Iridescent: a Tool for Rapid Semantic Annotation of Web Service Descriptions

Thanos G. Stavropoulos, Dimitris Vrakas and Ioannis Vlahavas

Computer Science Department
Aristotle University of Thessaloniki
Thessaloniki, Greece
{athstavr, dvrakas, vlahavas}@csd.auth.gr

School of Science and Technology
International Hellenic University
Thessaloniki, Greece

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 number of different methodologies and the lack of tools, which both force them to acquire expert knowledge and commit to exhausting manual labor. This work proposes a novel functional and user-friendly graphical tool, named Iridescent, intended for both expert and non-expert users, to 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 points and requirements. The tool's functionality is presented along with usage scenarios that demonstrate how the tool and SAWSDL enable Intelligence in an Ambient Intelligence environment. Finally, Iridescent was methodically tested for its usability and evaluated by a range of both expert and non-expert users.

Categories and Subject Descriptors

D.3.1 [**Programming Languages**]: Formal Definitions and Theory – *semantics, syntax*
I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods – *representation languages.*
H.3.5 [**Information Systems**]: Online Information Services – *Web-based Services.*

General Terms

Algorithms, Management, Design, Experimentation, Human Factors, Standardization, Languages.

Keywords

Semantic Web Tools; Web Services; Semantic Web Services; Service-Oriented Architecture

1. INTRODUCTION

The contemporary use of the Web has currently penetrated and altered everyday life. Traditionally, the Web is an infinite source of knowledge (data) that people can look up and exploit. Popular demand and technology developments have recently led to a new way of interacting with the Web, the so-called Web Services. Web Service technology allows users to get things done, instead of simply looking up data. Service interoperability is supported by well-defined web standards, such as the Web Service Description Language – WSDL [1], which provides a universal, platform independent interface description. However, WSDL enables syntactic interoperability only, as it type-defines inputs and outputs of service operations. That is a main obstacle in enabling intelligent Web Service clients such as Intelligent Agents. The same issue has long ago been encountered in the Web of data. Vast amounts of syntactically described data e.g. mostly in HTML are scattered across Web pages, while their meaning can only be appreciated by human users. The Semantic Web technologies have emerged to tackle this obstacle and semantically describe data using ontologies i.e. taxonomies of inter-related concepts.

Consequently, much work during the last decade strives to realize the synergy of the two fields, Semantic Web and Web Service technologies, and bring about Semantic Web Services. The efforts initially focused on top-down approaches, which are essentially ontologies themselves: the so-called upper ontologies for services. OWL-S[1] Semantic Markup for Services is such an ontology that describes a service's functional and non-functional aspects and the grounding of the service to running instances. The Web Service Modeling Ontology WSMO[2] is an alternative approach follow-up. More recently, efforts were oriented towards lightweight approaches, the bottom-up approaches. The initial and most popular one, Semantic Annotations for WSDL, SAWSDL[3], became a W3C recommendation. It enables lightweight annotations directly over WSDL. SAWSDL introduces two kinds of extension attributes: Model Reference and Schema Mappings. Model References are the annotations themselves. They are references to ontology concepts/classes. Schema Mappings are either Lifting or Lowering. They specify transformations of XML schema into ontological constructs (Lifting) and vice versa (Lowering). Being a bottom-up approach, the SAWSDL solution prioritizes running instances of services and minimum complexity. It has already successfully been employed for Service Matching [2], Discovery [3], Selection and Composition [4]. As the lightweight approach is gaining momentum, the WSMO consortium has followed up with a lightweight version of WSMO, WSMO-Lite, which builds upon SAWSDL. In this work we strive to build a minimum solution for enabling intelligence on the client side, using just SAWSDL. Thus, minimizing complexity and maximizing interoperability.

In parallel with prototype development, there have also been attempts to build tools to support and disseminate their use. Besides, the advancement of the Semantic Web itself is currently tackled by the narrow spread of its technologies, as both users and industry still hesitate to use common, agreed-upon models. The Semantic Web

[1] OWL-S W3C Submission: http://www.w3.org/Submission/OWL-S/

[2] WSMO W3C Submission: http://www.w3.org/Submission/WSMO/

[3] SAWSDL W3C Recommendation: http://www.w3.org/TR/sawsdl/

Service tools, especially, often become neglected after their initial release, hindering expert and non-expert users alike. Up to date, there are few working tools for Semantic Web Services, puzzling potential users and hindering the adoption of these technologies. Expert users, who are already able to manually create the descriptions, coding the XML files, are also discouraged by the limitations of existing tools. Namely, expert users are able to make batch changes on raw code fast, but not on graphical tools. Thus, tools slow down their productivity.

