A Reusable 3D Visualization Component for the Semantic Web

Alessio Bosca*
Politecnico di Torino, Italy

Dario Bonino†
Politecnico di Torino, Italy

Marco Comerio‡
Università degli studi di Milano-Bicocca

Simone Grega§
Universita' degli studi di Milano-Bicocca

Fulvio Corno¶
Politecnico di Torino, Italy

Abstract

Ontology visualization and exploration is not a trivial task as many issues can affect the effectiveness of interactions. As ontologies are, in the general case, quite connected graphs where concepts are the nodes and semantic relationships the edges, the problems include space allocation, edge superposition, scene over-crowding, etc.

In this paper we propose a solution for the visualization and the exploration of ontologies using a 3-dimensional space, where information is represented on a 3D view-port enriched by visual cues. Our visualization tool aims at tackling representation issues of ontology models (as space allocation or the completeness and readability of displayed information) by adopting different views, at different granularities, in order to grant a constant navigability of the rendered model. Each provided view represents semantic information according to a different, task-based visualization paradigm, at a suitable level of detail.

Besides being primarily implemented as a Protégé plug-in, the proposed solution (named OntoSphere3D) is designed to be a reusable visualization component within Semantic Web applications; in fact, every scene can be exploited as a standalone facility that provides access to ontological data through an intuitive and appealing 3D interface. A case-study, is presented, where re-usability is demonstrated by integrating the OntoSphere3D visualization inside an Eclipse-based tool for Web Service design (called Web Services Design Tool) developed by some of the authors in the context of another research project.

CR Categories: I.6.9 [Information Visualization]: Visualization techniques and methodologies—;I.2.13 [Knowledge modeling]: ;— [I.2.13]: Ontology languages—;

Keywords: 3D graphics, Semantic Web, Ontology Exploration, Visualization

1 Introduction

As nowadays Semantic Web Technologies are constantly evolving to an ever-increasing maturity, a developer can start to seriously consider the opportunity to provide semantically tagged content, being the needed standards and tools already available. However, the current web panorama shows a very little adoption of semantics in contrast with the considerable added value that can be achieved. The reasons for such a poor exploitation can be various as different aspects are involved: technology immaturity, failing dissemination, user and developer reluctance to changes, etc.

In the sea of possible failures and shortcomings, interfaces play a relevant role, often discriminating good solutions from bad ones. This is especially true for tools related with knowledge modeling and visualization, where the involved information can be complex and can involve multidimensional issues. In particular, the cardinality of the involved information, (i.e., the very large amount of conceptual entities that are usually encompassed by a formal model) strongly impacts the scenario and requires countermeasures to keep tools interfaces usable.

De facto interfaces for knowledge modeling (i.e. for ontology creation and visualization) already exist, such as Protégé and OntoEdit, which are complete IDEs that address in a single application all the aspects related to ontology creation, checking and visualization (through proper plug-ins). These tools, although adopting rather different paradigms for editing, visualizing and inspecting ontologies, have in common a 2-dimensional approach to ontology visualization. Visualizing knowledge in two dimensions is a challenging task, since many dimensions are involved: instances, concepts, hierarchical relations just for naming a few. 2D approach is a well studied paradigm that already produced good solutions (such as: GraphViz, Jambalaya, OntoViz, etc), nevertheless, mapping the many dimensions involved by an ontology on only two dimensions can sometimes be too restrictive.

In this paper we propose a solution for the visualization and the exploration of ontologies using a 3-dimensional space, where information is represented on a 3D view-port enriched by visual cues. The approach aims at tackling visualization issues for ontology visual models by adopting a dynamic collapsing mechanism and different views, at different granularities in order to grant a constant navigability of the rendered model.

A preliminary description of the work has been presented in [Bosca and Bonino 2006] and the innovation of proposed work consists in the enhancement of the visualization capabilities through the full support and graphical representation of the OWL logic constraints (restrictions, cardinalities, disjointness) along with the introduction of animations within scene interchanges in order to preserve a coherent contextual perception for users. Another major innovation consists in the reorganization of the internal architecture of the tool to allow its reuse as visualization facility for applications, providing access to ontological data through an intuitive and appealing 3D interface. In other words, besides being implemented as a Protégé plug-in, Ontosphere3D also aims at being a reusable visualization component within Semantic Web applications. An effective integration with the Web Services Design Tool is presented as a case-study.

