WSMO Studio – an Integrated Service Environment for WSMO

Marin Dimitrov, Alex Simov, Vassil Momtchev, Damyan Ognyanov

OntoText Lab. / SIRMA
135 Tsarigradsko Shose Blvd., Sofia 1784, Bulgaria
{firstname.lastname}@ontotext.com

Abstract. The Web Services Modelling Framework (WSMF) and the Web Services Modelling Ontology (WSMO) provide a unique, highly innovative perspective onto the Semantic Web and Web Services technologies. This paper introduces the *WSMO Studio* – a prototype that supports and elaborates this innovative perspective, making the technology easy to use and transparent for the end user. Our previous work on SWWS Studio has provided us with important experience and feedback, which we will reuse in our present work on *WSMO Studio* in order to provide a high quality integrated service environment for the Semantic Web Services domain.

1 Introduction

Robust, mature and easy-to-use tools play crucial role for the easy adoption of any new technology. In fact the overall value of a technological innovation can be severely undermined by the lack of proper tools that support it.

The Web Services Modelling Framework (WSMF, [10]) and the Web Services Modelling Ontology[1] (WSMO, [16]) provide a unique, highly innovative perspective onto the Semantic Web and Web Services technologies. We present a prototype that supports and elaborates that perspective, making the technology easy to use and transparent for the end user. In particular, we present the *WSMO Studio*[2] – an open source Integrated Service Environment (ISE) for the Semantic Web Services domain, developed within the scope of the EU-funded DIP project [1].

WSMO Studio is the successor of SWWS Studio [7], which has already been successfully used within several research projects. Our previous work on SWWS Studio provided us with important experience about the requirements of the users in the Semantic Web Services domain. *WSMO Studio* is an effort to provide an integrated service environment that better suits the needs of the users and overcomes the limitations of its predecessor.

This paper is organised as follows: section 2 presents the functional and non-functional requirements for an Integrated Service Environment for the Semantic

[1] http://www.wsmo.org
[2] http://www.wsmostudio.org

Web Services domain, section 3 presents the goals of the *WSMO Studio* and section 4 presents details on the architecture, components and functionality of the first prototype of the Studio.

2 Requirements

End-user Semantic Web Service tools should assist potential users for ontology creation, service description, discovery and composition. Based on our previous experience with building such tools we may outline some specific functional requirements for such client Graphical User Interface (GUI) tools:

- Means for effective interaction with ontologies. The UI should provide functionality for creating, versioning, browsing ontologies (including very large ontologies).
- Means for effective interaction with repositories – publishing and browsing of ontologies, goal descriptions, service descriptions, mediators, etc.
- Means for describing the capabilities of services, formulating goals and specifying mediators. The User Interface should allow even users with no good knowledge of logic to generate such descriptions (the tools should provide the underlying translation between logical formalisms if necessary).
- End-user interface for service discovery (e.g. finding services that achieve a specified goal). Such a GUI will assist the user for discovery of services that can later be used for manual service composition.
- Means for manual service composition and workflow specification.

2.1 Functional requirements

In the context of WSMO, the high-level requirements may be re-formulated and detailed to cover various aspects of the SWS lifecycle:

- Working with ontologies (including creating / modifying ontologies for SWS, mediation between ontologies, translation between different formalisms, etc.)
- Creating SWS descriptions in a WSMO centric way: goals, mediators, web services
- Describing service orchestrations and choreographies [18][17]
- Import and export of the SWS descriptions in various formats as specified in [6] (in the future other formats may emerge as well)
- Interaction with SWS repositories (such as [12]) and ontology datastores (for example Sesame[3])
- Interaction with SWS runtime environments, such as WSMX [5][21]
- Working with existing Web Service standards that may be indirectly relevant to the semantic description of a service (such as WS-MetadataExchange [3], WS-Policy [2], UDDI [19] and WSDL [20])

Note that this list presents the major functional requirements at present but it is far from static - as the Semantic Web Services domain evolves, it is likely that additional requirements will emerge as well.

[3] http://www.openrdf.org/

2.2 Non-functional requirements

The main goal of integrated environments such as *WSMO Studio* is to make users more productive[4]. Tool productivity can usually be increased in two ways:

- *by adding more features related to a specific task* (e.g. 'vertical' extension), for example better ontology editors, ontology viewers capable of visualising huge ontologies, service composers supporting more powerful workflow primitives, etc.
- or, *by providing features covering more tasks* (e.g. 'horizontal' extension), for example providing functionality for new tasks / perspectives, integration with already existing tools, etc.