This work introduces a novel functional tool, named Iridescent, for adding and editing semantic annotations to Web Service descriptions. The tool aids users in two ways: it facilitates manual labor by providing a user-friendly visual alternative to coding and it semi-automates the process by suggesting annotations, boosting productivity. The latter function especially, makes batch changes easier and faster. After all, the tool's uttermost purpose is to disseminate the use of Semantic Web Service technologies overall. To reveal current limitations and drawbacks, a state-of-the-art review has been carried out. Based on that, this work presents advancements made in usability and functionality. To demonstrate how the tool and the SAWSDL standard primitively enable intelligence, usage scenarios in the Ambient Intelligence domain are showcased. Finally, the effectiveness and usefulness of Iridescent was evaluated and measured by a set of testers.

The paper is structured as follows: in the next section existing solutions are surveyed and compared to justify the motivation and novelty of this work. The third section presents aspects of the proposed tool in detail, describing the motivation behind it, its requirements and its functionality. Usage scenarios on existing services along with a set of Ambient Intelligence client applications are presented on the fourth section. The fifth section presents usability evaluation and testing results. Finally, future work and conclusions drawn are listed on the corresponding sections.

2. STATE OF THE ART COMPARISON

This section reviews the state-of-the-art in SAWSDL annotation tools during the last decade. Throughout the survey, potential flaws and improvements are investigated. Each tool is presented in a separate subsection, while the final subsection presents a comparison between state-of-the-art and the proposed tool.

2.1 RADIANT

Radiant [5], is historically the first tool for SAWSDL annotation of web service descriptions. It is a plugin for the Eclipse IDE[4] (not a standalone application) and can be found online[5] along with a short manual and a video tutorial. Radiant mainly incorporates an ontology viewer pane and a WSDL text editor into Eclipse. A WSDL tree-viewer is already provided by Eclipse. First of all, the user has to create a project and add one or more WSDL files, from local storage only. Then, he has to switch to an Eclipse perspective, to view the ontology panel. Users can load a single ontology (one at a time) from a local file or URL into the ontology panel, which displays it in a tree structure. That panel also hosts the toolbar buttons for all annotation operations. The WSDL Editor naturally shows WSDLs in highlighted text form. The first step for the annotation is adding the SAWSDL namespace import. To do so, users have to manually place the cursor in the exact point of wsdl:definitions where they want to insert it within the text, right click the ontology root class and choose the appropriate option. This process is completely manual and not intended for non-developers. Its steps may also be confusing as the SAWSDL has nothing to do with the ontologies root class. In Radiant, users can also add a namespace definition i.e. prefix for the ontology at hand, which is a plus, as it helps shorten Model Reference URIs.

However, the annotations themselves can utterly not be added. Contrary to what the user guide and video show, annotations cannot be added neither by Drag 'n' Drop nor by right clicking classes. A second major issue is the confusing environment infested with left-over references to the predecessor of SAWSDL, WSDL-S. The terms "WSSEM namespace", "Action", "Precondition", "Effect" found on almost all context menus and buttons in Radiant, all belong to WSDL-S. The plugin was last modified on 29/5/2007 and was intended for Eclipse 3.2 and Java 1.5.

2.2 WSMO STUDIO

WSMO Studio is an IDE that integrates various tools for Semantic Web Services. It was initially presented in [6] and can still be found online[6] as an open source project. Like Radiant, it comes in the form of a plugin for the Eclipse IDE, but also as a standalone application. The motivation behind WSMO Studio was to support the WSMO technologies, but it substantially expanded to lightweight annotations with SAWSDL. Specifically it is comprised of a WSMO editor, a WSML reasoner and a WSMX adapter. WSMO Studio was developed between 2005 and 2009 (last update of the SAWSDL plugin on SourceForge was in 2007) again for Eclipse 3.2 and Java 1.5. Unfortunately, the authors could not find any working setup (standalone or plugin), nor any online documentation as to what it does exactly. The features presented in the comparative subsection below are based on screenshots from the software.

2.3 SOWER

SOWER or "WSMO-Lite Editor" is an open source web application that supports SAWSDL and WSMO-Lite lightweight service annotations. During the SOA4All project, which ran between March 2008 and February 2011, many applications have been developed, among which SWEET (Semantic Web sErvices Editing Tool) and SOWER (Sweet is nOt a Wsdl EditoR). SWEET is a RESTful-Service annotation tool. It produces hRESTS and MicroWSMO annotations on HTML descriptions. It is very similar to SOWER in design and usability, although slightly more sophisticated to better support the nature of the different formats. RESTful service annotation is outside the scope of this paper, thus, only SOWER is taking part in the comparison subsection.