The paper is organized as follows: Section 2 presents relevant related works. Section 3 introduces the main issues in visualizing ontological data and provides a description of the tool analyzing

*e-mail: alessio.bosca@polito.it
†e-mail: dario.bonino@polito.it
‡comerio@disco.unimib.it
§e-mail: grega@disco.unimib.it
¶fulvio.corno@polito.it

Web3D 2007, Perugia, Italy, April 15–18, 2007.
© 2007 ACM 978-1-59593-652-3/07/0004 $5.00

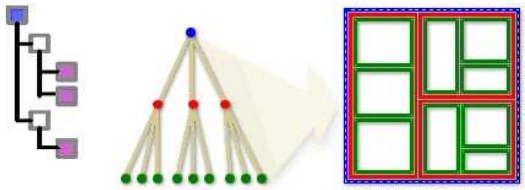

Figure 1: *TreeViews: indented (A), nodes and arcs (B), TreeMap (C).*

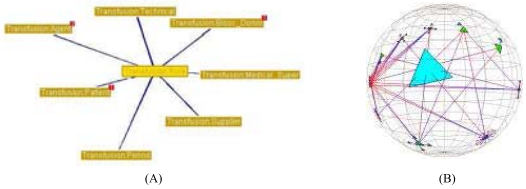

Figure 2: *Neighborhood View (A), Hyperbolic View (B).*

the various solutions designed for conveying information in more efficient ways. Section 4 introduces the case-study and details how the visualization tool can be integrated within an external application. Eventually section 5 concludes the paper and proposes some future works.

2 Related Works

The issue of visually representing abstract information is not new; relevant backgrounds for our work can be found in the field of Information Visualization as well as in best practices techniques adopted by existing solutions for ontology representation.

Information Visualization [Schneiderman 1999] (p.7) is defined as "the use of computer-supported, interactive, visual representation of abstract data to amplify cognition", therefore research activities within such field investigate the mapping of data records and attributes onto a visual representation, capable of enhancing and easing user understanding of presented information. Without any obvious spatial mappings or visible form, the challenge of visualizing abstract information consists in determining effective visual representations and interaction schemes for human analysis and exploration. In the scope of our project the work of Nicholas Polys, [et al. 2005] (p159-160), assumes a particular relevance as it proposes the use of different representations and interaction techniques for different data types and task and such principle is one of the core strategies of our approach.

The existing techniques for the visualization of ontologies can be summarized in four main visual schemes, possibly cooperating in more complex scenarios: network, tree, neighborhood, and hyperbolic. The network view represents an ontology as a generic network of connected elements and is usually exploited when the knowledge elements cannot be conveniently organized in hierarchies. The tree (or hierarchical) view, instead, is generally used for more structured ontologies. However, the simple hierarchical representation provided by this view is unable to represent connections between distinct sub-trees that violate the dominant taxonomic structure. In such a case, the connections violating the hierarchy are indicated in separate views, so complicating the navigation of the structure. The most common examples of tree views are based on indentation, as in file system browsers, or on diagrams with nodes and arcs. However, a tree-map view has also been proposed by Schneiderman [Shneiderman 1992], at the Maryland University, which uses nested rectangles to represent sub-classes (Figure 1, C).

The main advantage of tree views is that they can be displayed with rather little effort in comparison with network-oriented views. More importantly, entire sub-trees can be easily collapsed (i.e., temporarily hidden) to concentrate the attention on the rest of the knowledge base. The next two schemes apply similar principles on network-based structures: in fact, both the neighborhood and the hyperbolic views (Figure 2) focus the attention on a chosen node and its nearest neighbors. In the former case only the semantically nearest nodes are displayed, whereas in the latter case the nodes are displaced onto a semi-spherical surface, projected onto the visualization area, therefore magnifying the central nodes while shrinking the peripheral nodes.

The aforementioned representation schemes have been utilized in numerous applications with assorted enhancements.

Protégé [Knublauch 2003] has been developed by Stanford Medical Informatics at the Stanford University School of Medicine. It is intended as editor for general knowledge base systems. In Protégé, functionalities and visualization capabilities depend on the installed plug-ins, therefore it is quite difficult to perform a comparative analysis with the other tools available. It can only be observed that Protégé is, in general, more suited for advanced users as profound knowledge about ontologies is required and knowledge of description logic is recommended because a lot of interface elements are more or less related to logics notation. Protégé natively allows the interactive creation and visualization of classes in a hierarchical view. Each concept in the tree can be displayed along with additional information about the related classes, properties, descriptions, etc., which can all be quickly edited. Other panels manage class instances, alternative user interfaces, queries, and possibly other extensions which can be easily added to the framework as plug-ins. Particularly, various plug-ins are available for Protégé in order to enhance the visualization of the ontology and are therefore here discussed.

The OntoViz [OntoViz] plug-in displays an ontology as a graph by exploiting an open source library optimized for graph visualization [Gansner and North 1999]. Intuitively, classes and instances are represented as nodes, while relations are visualized as oriented arcs. Both nodes and arcs are labelled and displaced in a way that minimizes overlapping, but not the size of the graph. Therefore, the navigation of the graph, enhanced only by magnification and panning tools, does not provide a good overall view of the ontology, as the graphical elements easily become indistinguishable. OntoViz supports visualization of several disconnected graphs at once. The users can select a set of classes or instances to visualize. OntoViz generates graphs that are static and non-interactive which makes it less suitable for the visualization of large ontologies. TGViz [TGVizTab], similarly to OntoViz, visualizes Protégé ontologies as graphs. In this case however, the displacement of nodes and arcs is computed using the spring layout algorithm implemented in the Java TouchGraph library [TouchGraph].

