Iridescent: a Tool for Rapid Semantic Web Service Descriptions

Thanos G. Stavropoulos[1,2], Dimitris Vrakas[1,2] and Ioannis Vlahavas[1,2],

[1] Aristotle University of Thessaloniki, University Campus,
54200 Thessaloniki, Greece
[2] International Hellenic University, 14th km Thessaloniki-N.Moudania,
54400 Thermi, Greece

{athstavr,dvrakas,vlahavas}@csd.auth.gr

Abstract. Although the Semantic Web and Web Service technologies have already formed a synergy towards Semantic Web Services, their use remains limited. Potential adopters are usually discouraged by the plurality of methodologies and the lack of tools which in turn force them to acquire expert knowledge and commit to exhausting manual labor. This work proposes a novel, functional and user-friendly tool, named Iridescent, intended for both expert and non-expert users to rapidly create and edit Semantic Web Service descriptions, following the SAWSDL recommendation. The tool's aim is twofold: to enable users manually create descriptions in a visual manner, providing a complete alternative to coding, and to semi-automate the process by matching elements and concepts and suggesting annotations. A state-of-the-art survey has been carried out to reveal critical requirements and compare Iridescent to existing tools. Usage scenarios demonstrate how Iridescent enhances the authoring process and in turn enables Intelligence e.g. in an Ambient Intelligence environment. Finally, the tool was methodically tested for usability and evaluated by a range of expert and non-expert users.

Keywords*:* Semantic Web; Web Services; Semantic Web Services; Tools;

1 Introduction

As Web Services and WSDL[1] have emerged, for users to get things done over the Web, Semantic Web technologies promise to enhance their lifecycle. Initial top-down approaches, such as OWL-S[2] and WSMO[3] are upper ontologies that define numerous functional and non-functional aspects for a Service focusing less on its running instances (its grounding) for invocation. More recently, approaches shifted towards a

[1] Web Service Description Language - WSDL (W3C Recommendation): http://www.w3.org/TR/wsdl
[2] Semantic Markup for Services - OWL-S (W3C Submission): http://www.w3.org/Submission/OWL-S/
[3] Web Service Modeling Ontology - WSMO (W3C Submission): http://www.w3.org/Submission/WSMO/

adfa, p. 1, 2011.

lightweight, bottom-up methodology such as the SAWSDL[4] W3C recommendation. The WSMO consortium followed up with WSMO-Lite. SAWSDL has successfully been employed to semantically enhance Matching [1], Discovery [2], Selection and Composition [3]. The lack of functioning tools, however, hinders the wide-spread of these technologies [4][5]. This work introduces a tool that provides both experts and potential adopters with a visually-appealing and semi-automated way to semantically annotate services.

2 State Of the Art Comparison

The novelty of the application and some design choices are justified through a comparison with current state of the art. The first two tools, Radiant [6] and WSMO Studio [5], were both implemented as plugins for the Eclipse IDE 3.2 and java 1.5. Hence both are quite outdated and misfunctional. Radiant[5] integrates an ontology tree pane, and WSDL highlighted-text editor into the IDE. Available actions are accessible through buttons on the pane and context menu. However, annotations through Drag'n'Drop or else are not working, contrary to what online documentation shows. The only working function is the SAWSDL namespace addition, for which, unfortunately, the user has to place the cursor in the code, exactly where it should be. Additionally the interface is confusing (buttons that regard services are on the ontology pane), infested with outdated references to the SAWSDL predecessor, WSDL-S (e.g. WSSEM namespace, Action, Effect etc.). WSMO Studio[6] is open-source and available as stand-alone application as well. It was intended to support WSMO-related technology (such as WSML and WSMX). The SAWSDL annotation component was lastly updated in 2007, and hence no working setup was found. The information presented on tables is based on documentation. The third and last tool is the SOWER[7] open source web application. The SOA4All project (2008-2011) created SWEET (Semantic Web sErvices Editing Tool) for RESTful Service annotation (outside the scope of this work) and it's equivalent for SAWSDL and WSMO-Lite, SOWER (Sweet is nOt a Wsdl Editor). Its features are studied in the comparison that follows.