[4] Eclipse IDE: http://www.eclipse.org/

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

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

SOWER can be found online at the project's website[7]. Instead of documentation, an explanatory video covers all of its functions. The graphic interface is good-looking, friendly and intuitive, as all operations are carried out via Drag 'n' Drop. The user can open multiple ontologies, that are inserted into a single tree (displayed on the left pane) and a WSDL service description (displayed on the upper right pane in tree form and the lower right pane in text form). Both types of documents can be opened from a URL and from a special kind of storage service used in many SOA4All tools. In the background, SOWER silently adds the SAWSDL namespace, which is desirable in most cases. However, this may confuse the user as he does not realize whether it was added or not. Additionally, he cannot remove it without viewing and manipulating code. Users can keyword-search the ontology for concepts and Drag 'n' Drop them on WSDL elements to annotate them. The resulting file can be saved to either iServe (the project's special provisioning platform) or to the special storage service (repository) from where it can later be downloaded.

2.4 COMPARATIVE OF TOOLS

A comparative of state-of-the-art tools and the proposed solution i.e. Iridescent is presented in this subsection. Table 1 considers some general aspects of the tools. Their application architecture ranges from desktop (either as Eclipse plugins or standalone applications) to web i.e. SOWER. We argue that although editing web service descriptions is a web-related task, internet availability should not be a requirement. After all, text, editors and even IDEs would not be so usable, were they online only. We also discarded the storage system solution, found in SOWER. SOWER users must explicitly save files on a semi-organized folder structure where everybody can toss their files in, and then download them for personal use. This should be an optional and not necessary step that would also require proper organization and user authorization for the file structure. It may be considered as a future addition, according to user demand.

Table 2 and Table 3 summarize the ontology and service file handling capabilities of the tools. All tools can open files locally and some also from URL. Such files are indeed primarily online, but service engineers actually work locally as well, during testing, work in progress or during unavailable connections as mentioned above.

Another issue is loading multiple files at the same time. Multiple open OWL files are useful for annotating from various sources and interchanging between them without repeating the opening process. SOWER opens multiple ontologies but appends all their classes on the same taxonomy tree. Hence, it potentially becomes too large to handle (although SOWER provides a handy search function). For that purpose, Iridescent separates the ontologies loaded and provides a dropdown box to quickly switch between them.

Multiple open WSDL files are also useful, for annotating multiple files and interchanging between them. Besides, a common authoring practice, also followed by the auto-generated WSDLs by the popular NetBeans Java IDE, is separating schema in a separate .xsd file. In that case, opening multiple files is a prominent need as descriptions are practically split in two. Early tools, being Eclipse plugins, inherently support opening multiple WSDL files, in tabs. Iridescent supports multiple files in tabs as well, but additionally, it automatically opens referenced files within a file i.e. schema files.

Likewise, OWL files contain references to other files as well, i.e. imports. Most existing ontology editors do automatically open these

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

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 app.

Table 2. Handling of WSDL files in SAWSDL tools

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

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 (silent)	✓ add, remove
Annotation	✓Drag 'n' Drop, ✓Right click	✓	✓Drag 'n' Drop	✓ Drag 'n' Drop, ✓ Right Click, ✓ Menu
Recommendation	×	×	×	✓

imports as they consist a necessary part of the ontology. SOWER and Iridescent both open these imports. However, SOWER does not offer the capability of locating a missing import, neither locally nor online. Iridescent offers this feature, in a manner similar to the one in Protégé, one of the leading ontology editors. All in all, Iridescent strives to dynamically resolve all file imports and bridge the gaps for annotating multiple files from multiple sources.

Finally, functionality and added value aspects are presented on Table 4. Quite similarly, all tools display WSDL code, WSDL hierarchy trees and OWL hierarchy trees. Traditionally, annotations happen via Drag 'n' Drop of OWL classes onto WSDL nodes. Although this is indeed the most intuitive manner, it requires accuracy and exploring. Iridescent gives access to all operations through various ways (the simplest being buttons, context-menus and toolbar buttons and the more sophisticated recommendation function). That way it provides a thorough view of the application's functionality at a glance requiring less exploring. Besides, the user evaluation survey performed after the tool's completion, verified the importance of providing alternative methods. It revealed that 60% of users would use both Drag 'n' Drop and context menus regularly. Another issue is the SAWSDL namespace import. In Radiant, this import has to be inserted in the text. In SOWER it happens silently, which can be unintentional. Iridescent makes the SAWSDL namespace visible and manageable (reversible) like every other change on the description files. Finally, Iridescent introduces annotation recommendations to accelerate and facilitate the annotation process. Concept names are matched against WSDL nodes and users can select which of these recommendations to actually commit. This is a novel step towards annotation automation in SAWSDL editors and SAWSDL annotation overall.