As noted in [15], Integrated Development Environment (IDE) users rarely focus on only one aspect or task when they work. Indeed users usually play different roles and perform different related tasks in their everyday work, so the environment should provide functionality covering various perspectives and tasks.

In the scope of the Semantic Web Services, there are several roles such as:

- *ontology engineers* performing tasks such as creating / modifying / versioning / storing ontologies
- *service annotators* performing the actual semantic annotation of web services, e.g. translating a standard WSDL service, described in terms of ports, operations, inputs and outputs, into a WSMO service, described in terms of capabilities, pre/post conditions, assumptions, effects and mediators (all expressed in terms of the proper logical formalism)
- *service users* querying for services that can be integrated into their business process descriptions
- *service administrators* responsible for managing service repositories, datastores and execution environments

An integrated environment for Semantic Web Services should provide sufficient functionality relevant to each of these perspectives. At the same time the functionality should be provided an integrated way, since it is unlikely that a particular user working in the domain of SWS focuses only on a single task, e.g. the end result should be a cohesive and flexible Integrated Service Environment (ISE, [4])

Another important non-functional requirement can be identified with respect to the extensibility of a tool. As stated in [15]:

> *... open-ended extensibility is essential in the commercial IDE arena because no IDE vendor could possibly provide a sufficient set of useful tools to satisfy all customer needs. Which third party tool will be bundled as an add-in for a particular IDE is determined by market forces.*

[4] Note that different types of users may have different expectations (both functional and non-functional) from a tool

This statement, based on commercial IDE experience, can equally well be applied for ISE tools for the Semantic Web Services domain. Lack of extensibility is one of the main drawbacks of current SWS tools such as SWWS Studio [7] and IRS [13] at present.

Based on the above observations, we may define the following non-functional requirements for an Integrated Service Environment for the SWS domain:

- *Role-oriented development* – the ISE should allow for the end user to customise its functionality in a way that maximises its productivity for a specific goal. For example, specific functionality may be added (removed) if it is relevant (irrelevant) to a specific user task or perspective. If a sufficient level of flexibility is provided then the ISE may satisfy the needs of a broader target audience (from SWS domain experts to more naïve users).
- *Extensibility* – since it is difficult to envision and specify a stable set of requirements for an emerging domain as SWS, it is crucial that the tools built for the domain are highly extensible. This way, when the domain evolves in new directions, the tools will be able to follow this evolution and provide the relevant functionality for the respective users (the extensions may be provided not only from the original group or community that build the tool but also from 3^{rd} party contributors).
- *Open standards* – It is desirable that tools are designed and built in accordance with open standards and based on open and extensible architecture, so that the cost of adopting and extending the tools by 3^{rd} parties is low.
- *Flexible licensing* – an open source licensing of a product improves its adoption rate and increases both its quality and active community base (which contributes to the product). Nonetheless, important differences exist between various open source licences in terms of copyright, compatibility with proprietary licences. For an open ISE, which relies on 3^{rd} party contributions and extensions, it is important that the licence does not prevent such contributions.
- *Usability* – it is desirable that the tools provide the best UI experience for the end user. The UI should provide a convenient and easy-to-use abstraction of the domain being modelled in a way that maximises the end user productivity.

3 WSMO Studio Goals

The goal of the *WSMO Studio* effort is to provide a prototype that supports and elaborates that WSMO perspective onto Semantic Web Services technology, making it easy to use and transparent for the end user. In particular, we will provide an Integrated Service Environment for the Semantic Web Services domain, that satisfies the requirements outlined in the previous section.

In our opinion it is important that the functionality relevant to the SWS domain should be presented in a form that maximises its provided value, in other words both the functional and non-functional requirements should be handled by tool providers since focusing only on one of the aspects (for example by providing functionality in a non-extensible, proprietary way) is unlikely to provide the desired results in the long term.