This problem is less critical in Jambalaya [Storey 2001; Jamabalaya], another ontology viewer for Protégé, based on a tree-map scheme or rather nested interchangeable views, namely Simple Hierarchical Multi-Perspective (SHriMP). SHriMP is a domain-independent visualization technique designed to enhance how people browse and explore complex information spaces.An animated view of the ontology graph facilitates the navigation and browsing at different lev-

els of abstractions and details, both for classes and relations, while keeping low the learning curve through well-known zooming and hypertext link paradigms. However, text labels and symbols tend to overlap when the ontology grows in complexity and it is difficult to understand the relations among classes or instances.

OWLViz [OWLViz Tab] is a Protege plugin that visualizes the schema hierarchy, based only on the subclass relationship. OWLViz allowins comparison of the asserted class hierarchy and the inferred class hierarchy. OWLViz integrates with the Protege-OWL plugin, using the same colour scheme so that primitive and defined classes can be distinguished, computed changes to the class hierarchy may be clearly seen, and inconsistent concepts are highlighted in red. OWLViz has the facility to save both the asserted and inferred views of the class hierarchy to various concrete graphics formats including png, jpeg and svg.

The Visualizer plug-in of OntoEdit [OntoEdit ; Sure et al. 2002] proposes a bi-dimensional graph-based view of the ontology using colored icons as nodes accompanied by contextual tooltips, such as colored borders or spots other than the usual labels. However, in the resulting view both concepts and properties are represented as vertices, which can sometimes result confusing. Furthermore, graph lay-outing is done automatically and does not permit vertices re-allocation through dragging. As a consequence, the only possibility to browse the graph is by navigating the structure starting from the "root" node.

IsaViz [IsaViz] is another graph-based visualization tool for RDF models based on the GraphViz library. In this case, the principal enhancement to the previously mentioned approaches based on graphs is the Radar View, which, similarly to Jambalaya, displays a simplified network overview of the overall ontology in a small window, highlighting the currently edited region in a rectangle. In addition, icons and colors are also exploited to concentrate information, while different visualization styles and layouts are supported through the GSS (Graph Style Sheet) language, derived from the well-known CSS (Cascading Style Sheet and SVG (Scalable Vector Graphics) W3C recommendations. However, it is still not possible to customize the level of details for big ontologies.

OntoRama [OntoRama] is an ontology browser for RDF models based on a hyperbolic layout of nodes and arcs. As the nodes in the center are distributed on more space than those near to the circumference, they are visualized with a higher level of detail, while maintaining a reasonable overview of the peripheral nodes. In addition to this pseudo-3D space, OntoRama also introduces the idea of cloned nodes in order to reduce the number of crossed arcs and enhance the readability. The duplicate nodes are displayed using an ad-hoc color in order to avoid confusion. Unfortunately, this application does not support editing and can only manage RDF data. Many other tools are available and an exhaustive comparison between them (including the ones described in this paper) is available at [Survey].

3 OntoSphere3D

3.1 Ontology Visualization Issues

Ontology visualization implies abstracting from the formalism of the underlying data and graphically modeling the information contained in a given knowledge base. Independently from the approach adopted, visualizing complex graph structures like ontologies presents strong completeness and readability issues, which become more and more important as the number of nodes increases. Displaying a great amount of items at the same time on the screen worsens, in fact, the graphical perception of the scene and complicates spotting details, requiring users to autonomously extrapolate information from a chaotic assortment of data. This has to be avoided in every user-centered application, and in particular in knowledge management systems that, instead, shall support easy organization and representation of information meaning. In order to properly present data, a successful visualization strategy shall address completeness and readability issues. Completeness requires to visually represent all the relevant information associated to a given element of the model, while readability enforces a clear and easily interpretable presentation of data. In general terms, a visualization strategy shall find a possibly optimal, dynamic trade-off between completeness and readability and shall favour a display scheme instead of another depending on inferred user needs.

In the ontology representation domain, strategies are varied and share the common feature of representing data on a 2-dimensional view-port. OntoSphere3D, instead, uses a 3D space as a means to represent and explore data more effectively. Adopting a 3D representation space, in facts, offers several advantages: at first, it implements a natural perspective for human beings and, at second, it offers an additional dimension along which organizing data to be displayed. Moreover, a 3D space, implicitly carries a "focus" information, through space allocation. This means that objects located in a 3D environment can be dynamically moved to the fore or in the background and such opportunity offers a further instrument to emphasize and highlight some elements with respect to others. The targets of inspection are naturally placed closer to the viewer and increased in size, while others elements are kept on the scene but their presence is reduced and veiled thus not interfering with user attention.

3.2 Visualization Strategies: Views and Visual Cues

OntoSphere3D tackles completeness and readability issues exploiting two different representation principles:

- To increase the number of "dimensions" (colors, shapes, transparency, etc.) which represent concepts features and convey additional information without adding the burden of further graphical elements, such as labels, on the scene.
- To automatically select which part of the Knowledge Base has to be displayed and the detail level that has to be used in the process, on the basis of user interaction with the scene.