Table 1 considers some general aspects of the tools. Application architectures range from desktop (either as Eclipse plugins or standalone applications) to web. Iridescent was implemented as a platform-independent Java application since, although editing service descriptions is a web-related task, its availability should not be susceptible to internet availability. The storage system solution, found in SOWER, where the user must save files on a semi-organized folder structure was also discarded. This should be an optional and not a necessary step that would require proper organization and authorization. **Table 2** and **Table 3** present ontology and service file handling capabilities. All tools open local files and some from URL. Some tools support simultaneous multiple file handling. Iridescent is the only tool that supports annotation of multiple open WSDLs (in tabs like Radiant and WSMO Studio) from multiple Ontol-

[4] Semantic Annoations for WSDL - SAWSDL (W3C Recommendation): http://www.w3.org/TR/sawsdl/

[5] Radiant online: http://lsdis.cs.uga.edu/projects/meteor-s/downloads/index.php?page=1

[6] WSMO Studio online: http://www.wsmostudio.org

[7] SOWER online: http://stronghold.ontotext.com:8080/wsmoliteeditor/

ogies, also being able to interchange between them (instead of appending them on the same tree like SOWER). Besides, a common case where multiple open files are needed is when WSDL schema (ComplexTypes etc.) is placed in a separate .xsd file (also the practice of the popular NetBeans IDE). In such cases, Iridescent automatically opens referenced files, either ontology imports (also in SOWER) or .xsd files. Finally, Iridescent provides automatic but reversible and visible namespace addition (unlike in SOWER where it is silent and transparent) and many alternatives for SAWSDL authoring using Drag'n'Drop, buttons and context menus (which require less precision), as shown on **Table 4**. All in all, Iridescent takes all the fine attributes from SOWER such as search, multiple ontologies and Drag'n'Drop and extends them providing automations, alternatives and visual enhancements. Its most novel feature is the automated annotation recommendations it introduces, to boost productivity.

Table 1. General Aspects of SAWSDL tools

Aspect	Radiant	WSMO Studio	SOWER	Iridescent
Year	2007	2007	2011	2012
Documentation	✓	×	×	✓
Architecture	Eclipse 3 plugin	Eclipse 3 plugin, standalone	Web app.	standalone

Table 2. Handling of WSDL files in SAWSDL tools

Aspect	Radiant	WSMO Studio	SOWER	Iridescent
Local	✓	✓	✓	✓
Web	×	-	✓	✓
Multiple	✓	-	×	✓
Imports	×	×	×	✓

Table 3. Handling of OWL files in SAWSDL tools

Aspect	**Radiant**	**WSMO Studio**	**SOWER**	**Iridescent**
Local	✓	✓	✓	✓
Web	✓	-	✓	✓
Multiple	×	-	✓ same tree	✓ separately
Imports	×	-	✓	✓
Find	×	-	✓	✓

Table 4. Added functionality in SAWSDL tools

Aspect	**Radiant**	**WSMO Studio**	**SOWER**	**Iridescent**
Namespace handling	✓add (outdated)	-	✓ add	✓ add/remove
Annotation	✓Drag 'n' Drop, ✓Right Click	✓	✓Drag 'n' Drop	✓Drag 'n' Drop, ✓Right Click, ✓Menu
Recommendation	×	×	×	✓