3. IRIDESCENT

This work introduces the Iridescent tool[8], thoroughly described in this section in terms of its functional and non-functional requirements. First of all, the motivation for engineering the tool was the lack of tools and methodologies [7], hindering the dissemination of semantic web technologies, and especially semantic web service technologies. Ever since the arousal of the Semantic Web Service notion, with top-down approaches OWL-S and WSMO, the lack of proper tools was eminent [6] [8]. Some work tries to automatically generate (top-down) semantic descriptions, acknowledging both their complexity and lack of tools [9]. High complexity led to the rise of light-weight annotations. SAWSDL lightweight annotations have already been proved to be suitable for enhancing the major stages of a web service's lifecycle; namely Matching [2], Discovery [3] and Composition [4] and enabling intelligence on the client side. The state-of-the-art survey has shown considerable room for improvement in enhancing accessibility and ease-of-use for both experts and non-experts in such a tool.

3.1 REQUIREMENTS

A set of requirements for the proposed tool was formed based on the state-of-the-art survey and is listed below. In each case, trade-offs and choices made are explained.

Compatibility

The tool must be platform independent, as it targets a wide audience; research, industry and individual developers. The Java platform was chosen to support this aspect.

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

Availability

Being intended for engineers, the tool, as well as resources for development, must always be accessible. For online applications, availability is restricted by connectivity constraints and the hassle of uploading and downloading files. Therefore, the tool was implemented as a desktop application, much like IDEs. Still, it requires no installation, is platform independent and not computationally demanding at all. The resources i.e. ontologies and service descriptions, can be loaded from both the Web and local storage.

Usability

The application must be as user-friendly as possible even for users with no Semantic Web background. The interface's layout and design should be intuitive and fast to learn. Iridescent's layout is structured like most similar tools. An ontology panel holds everything that has to do with ontologies in one place and a service panel holds service-related files and features. The two are separated by distinctive colors (there are also color themes to choose from). A novel representation scheme has been constructed to facilitate visual discriminations and learning. The pattern in general is that all SAWSDL notions have cyan icons, WSDL notions have blue icons and ontology classes have red icons. A short legend for this representation is available on the help menu and shown on Figure 1. For intuitiveness and demanding less exploring, all basic operations can be accessed in many ways. Drag 'n' Drop may be the most instinctive and fast, but not an obvious option for some users, especially the new ones. Hence, annotations can also be applied via a toolbar button or right click (context menu). Usability is also enhanced by automations, such as automatically opening imported files both for ontologies and services. Finally, multiple open files simultaneously are a must for multitasking and a standard amongst most computer applications.

Extensibility

The tool must be extensible to future technology and format developments. Currently, the ontology tree supports OWL/RDF ontologies but can be extended upon demand. Any format of schema mapping is allowed (as only the URL of the transformation is required to specify one).

Documentation

Helpful and up-to-date documentation is needed for all applications. Learning material for Iridescent currently available online comprises of a presentation and the manual. The presentation, also used to educate beta testers, initially teaches basic knowledge of the Semantic Web Service notions and secondly guides one through three SAWSDL creation scenarios, explaining the tool's complete functionality.