Color is used to differentiate elements as it increases the amount of information that can be integrated into the visual representation, creating different layers of information. In the is-A hierarchies different colors are used for visually grouping different levels; the white color identifies a collapsed element while the black indicates the focus. The green instead is the base color for incoming relations, red for the outgoing ones and datatype properties are depicted in blue. Furthermore, through the relations contextual menu, users can specify custom colors for specific relations they want to "highlight". The size of an element intensifies/weakens its presence and impact within the scene, as in collapsed nodes where the size grows with the number of subsumed / aggregated entities; similarly, in arrowhead cones the size grows with the number of relations. Transparency increases the "abstraction" of a feature or highlights the presence of a hidden feature. In fact it marks the presence of instances and differentiates inherited relations from the direct ones. Finally it lessens the burden within the scene of default structural elements (as is-A relations in a tree view).

The dynamic selection of the elements to represent is a particularly important feature for improving overall system performances,

since scale factor constitutes a strong issue in visualizing ontologies.In order to fulfill such principle, OntoSphere3D exploits different scene managers that present and organize the information on the screen, each one according to a differently detailed perspective. The idea behind such approach is that different tasks demand different amounts of information and different presentations, emphasizing certain features of data and omitting the others. Each ontology feature is made accessible and can be visualized exploiting a focusing mechanism that, on the basis of user interaction with the graphical elements, allows shifting from high-level pictures to more detailed representations.

3.3 Architecture Overview

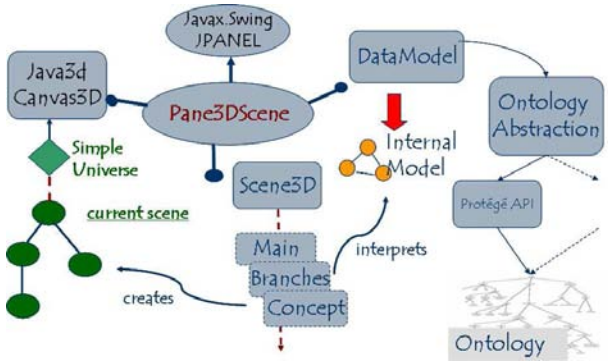

Figure 3: *Architecture Overview*

The architecture of the tool ideally conforms to the Model View Controller (MVC, see [Reenskaug 1979]) design pattern, typical of Web Applications domain. MVC predicates the separation between the model, holding the domain information, and its presentation. Therefore changes to the user interface don't impact data handling, and data can be reorganized without changing the user interface. The MVC design pattern solves this problem by decoupling data access and business logic from data presentation and user interaction through the introduction of an intermediate component: the Controller. This component is charged of event handling, typically user actions; it coordinates the presentation of the different views and may invoke changes on the model.

Ontosphere3D implements such design pattern by using an internal representation of model that depends on an ontology abstraction layer, and different graphical scene managers that presents the information at different detail levels, each according to its own strategy. The role of the controller in the tool is played by the Pane3DScene component that loads the ontology model and hosts the 3D canvas where the different scene managers represent the information. The controller also manages the scene manipulation behaviors and controls the scene transition animations that organize the flow of elements on the screen between two different visualization schemes.

3.4 User Interface

The OntoSphere3D user interface is quite simple, mouse centered, and supports scene manipulation through rotation, panning and zoom. It is strongly bound to the "one hand" interaction paradigm, allowing to browse the ontology as well as to update it, or to add new concepts and relations (thanks to the integration in the Protégé framework). Furthermore, an intuitive navigation interface, featuring direct manipulation of the scene allows domain experts, who have little technical skills in the field of Semantic Web, to graphically explore ontology components and interact with them.

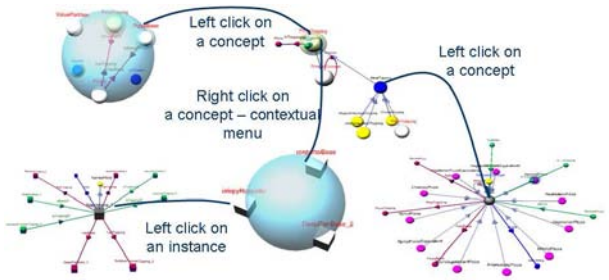

Figure 4: *Scenes Interactions*

Ontology elements are represented as follows: concepts are shown as spheres, instances are depicted as cubes, literals are rendered as cylinders and the semantic relationships between entities are symbolized by arrowed lines. Together with shapes, colors and sizes are used to convey additional information on the elements within the scene. Concepts and instances are click-able: left clicks perform a focusing operation, shifting the currently visualized scene to a more detailed view; central clicks are used to expand/collapse elements, while right clicks open a contextual menu offering a set of alternatives that depends on the element properties.

Relations as well can be clicked; in particular, a left clicks shows a textual representation of the relation properties and constraints (as cardinality) while a right clicks offer a proper contextual menu. Whenever a relations has a cardinality constraint it is represented with a squared section, otherwise it is round.

Manipulation of the whole scene instead occurs through the interaction with the background: left-clicking and moving the mouse cause the scene to rotate, pressing central button and shifting the cursor translates the scene and the mouse-wheel controls the zoom. A certain degree of scene personalization in terms of sizes of graphical components, distances between them and colors is supported through a proper option panel.