3 Iridescent's Features and Functions

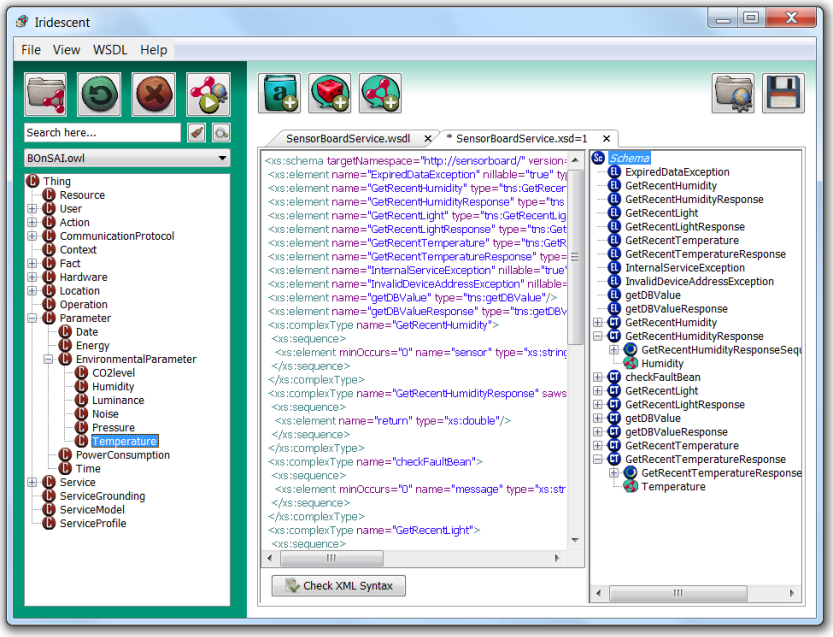

Fig. 1. Iridescent main application window

The Iridescent Java application can be found online along with a manual and learning material[8]. Iridescent introduces its own representation for terms: ontology classes are red, service elements are blue and semantic elements are green. The main window, shown on **Fig. 1** contains an ontology panel (left) and a service panel (right). The former provides all ontology-related actions: open, reload external changes, close, keyword-search for class (with auto-complete), and annotation recommendations for the current ontology and service. Loaded ontologies are displayed in a tree-structure and interchanged using a dropdown box. The service pane similarly provides service-related actions: open, save and add SAWSDL elements (fully supports WSDL 1.1 and 2.0). It opens files in tabs (automatically opens imported files), and for each one displays a tree on the right and the corresponding code for the selected node and its children on the left. Open ontologies and services are saved and opened automatically on the next session. The menu bar repeats some of these functions, and additionally hosts the theme chooser, legend and about dialogs. To ensure that all users can instantly find functions, there are many alternative ways to access each one:

- Add/Remove SAWSDL namespace: from the service panel button, WSDL menu or by right-clicking the WSDL description/definition node.

[8]Iridescent online: http://lpis.csd.auth.gr/people/thanosgstavr/applications/iridescent.html

- Add Schema Mapping (Lifting or Lowering): from the service panel button, WSDL menu or by right-clicking the target Service element on the tree.
- Add Model Reference: from the service panel button, WSDL menu or by right-clicking the target service node. Additionally via Drag 'n' Drop of an ontology class on the desired Service element, or via recommendations.
- Remove any SAWSDL element (namespace, Model Reference or Schema Mapping): right-click target element.

The recommendation function invokes the corresponding dialog (**Fig. 2**) where the active ontology and service are searched for string name matches. Three algorithms are provided: the normalized Levenshtein distance, fuzzy string search (or approximate matching. Calculates the minimum Levenhtein distance for all substrings of two strings), and Common Words. The latter is a custom algorithm based on the observation that ontology and service node names are concatenated phrases where each word begins with a capital letter e.g. Immune_System or ImmuneSystem. It splits names at capital letters and looks for common words in both phrases. Hence, it is especially efficient for phrases that contain the same word but one is not the substring of the other. Results can be filtered using a rating threshold and/or a keyword and sorted. Multiple selections are instantly committed. In the use case scenarios provided with Iridescent (the ontology used is BOnSAI [7] and the services of the Smart IHU project) inputs and outputs (namely at ComplexType level) of operations are annotated. There are numerous matches: "Temperature" – GetRecentTemperatureResponse and "Humidity" - GetRecentHumidityResponse (fuzzy string search or Common Words, **Fig. 2**), PowerConsumption – ReadPowerResponse and SwitchAction - SwitcnOn and SwitchOff (Common Words). The resulting annotations can of course be used in various AI client-applications e.g. expert systems, planning etc.

