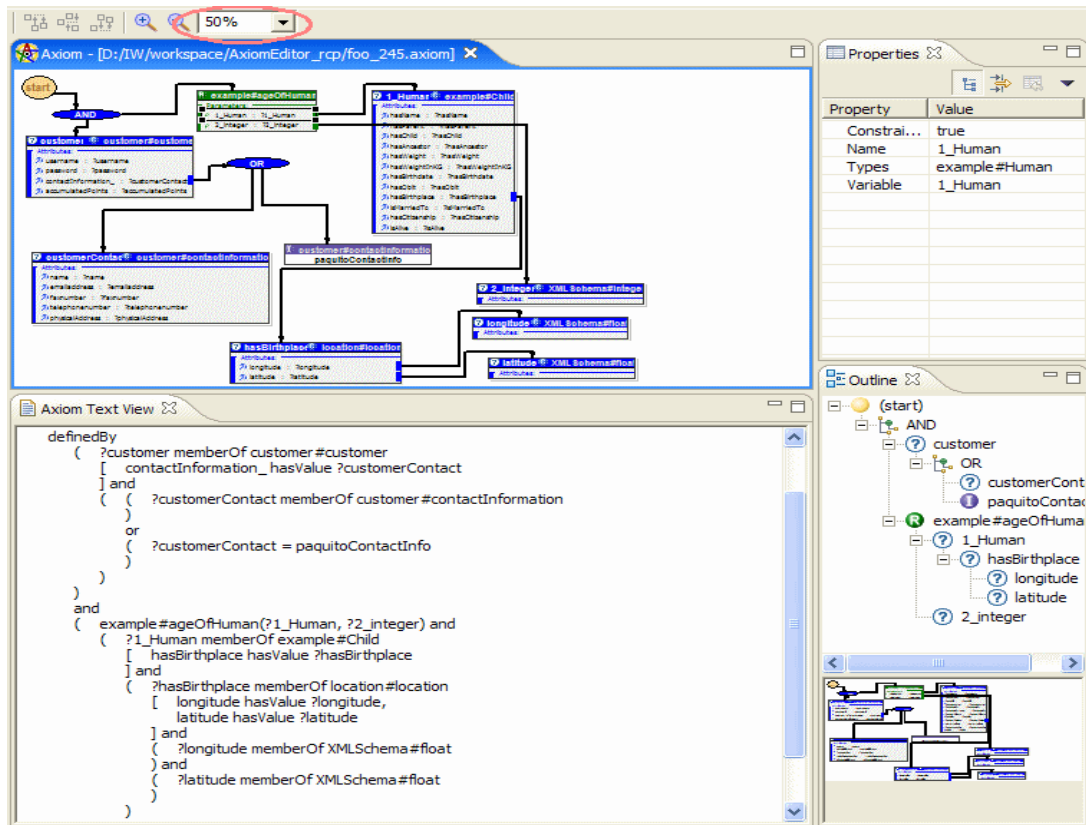
When the diagram grows large it becomes difficult to perceive its high-level structure and to navigate between the different parts. The Outline View and the Thumbnail can help in this situation. However it is often better to zoom out the diagram and take a more distant look at it.



Zooming is done by means of the zoom controls located at the top of the workspace.

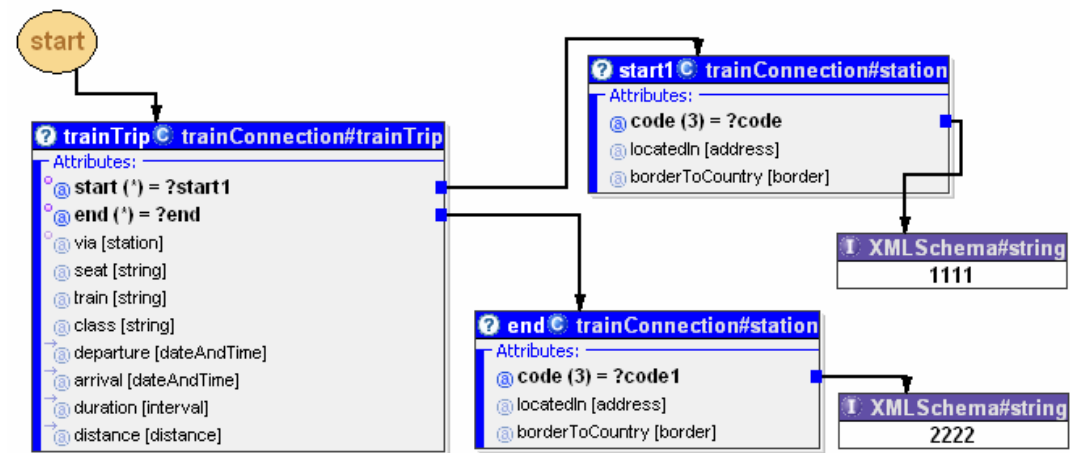


Selecting zoom factor of 50% and realigning the views would have a similar effect on the workspace:

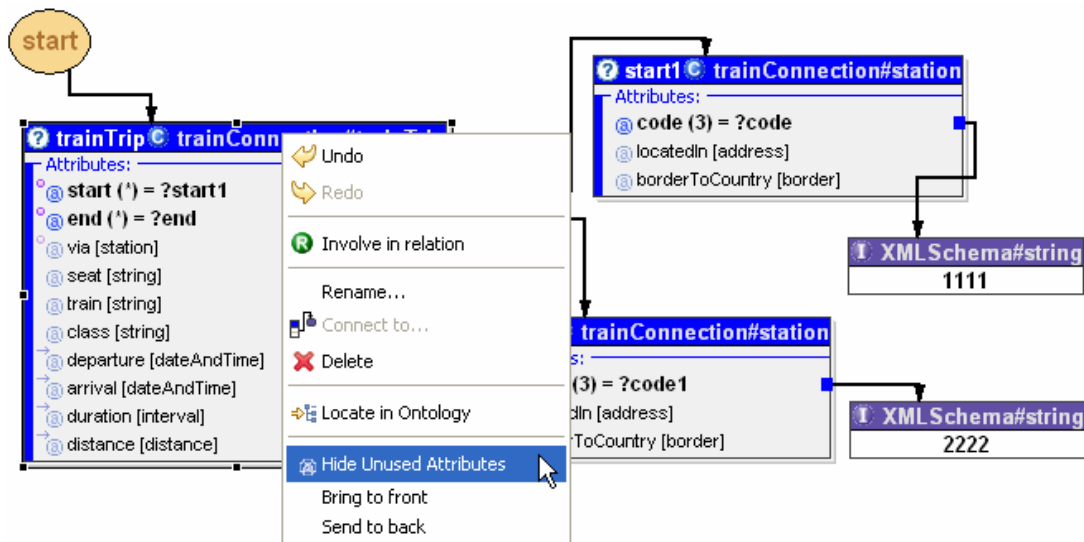


4.16 Hiding unused attributes

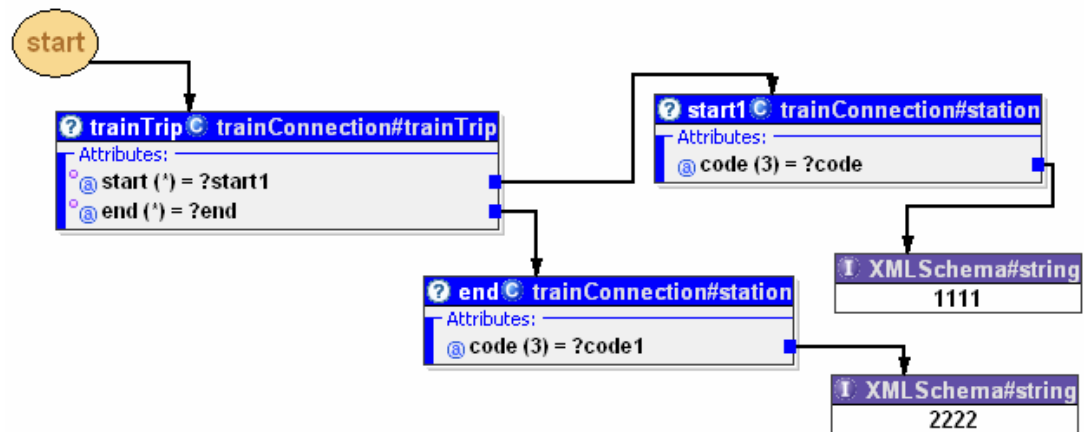
When we add variables to the axiom they are displayed with all available attributes. Since their number may be quite large, the area occupied by a single variable will also be large. In this example we have only 3 variables which occupy almost half of the model area:



Since we do not want to refine any other attributes of these variables, it would be convenient if we could minimize the area they occupy. There is a convenient operation for such cases called “Hide Unused Attributes”. It is applied on variables, like this:

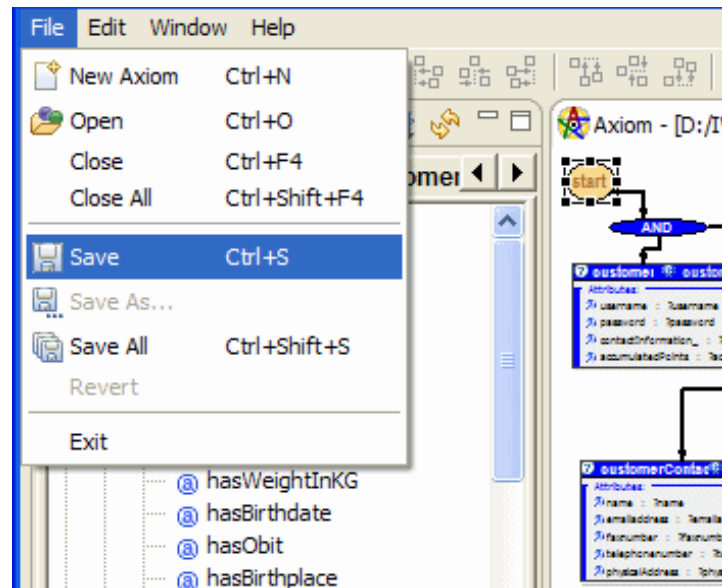


Here is the result of this command applied to all variables of the axiom:



4.17 Saving and Loading axioms

The user can select Save from the main menu and store the working progress or the complete axiom in a proprietary binary format. The file extension is “.axiom”. The saved axiom can later be opened by the Open command.