Scenes interchange in managing the graphical space as user attention shifts from one concept to another, being the attention focus implicitly inferred from user's interaction with the scene (e.g., a concept selection with a mouse click, see Figure 4). Simple animations guides the transitions through scenes in order to maintain a coherent perception of the information for users. Four different scenes are available: a main scene, a tree-focus scene, a concept-focus scene and a couple of instance-focus scene; they are explained in the following subsections.

3.4.1 Main Scene

This perspective constitutes the main scene of OntoSphere3D and the starting point for ontology inspection; it presents a big "earth-like" sphere bearing on its surface a collection of concepts represented as small spheres (see Figure 5). No taxonomic information is visualized in this representation, which only shows direct "semantic" relations between elements, usually a graph not fully connected. Atomic nodes, the ones without any subclass, are smaller and depicted in blue while tree-root nodes are colored in white and their size is proportional to the number of elements contained in their own sub-tree. The main scene, constitute the starting point in ontology inspection as it represents the ontology primitives (i.e. the

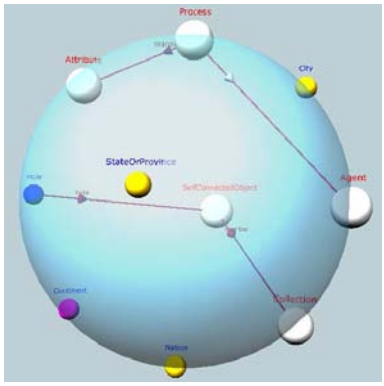

Figure 5: *mainScene*

root concepts) so depicting the conceptual boundaries of the domain and providing a very good hint to the question: "what's the ontology about?".

3.4.2 Exploring sub-Branches

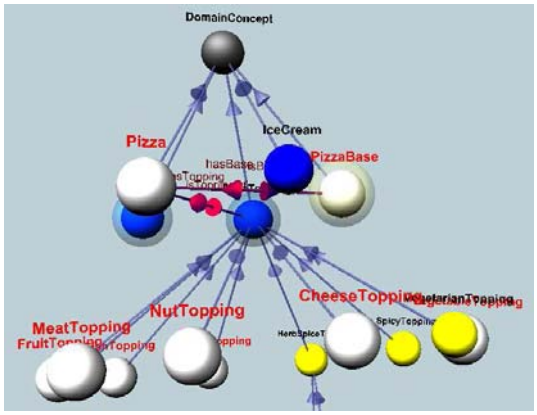

Figure 6: *Tree Focus Scene*

The scene (see Figure 6) shows the sub-tree originating from a concept, along the *Is-a* relation. It displays the predominant hierarchical structure as well as other semantic relations between classes. Since use evidence proves that too many elements on the screen at the same time, hinder user attention, the scene completely presents only three fully-expanded levels at a time. While the user browses the tree, the system automatically performs expansion and collapsing operations in order to maintain a reasonable scene complexity.

Collapsed elements are colored in white and their size is proportional to the number of elements present in their sub-tree. Concepts located at the same depth level within the tree have the same color in order to easily spot groups of siblings. *Is-a* relations are displayed with a neutral color (gray) and without label, whereas other semantic relations are colored. Whenever a relation is directed toward a node not present on the scene, a placeholder for that node is added in the proximity of the given element. No inherited properties or logical constraint are presented in this view

3.4.3 Focusing on a Concept

This perspective depicts all the available information about a single concept, at the highest possible level of detail (Figure 7). It reports the concept's parent(s) and its ancestor(s) and the semantic relations in which it is involved, both the ones specifically declared for the given concept and the ones inherited from ancestors. Semantic relations are drawn as arrows terminating in a small 3D iconic element representing the range. Direct relations are drawn close to the concept and with an opaque color, while inherited ones are located a bit farther from the center and depicted with a fairly transparent color. Besides structural information, the scene reports as well the logical constraints expressed on the class as disjoints elements, logical restrictions (cardinality, universal and existential quantifiers) or anonymous super-classes (as intersections, enumerations or unions..). These constraints are also used by the tool in order to filter out the inherited relations that has been redefined at a more specific level or the ones that don't comply to such constraints. The aim in fact is to provide all the relevant information filtering out the rest. This scene is pretty useful during consistency checking operations because it eases the spotting of inconsistent concepts or relations, e.g. whenever a concept inherits from an ancestor a property that "conceptually" contrasts with other features of its own.