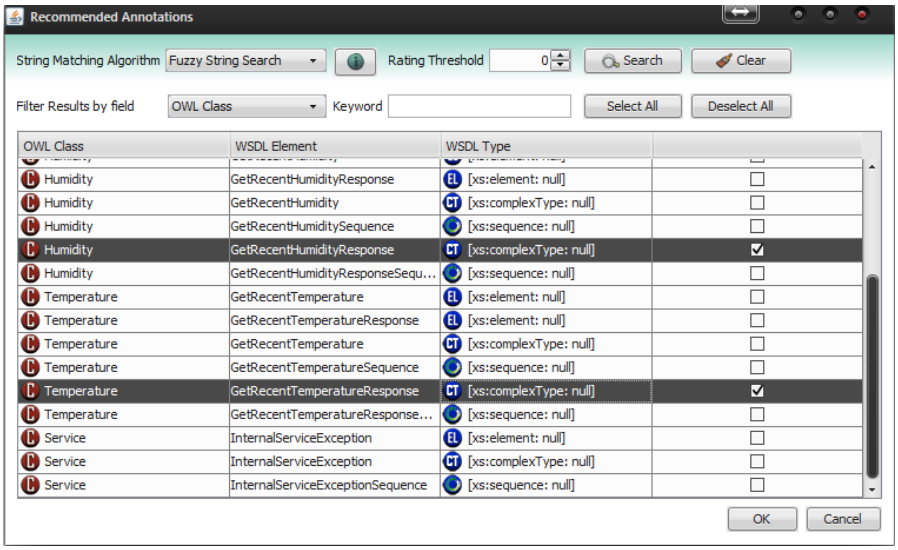

Fig. 2. Recommendations in SensorBoard scenario

4 User Evaluation

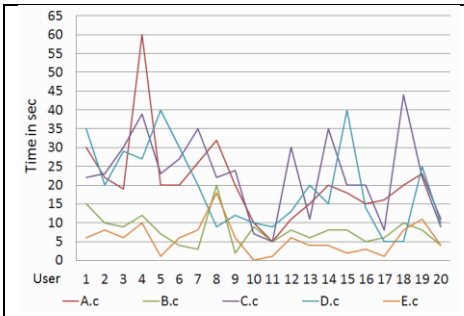

Fig. 3. Times for each task on code view

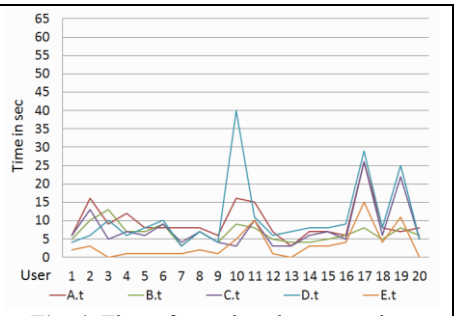

Fig. 4. Times for each task on tree view

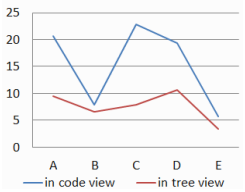

Fig. 5. Averages of times for each task

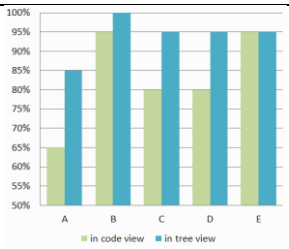

Fig. 6. % of correct answers for each task.

Twenty users were engaged in a two-phase evaluation of the application: Seventeen computer science students, two of which had experience with WSDL and one with SAWSDL, a PhD student, a MSc student and one user with no computer science background. In the first phase, the effectiveness of the representation model (i.e. tree and code layout) was evaluated. Users were assigned five tasks: to count a WSDL's A) messages B) elements C) ComplexTypes D) SAWSDL attributes and E) Model References once in tree view and once in (highlighted) code view in random order as a measure of how effectively users perceive data. All users performed similarly, as shown on **Fig. 3** for code and **Fig. 4** for tree view, except a few outliers, the very fast #11 and the slow #4 and #10. Contrary to expectations, skilled users (SAWSDL/WSDL developers #3, #1, #2) and the non-computer science student (#17) performed on average. **Fig. 5** confirms that code is in average much more time demanding (15s - except tasks B, C that were easy). Tasks in tree view take much less (7s), and almost always the same amount of time (7s standard deviation for code, 3s for tree view). . **Fig. 6** shows that meanwhile answers are faster and more correct in tree view. The more time-consuming tasks were also the most poorly answered (A,C,D in code). In a rating session, users answered that they prefer to view both tree and code (100%), rated the representation with 4.7/5 and some suggested to have node categories instead of trees. In the second phase, users performed timed tasks to measure the tool's productivity and usability. They had to locate an ontology class manually (task I) and using search (II), and annotate using Drag 'n' Drop (III) or menu (IV). The same annotations were then committed via recommendations (V).

Table 5 shows that all tasks are stable (low deviation except task I, manual search). Also a manual annotation (task I or II and III or IV) takes on average at least 9s. Two annotations using recommendation take 13s in average. The time difference grows exponentially since more recommendations can be selected instantly. Finally users replied that they would use both menu and Drag'n'Drop (65%), or Drag'n"Drop only (35%). They rated all of its functions above 4/5, 4.8/5 for its usefulness, and some suggested keyboard shortcuts, more functions for code, optional split/tree/code view and function to expand/collapse all tree nodes.