4.18 Saving and Loading axioms as wsml files

It is possible to store the constructed axiom as a wsml file by clicking on store (“disket”) icon located at the right of the Axiom Text View header. Such a file contains only text of the axiom written in WSML language. The stored text may be edited by all text editors or loaded again by the INFRAWEBs Axiom Editor.

The current version of the INFRAWEBs Axiom Editor allows automatically restoring a graphical model of an axiom from its textual wsml representation. However, since any graphical information has not been stored, the restored graphical model can significantly differ from the initial graphical design of the axiom. Moreover, in some cases the original logical structure of the axiom can not be restored.

Complete restoring both logical and graphical structure of axioms from their textual (wsml) representation is under development and we hope that this problem will be solved in the next version of the INFRAWEBs Axiom Editor.

5 CONCLUSION AND FUTURE TRENDS

This User's Guide is intended to get the users acquainted with the general operation of the INFRAWEBs Axiom Editor. A summary of the editing operations can be found in Appendix A.

The INFRAWEBs 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)*.

The current version (1.0.5) of Axiom Editor possesses many restrictions. There are many axioms, which cannot be created solely with Axiom Editor. However, we believe that this tool is a fairly flexible and can be easily learned and used even by novice SWS users.

Main directions for future development of the Editor include:

- Transformation of the INFRAWEBs Axiom Editor to an integrated Service Capability Editor by extending it with some customized modules of WSMO Studio and integrating with more advanced realization of INFRAWEBs Case-based Memory [Agre and Boyanov 2006].
- Extending application domain of the INFRAWEBs 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 WSMML ontologies as well.
- Developing a robust method for reconstruction of the graphical model of the WSMML-based axiom from its plain text representation.

REFERENCES

[Agre and Boyanov 2006] G. Agre and A. Boyanov. INFRAWEBs Deliverable D5.4.2.2. Realization of Service Composition in Design Time: Self Organizing Case Memory, February 2006.

[Bruijn et al. 2005] Bruijn, J.; Lausen, H.; Krummenacher, R.; Polleres, A.; Predoiu, L.; Kifer, M.; Fensel, D.: D16.1 – The Web Services Modeling Language (WSML). WSML Draft, October 2005.

[Roman et al., 2005] D. Roman, U. Keller, H. Lausen (eds.): Web Service Modeling Ontology, WSMO Final Draft, version 1.2, 2005.

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

APPENDIX A. TABLE OF OPERATIONS

This table contains a summary of the most important semantically-correct operations in Axiom Editor.

Operations for creating elements of Axiom Model	
Create a variable	This operation creates a new variable in the graphical axiom modeling 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. This guarantees the uniqueness of variable names across the axiom.
Create an operator	The operation creates a new logical operator of a specified type in the modeling area. The operator's type is selected from the menu – it can be OR, AND or NOT.
Create an instance	The operation adds an instance to the graphical modeling area. The user is given the opportunity to select the instance from Ontology Store.
Create a connection (advanced mode)	The operation creates a new connection between two elements placed on the modeling 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	The operation adds a relation to the modeling 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 Capability Editor takes care of changing the variable's name from the old one to the new at all its occurrences in the model.
Declare a variable as shared	Each variable defined in the axiom model can be marked as being a “ <i>Shared variable</i> ”. This means that such a variable may be used in more than one axiom and it must represent one and the same object in all these axioms.
Involve a variable in a relation (Advanced mode)	A variable that has been already placed at the axiom modeling 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.
Hide unused	Minimize graphical appearance of a variable by hiding all its

attributes	attributes that are not refined.
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	<p>The operation creates a new variable at the modeling 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.</p> <p>The name of the new variable is automatically set equal to the name of the selected attribute value being refined.</p>
Refine an attribute by an instance	The operation 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 modeling 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.
Additional attribute value	Allows to add an additional value for refining a multiple value 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 variable attributes”).
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	This operation is used for changing the type of an operator selected from the modeling area.
Delete an operator	The use of this operation 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 An operand	This operation adds a new operand to a selected operator placed at the modeling area and links them by a connection. The new operand can be either an existing model element from the modeling 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	This operation 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	Deletion of an instance leads to deletion of that instance along with all connections incoming to it from the model, thus keeping the axiom consistent.
Operations on Connections	
Insert an alternative	We would like to remind that 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	This operation 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	This operation inserts a NOT operator in the middle of any connection.
Reconnect a source/target element (Advanced mode)	This operation 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.