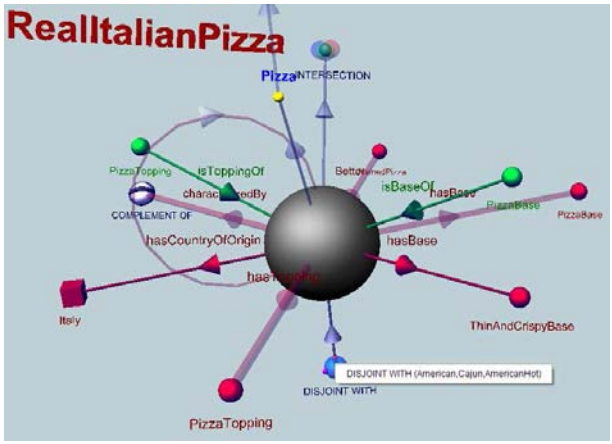

Figure 7: *Concept Focus Scene*

3.4.4 Exploring Instances

Whenever a concept has direct instances (in the tree focus scene or in the concept focus scene) its sphere is depicted surrounded by a translucent sphere, similar to a shell (see Figure 6, A). Direct instances, as well the whole set of instances within the sub-tree originating from the concept, can be visualized by right-clicking on this shell-surrounded concepts. Inspection of instances can be done using three scenes: the first, reachable through the right-click on concepts surrounded by a "'halo"', follows the main scene visualization paradigm (representing instances as cubes on the surface of a bigger sphere) and simply list a set of individuals. White cubes represent instances related with other entities while blue ones are simple, unconnected nodes. The second scene presents the graph of facts relative to a given instance and the adopted visualization strategy is similar to the Concept Focus Scene, representing related instances as a interconnected cubes (see Figure 8, B). The third scene instead a dependency tree originating in a given instance and insisting on a previously selected relation.

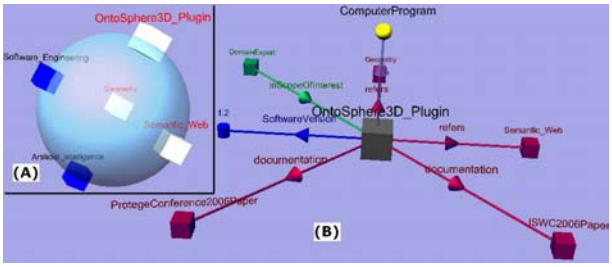

Figure 8: *(A) Instances Focus Scene (B) Facts Focus Scene*

4 WSDTool: an integration case study

The Web Services Design Tool (WSDTool) [Comerio et al. 2007b] has been developed in order to allow the different actors (project leader, software designer, business expert and domain expert) involved in Web services design to define and address quality requirements. WSDTool provides a graphical interface aiming at supporting ontologies management and modelling activities. The tool has been implemented as an Eclipse plug-in [Eclipse 2005] to primarily guide the execution of the Web Services Modelling Design (WSMoD) methodology [Comerio et al. 2007a].

The WSMoD approach consists in incorporating and refining non functional properties (NFP), as well as functional requirements, along the Web service design process. The advantage of managing NFP is twofold: achieving at the end of the process, a ready-to-implement specification, and reducing the risk of delivering unsatisfactory services. In fact, the knowledge of the technical, organizational and social characteristics of the environment in which the service will be deployed, allows the designer to effectively evaluate and tailor the service specifications.

WSMoD makes use of ontologies in order to understand, discover, classify and reason on NFPs that are related to user constraints, preferences, technological features (such as devices and networks), and domain peculiarities. Ontologies provide a description, classification and characterization of services, context (user and channel characteristics) and Quality of Service (QoS). WSMoD therefore uses ontologies in order to provide a formal organization of the knowledge that can be exploited to rationalize the process of decision and evaluation.

The WSMoD methodology (see Fig.9) is composed of five main phases. The *Service Identification* phase specifies, from a business point of view, the features the service will offer, and the business constraints. This phase produces a complete, informal specification of functional and non-functional requirements. Then, the functional requirements are the input of the *Service Modelling* phase, that has the goal of defining the functional model of the Web Service. The non-functional requirements and the functional model are inputs of the *High-Level Re-Design* phase, whose goal consists in revising, and possibly changing, the functional model in order to include non-functional requirements as well. The next phase, *Customization*, revises the current design model by considering a possible context of execution. Finally, the *Web Service Description* phase translates functional models obtained into standard WSDL interfaces, augmented with quality descriptions in WSOL [Tosic et al. 2002].

To support the execution of these phases, WSDTool provides different functionalities (e.g., service modelling, QoS evaluation, etc...) embedded in specific panels. In particular, the tool supports ontology management (i.e., import, navigation and editing of ontologies)

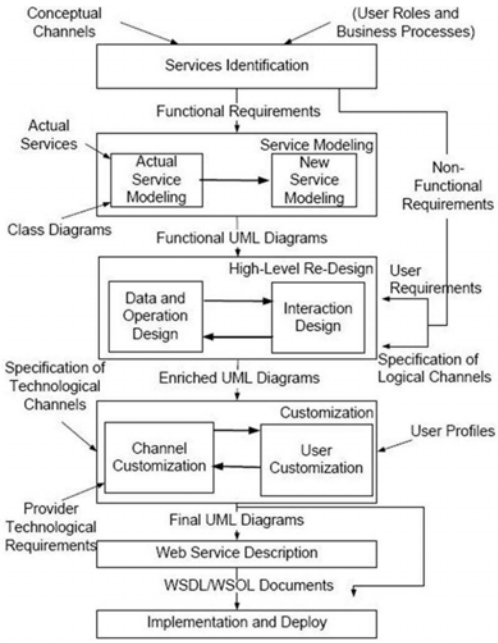

Figure 9: *WSMoD methodology*

using the facilities provided by the OntoSphere3D tool. The OntoSphere3D component has been integrated in the WSDTool Eclipse plugin and the different graphical scenes, described in Section 3, can be directly visualized within WSDTool components in order to guide the different actors along the Web services design. In order to perform this integration, two problems have been tackled:

1. Ontosphere3D has been implemented with the Abstract Window Toolkit (AWT) while Eclipse uses the Standard Widgets Toolkit (SWT);

2. the different scenes provided by OntoSphere3D must be automatically visualized and updated along the design process.