The functionality of *WSMO Studio* will be provided by means of:

- *Design* of a framework for an integrated service environment compatible with the WSMO approach of describing SWS. The architecture will be based on the Eclipse[5] architecture [15][14]
- *Development* of a set of components (plug-ins) that add specific functionality to the service environment
- *Integration* of already existing components into the service environment (for example existing components for WSDL)
- *Contributions* of plug-ins to the Studio by 3^{rd} parties

The outlined non-functional requirements will be taken into consideration when designing and developing *WSMO Studio*:

- Role-oriented development – *WSMO Studio* is based on the Eclipse platform and its component (plug-in) based model, which is highly flexible and customisable – plug-ins may be added, removed and customised from the end user in a declarative manner, without any new development effort. The same base functionality (e.g. set of plug-ins) may be extensively customised to cover the perspective of a specific end-user group.
- Extensibility – the Eclipse component model presents a declarative specification of ways to extend the platform (called *extension points*). New plug-ins may extend existing plug-ins and new plug-ins may be seamlessly integrated into the platform at any time.
- Open standards – *WSMO Studio* will be based on industry proven open standards and initiatives such as Eclipse, as well as emerging standard proposals as WSMO.
- Flexible licensing – our recommendation is that an open source licence, specifically LGPL [11], is used for the architecture and the common runtime and while the plug-in contributors are free to release their contributions under a licence most suitable for them. According to this scenario, an end user deployment may be comprised of both open source and proprietary components (distributed under a licence chosen by the respective author)
- Usability – the Eclipse platform has a highly configurable and portable UI, which offers a native and high performance user interface for a variety of window systems for Windows, Linux, OSX and QNX. The Eclipse 3.0 release has targeted especially responsive and scalable user interface. The Eclipse platform has defined a set of user interface guidelines that aim at providing the best GUI experience for the end user [9].

4 An Eclipse Based Integrated Service Environment

The *WSMO Studio* architecture is comprised of the following layers:

[5] http://www.eclipse.org

– *Common runtime* – a layer providing common functionality across all components of the *WSMO Studio*. Such functionality includes: creating WSMO models, export and import from various formats (as defined in [6]) and basic serialisation and datastore support. The runtime layer is based on the *wsmo4j* library[6] [8] and its extensions. 3^{rd} parties may extend this layer to provide specific functionality required by some of the UI components from the next layers (for example functionality for exporting / importing from a new format, a new serialisation mechanism or additional datastore support)
– *Components* (plug-ins) – various UI components for working with ontologies, WSMO descriptions and goal, service and ontology repositories. The first version of the prototype will provide three such components (described below) but 3^{rd} parties may add new components (for example, a GUI for WSMO orchestration and choreography descriptions)
– *Extensions* – various extensions and customisations of existing components may be provided as well, for example, a new axiom editor that may replace the default axiom editor part of the ontology component

The first version of the prototype will provide three specific components:

– *Ontology editor* providing functionality for ontology repository browsing and ontology editing (creating and modifying descriptions of ontology components: concepts, instances, relations and axioms). The ontology editor in *WSMO Studio* is not intended to serve as a replacement of full-featured ontology editors such as Protégé[7]. Its purpose is to provide only the minimal ontology related functionality that is often required when creating WSMO service descriptions and it may be replaced with a more featured ontology editor by the respective user.
– *WSMO editor* providing functionality for creating descriptions of goals, mediators and web services in accordance with the Web Services Modelling Ontology [16]. The main sub-components forming the WSMO editor are: a mediator editor, a goal editor and a service editor
– *Repository browser* providing a centralised view of the ontology datastores and service / goal repositories (that the end-user may use when annotating services), repository querying (since navigation is not always sufficient for locating the elements of interest) and import / export of elements into / from the repository

Each component corresponds to one of the perspectives[8] defined in *WSMO Studio*.

Perspectives are very useful for achieving role-oriented development with the *WSMO Studio* by presenting only the functionality specific to a specific task

[6] http://wsmo4j.sourceforge.net
[7] http://protege.stanford.edu/
[8] A *perspective* in Eclipse is a container for logically and functionally related views and editors. It is recommended that a perspective provides all the functionality related to a specific task, so that the user won't have to switch between different perspectives to accomplish his task / goal

(role) and abstracting from features irrelevant to the user's goal. Such a role-oriented customisation of the tool provides maximum usability to the end-user and lowers the overall tool complexity.

Perspective	Target users	Functionality
Ontology	Ontology engineers	Create, modify and manage ontologies
WSMO	Service annotators	Create WSMO descriptions for existing web services
Repository	Service users, service annotators, service administrators	Browse, query and manage a repository of services, goals and ontologies

Table 1. *WSMO Studio* perspectives

The perspectives that are defined[9] at present in *WSMO Studio* are presented in Table 1

5 Conclusion

This paper presents a work in progress for an Integrated Service Environment for the Semantic Web Service domain, called *WSMO Studio*. A detailed list of the functional and non-functional requirements for the tool was provided, as well as the rationale for these requirements. The concrete means to achieve these requirements were outlined in the scope of the *WSMO Studio* prototype. Future work will be focused on extending the functionality of the Studio to cover more aspects of the Semantic Web Services domain.