Table 5. Statistics of functionality evaluation

Metric\Task	I	II	III	IV	V
Average time (sec)	19	5	4	15	13
St. deviation (sec)	16	3	2	5	4

5 Lessons Learned

The evaluation sessions show optimistic results: the users seem eager to adopt a graphic tool and are fond of the intuitive representation, functions. Having its own visual, icon representation seemed very effective for new users. They seemed especially excited with the automatic recommendation function since it adds an automation effect. Suggestions from evaluators (GUI enhancements such as the code/tree/split view) as well as the community are bound to be implemented (an online form is provided for that). An important goal is more experimentation with real world data (possibly also more string matching), and exposing the tool to the wider public to receive feedback.

Acknowledgement. The authors would like to thank Theodoros Mylonides for his contribution.

References

1. Tran, V.X.; Puntheeranurak, S.; Tsuji, H.: A new service matching definition and algorithm with SAWSDL, Digital Ecosystems and Technologies, 2009. DEST '09. 3rd IEEE International Conference on , vol., no., pp.371-376, 1-3 June 2009 doi: 10.1109/DEST.2009.5276750
2. Iqbal, K., Sbodio, M. L., Peristeras, V., Giuliani, G.: Semantic Service Discovery using SAWSDL and SPARQL, Semantics, Knowledge and Grid, International Conference on In Semantics, Knowledge and Grid, 2008. SKG '08. Fourth International Conference on, Vol. 0 (2008), pp. 205-212, doi:10.1109/SKG.2008.87
3. Lécué, F., Gorronogoitia, Y., Gonzalez, R., Radzimski, M., Villa, M.: SOA4All: An Innovative Integrated Approach to Services Composition, ;in Proc. ICWS, 2010, pp.58-6
4. Valle, E. D., Niro, G., Mancas, C.: Results of a Survey on Improving the Art of Semantic Web Application Development, ISWC 2011
5. Dimitrov, M., Simov, A., Momtchev, V., Konstantinov, M.: WSMO Studio – a Semantic Web Services Modelling Environment for WSMO, ESWC 2007: 749-758

6. Gomadam, K., Verma, K., Brewer, D., Sheth, A. P., Miller, J. A.: Radiant: A tool for semantic annotation of Web Services, The Proceedings of the 4th International Semantic Web Conference (ISWC 2005) Galway, Ireland - Demo Paper, 2005
7. Stavropoulos T. G., Vrakas D., Vlachava D., Bassiliades N., BOnSAI: a Smart Building Ontology for Ambient Intelligence, in the proc. of WIMS 2012, Craiova, Romania

Appendix: Addressing Challenge Criteria

✓ The application is an end-user application. Not especially intended for general Web users but for Service developers, providers and even some users. Both experts and non-experts are able to use it.
✓ The information sources used (ontologies and services) are under diverse ownership. Apart from the limited examples provided for space, virtually any WSDL service and ontology on the Web can be used (e.g. BioPortal etc.) They are also heterogeneous. Only semantic annotations can disambiguate the syntactic WSDL service descriptions. They also contain substantial quantities of real world data. Services in the examples belong to the Sensor Web (i.e. return real-time environmental data). Any industrial or other online service can also be used.
✓ The meaning of data plays an important role to service annotation. The meaning is represented using Semantic Web technologies (mainly OWL/RDF/SAWSDL).
✓ The tool uses semantic data to annotate services (not possible without semantic data), that enables semantic services (irreplaceable by syntactic descriptions). Manipulation of ontologies and descriptions allows automatic annotations.

Additional Criteria.

✓ The application provides an attractive and functional interface (desktop application accessible on the web, platform independent (Java) requires no installation).
✓ The application is scalable as it can handle large files. Also in terms of design, users can search or use recommendations to navigate large files.
✓ Rigorous evaluations have taken place that demonstrate the benefits of semantic technologies, or validate the results obtained. User times and ratings show the effectiveness usability and usefulness of the tool. See the evaluation section.
✓ Novelty, in applying semantic technology to a domain or task that have not been considered before: automatic annotation in a tool, and minor other functions have never been provided before.
✓ Functionality is different from or goes beyond pure information retrieval. The main purpose of the tool is not information retrieval but development.
✓ The application has clear commercial potential, as a tool for industrial services (but not a large existing user base).
✗ Contextual information is not used for ratings or rankings
✗ Multimedia documents are not used in some way
✗ There is no use of dynamic data (e.g. workflows), nor in combination with static information
✓ The results are as accurate as possible. The recommendation results have a rating that can be filtered.
✗ There is no support for multiple languages and accessibility on a range of devices