The solution adopted to solve the first problem consists in the use of an eclipse package (org.eclipse.swt.awt) that allows a bridge among these two types of graphical libraries using a class named SWT_AWT. In particular, this class provides support for embedding AWT widgets within SWT composites and vice versa.

The second problem instead has been solved exploiting the API provided by the OntoSphere3D tool. WSDTool, in fact, holds an internal representation of the ontology model and accesses the different scene managers in order to provide the desired visualization. The tool, with respect to the architecture described in 3, substitutes the Panel3D in managing the different views and provides interfaces tailored to the different phases of the Web services design.

This integration evidences the possibility to use OntoSphere3D as a reusable component with little efforts in terms of code customization and without constraints in supporting the native 3D graphical implementation.

OntoSphere3D interface is mainly used in order to support the execution of the Service Identification phase. In fact, business experts involved in this phase use the ontologies to: (i) classify the new service in the proper service category according to its functional requirement; (ii) discover non-functional properties of the selected category; (iii) create the new service instance. The effectiveness of the OntoSphere3D along the execution of this phase has

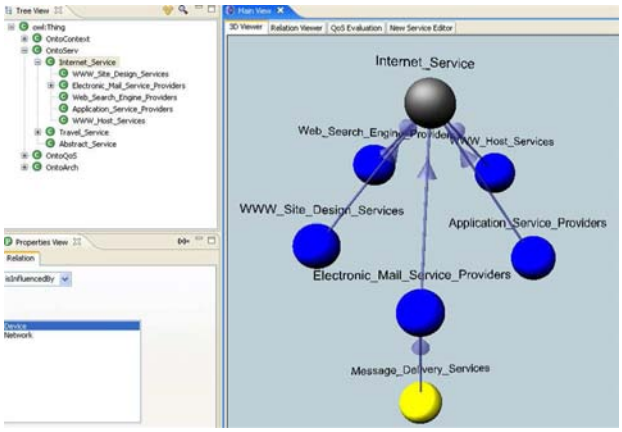

Figure 10: *Discovery of the FlexSend category*

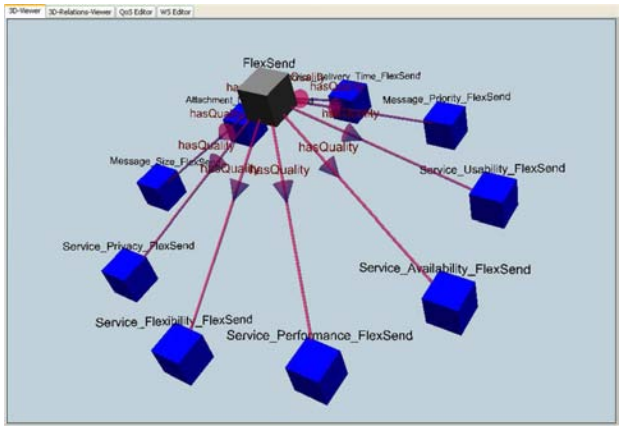

Figure 11: *Instance of the FlexSend service*

been evaluated with a case study: the design of a notification service, FlexSend, that delivers messages over different channels according to specific receivers contexts (preferences, device, activity). More details about such case study can be found in [Comerio et al. 2007a].

The Service Identification phase starts with the definition of the FlexSend functional requirements and the simultaneous creation of a UML use-cases diagram. Afterwards, the phase proceeds with the identification of the category that FlexSend shall belong to. Business experts, in fact, browse the service ontology in order to identify the most suitable category and to create the new instance. Starting from the Main Scene it is then possible to select the *Internet Service* category and visualize its sub-tree as shown in Fig.10.

Once FlexSend category has been individuated as *Message Delivery Service*, users can visualize its properties at a higher detail level with the Concept-Focus Scene. In particular, OntoSphere3D allows users to easily notice that each service included in the *Message Delivery Service* category is characterized by a set of QoS inherited from parent categories (e.g., *Service Usability* from *Internet Service*). Moreover, the tool underlines that each *Message Delivery Service* is influenced by the network and the device on which it is delivered.

After the service categorization, business experts investigate the availability of concrete services that (partially or totally) satisfy the functional and non-functional requirements of FlexSend, in order to reuse them within the design process. Users can perform this operation by simply inspecting the different instances of the selected category; OntoSphere3D facilitates such activity by highlighting concepts with at least one instance and surrounding them with a transparent "halo". Fig.10 shows that no instances are available, therefore the design of a new service is required.