References

1. DIP - Data, Information, and Process Integration with Semantic Web Services, IST project FP6-507483. http://dip.semanticweb.org.
2. S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Malhotra, A. Nadalin, N. Nagaratham, M. Notthingham, H. Prafullchandra, C. von Riegen, J. Schlimmer, C. Sharp, and J. Shewchuk. Web services policy framework, September 2004. Available at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policy.asp.
3. K. Ballinger, D. Box, F. Curbera, S. Davanum, S. Graham, C. Liu, F. Leymann, B. Lovering, A. Nadalin, M. Notthingham, D. Orchard, C. von Riegen, J. Schlimmer, I. Sedukhinand J. Shewchuk, B. Smith, G. Truty, S. Weerawarana, and P. Yendluri. Web services metadata exchange, September 2004. Available at http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-metadataexchange.pdf.

[9] 3^{rd} party contributions may define new, contribution specific perspectives as well

4. S. Benfield and P. Fingar. Managing web services. Internet World Magazine, May 2002. Available at http://www.internetworld.com/magazine.php?inc=050102/05.01.02tech1.html.
5. E. Cimpian, T. Vitvar, and M. Zaremba. D13.0: Overview and scope of WSMX. WSMX working draft, DERI, February 2005. Available at http://www.wsmo.org/TR/d13/d13.0/v0.2/.
6. J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel. D16.1: The Web Service Modeling Language WSML. WSML final draft, DERI, March 2005. Available at http://www.wsmo.org/TR/d16/d16.1/v0.2/.
7. M. Dimitrov, Z. Marinova, and P. Radkov. SWWS Studio – a WSMO compliant editor. In C. Bussler, D. Fensel, H. Lausen, and E. Oren, editors, *WSMO Implementation Workshop*, volume 113 of *CEUR*, Frankfurt, Germany, September 2004.
8. M. Dimitrov, D. Ognyanov, and A. Simov. wsmo4j programmers guide. Technical report, OntoText Lab., November 2004. Available at http://wsmo4j.sourceforge.net.
9. N. Edgar, K. Haaland, J. Li, and K. Peter. Eclipse user interface guidelines, February 2004. Available at http://www.eclipse.org/articles/Article-UI-Guidelines/Contents.html.
10. D. Fensel, C. Bussler, Y. Ding, and B. Omelayenko. The Web Service Modeling Framework (WSMF). *Electronic Commerce Research and Applications*, 1(2), 2002.
11. Free Software Foundation. GNU Lesser General Public License, version 2.1, February 1999. Available at http://www.opensource.org/licenses/lgpl-license.php.
12. R. Herzog, H. Lausen, D. Roman, M. Stollberg, and P. Zugmann. D10: WSMO registry. WSMO working draft, DERI, April 2004. Available at http://www.wsmo.org/2004/d10/v0.1/.
13. E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS-II: A framework and infrastructure for semantic web services. In *The Semantic Web*, volume 2870 of *LNCS*. Springer-Verlag, 2003.
14. Object Technology International Inc. Eclipse platform technical overview, 2003. Available at http://www.eclipse.org.
15. J. Des Rivieres and J. Wiegand. Eclipse: A platform for integrating development tools. *IBM Systems Journal*, 43(2), 2004.
16. D. Roman, H. Lausen, U. Keller, J. de Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Kifer, J. Kopecky, R. Lara, E. Oren, A. Polleres, and M. Stollberg. Web Service Modeling Ontology, v1.2. WSMO working draft, DERI, February 2005. Available at http://www.wsmo.org/TR/d2/v1.2/.
17. D. Roman and J. Scicluna. D15: Orchestration in WSMO. WSMO working draft, DERI, January 2005. Available at http://www.wsmo.org/2005/d15/v0.1/.
18. D. Roman, J. Scicluna, C. Feier, M. Stollberg, and D. Fensel. D14: Ontology-based choreography and orchestration of WSMO services. WSMO working draft, DERI, March 2005. Available at http://www.wsmo.org/TR/d14/v0.2/.
19. UDDI. UDDI version 3.0. Technical report, UDDI Working Group, July 2004. Available at http://www.uddi.org.
20. W3C. Web services description language (WSDL) version 2.0, part 1: Core language. W3C working draft, W3C, March 2004. Available at http://www.w3.org/TR/wsdl20.
21. M. Zaremba and M. Moran. D13.4: WSMX architecture. WSMX working draft, DERI, April 2005. Available at http://www.wsmo.org/2005/d13/d13.4/v0.2/.