The creation of FlexSend service is performed using the *new service editor*. The editor allows business experts to specify FlexSend as an instance of *Message Delivery Service* and to exploit OntoSphere3D features in order to perform the following steps: (i) inheritance of all the qualities related to *Message Delivery Service* category; (ii) possibility of changing this QoS list removing or adding new quality requirements; (iii) specification of constraints that FlexSend must fulfil; (iv) specification of influences of context characteristics (e.g., conceptual devices, networks, etc...).

After its creation, FlexSend appears as an instance of the *Message Delivery Service* category within the service ontology, therefore, it is possible to visualize its features. In particular, the Instance-Focus scene allows business experts to easily locate all non-functional properties linked to FlexSend during the editing operations. Moreover, it is possible to select one specific property and (with a right click) visualize the dependency tree originating in that instance. For example, the figure 11 shows the sub-tree obtained selecting *has quality* as property of interest.

5 Conclusions

This paper presented some enhancements of the OntoSphere3D plug-in which allow, from one side a better support to the visualization of complex ontology constructs (OWL restrictions, cardinality constraints,...) and, from the other side a better integrability into other applications. A real world case study has been presented, which shows the adoption of the presented visualization system into a Web Service Design Tool developed by some of the authors. The paper is mainly focused on the architectural and integration issues, while the extensive experimentation of the tool usability is currently being performed and will constitute the basis for possible future works.

References

BOSCA, A., AND BONINO, D. 2006. Ontosphere3d: a multidimensional visualization tool for ontologies. In *17th International Conference on Database and Expert Systems Applications-DEXA 2006, 5th International Workshop on Web Semantics (WebS 2006)*.

COMERIO, M., DEPAOLI, F., GREGA, S., MAURINO, A., AND BATINI, C. 2007. Wsmod: A methodology for qos-based web services design. *International Journal of Web Services Research*.

COMERIO, M., DEPAOLI, F., AND VISCUSI, G. 2007. An ontology management tool for qos-based web services design. In *Proc. of WEBIST (International Conference on Web Information Systems and Technologies) 2007*.

ECLIPSE, 2005. Eclipse platform technical overview. `http://www.eclipse.org/articles/whitepaper-platform-3.1/eclipse-platform-whitepaper.html`.

ET AL., V. G. C. C. 2005. *Visualizing Information Using SVG and X3D*. Springer-Verlag.

GANSNER, E. R., AND NORTH, S. C. 1999. An open graph visualization system and its applications to software engineering. *Software Practice and Experience 30*, 11, 1203–1233.

ISAVIZ. A visual authoring tool for rdf. `http://www.w3.org/2001/11/IsaViz/`.

JAMABALAYA. `http://www.thechiselgroup.org/chisel/projects/jambalaya/jambalaya.html`.

KNUBLAUCH, H. 2003. An ai tool for the real world: Knowledge modeling with protégé. *JavaWorld*.

ONTOEDIT. `http://www.ontoknowledge.org/tools/ontoedit.shtml`.

ONTORAMA. `http://www.ontorama.com/`.

ONTOVIZ. Ontoviz tab: Visualizing protégé ontologies. `http://protege.stanford.edu/plugins/ontoviz/ontoviz.html`.

OWLVIZ TAB. `http://www.co-ode.org/downloads/owlviz/`.

REENSKAUG, T., 1979. Models - views - controllers. Technical note, Xerox PARC, December 1979. A scanned version on `http://heim.ifi.uio.no/~trygver/mvc/index.html`.

SCHNEIDERMAN, S. C. J. M. B. 1999. *Information Visualization: Using Vision to Think*. Morgan Kaufmann.

SHNEIDERMAN, B. 1992. Treemaps for space-constrained visualization of hierarchies. *ACM Transactions on Graphics (TOG) 11*, 1, 92–99.

STOREY, M. A. 2001. Interactive visualization to enhance ontology authoring and knowledge acquisition in protégé. In *Workshop on Interactive Tools for Knowledge Capture, Victoria, B.C. Canada*.

SURE, Y., ERDMANN, M., ANGELE, J., STAAB, S., STUDER, R., AND WENKE, D. 2002. OntoEdit: Collaborative ontology development for the semantic web. In *Proceedings of the first International Semantic Web Conference 2002 (ISWC 2002), June 9-12 2002, Sardinia, Italia.*, Springer, LNCS 2342.

SURVEY, O. E. `http://www.xml.com/2002/11/06/Ontology\_Editor\_Survey.html`.

TGVIZTAB. A touchgraph visualization tab for protégé 2000. `http://www.ecs.soton.ac.uk/ha/TGVizTab/TGVizTab.htm`.

TOSIC, V., PATEL, K., AND PAGUREK, B. 2002. Wsol - web service offerings language. In *CAiSE '02/ WES '02: Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*, Springer-Verlag, London, UK, 57–67.

TOUCHGRAPH. `http://touchgraph.sourceforge.net/`.