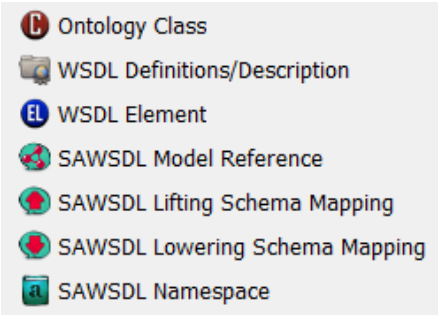

Figure 1. Legend of icon representations in Iridescent

Functional Requirements

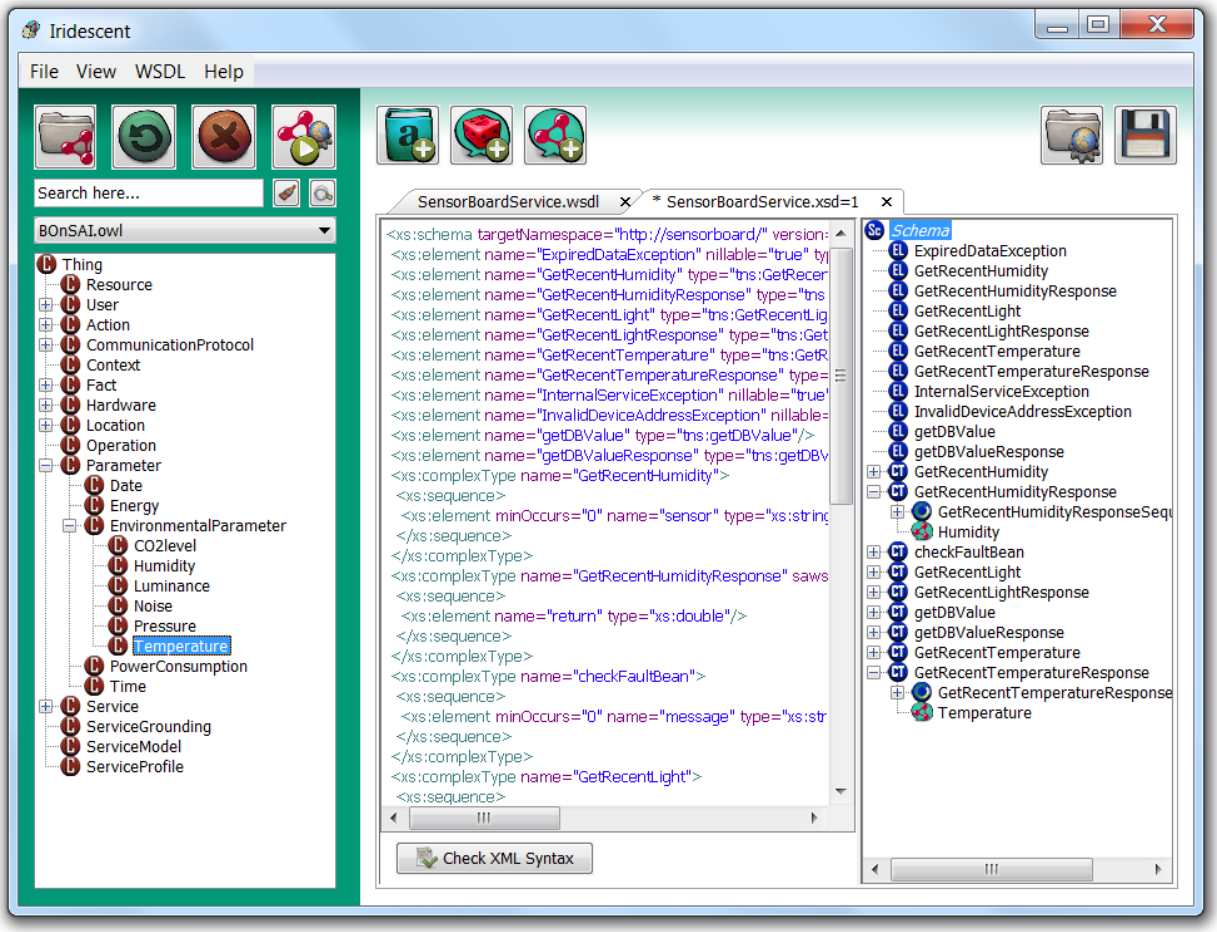

Figure 2. Iridescent main application window

Using the tool, a user should be able to fully create and edit SAWSDL files. Specifically, given well-formed ontologies and WSDL or SAWSDL service descriptions, he should be able to output well-formed SAWSDL files. SAWSDL, in essence, introduces three new elements (extension attributes) into the WSDL: Model References, Schema Mapping and the SAWSDL namespace import. The user must be able to add or remove any of the above without restriction, exactly as in manual coding. Added value facilities, such as the search function, should be provided. Furthermore the tool must provide a way to accelerate batch annotations. Hence, recommendations by matching have been introduced, to aid the user with annotation ideas, select and insert many annotations at once.

3.2 FUNCTIONALITY

Driven by the above requirements, Iridescent was developed to support all editing functionality for SAWSDL files and provides a complete graphic alternative to coding. Figure 2 depicts the main frame of the Iridescent Java desktop application. The frame is divided in two parts, made distinct by the contrasting colors: the Ontology Panel on the left and the Service Panel on the right. The Ontology Panel provides buttons for ontology-related functions such as Open, Close, Reload and Recommendations (from left to right). Ontologies are loaded on the Ontology Panel in a tree structure, automatically inserting imported ontology classes in the same tree structure. Multiple open ontologies can be interchanged using a dropdown box. Ontologies can be refreshed (reloaded) in case they are edited externally (e.g. in Protégé). The search box provided can keyword-search concepts in the ontology and is enhanced with Auto-Complete.

To enable automatic annotation we considered that humans use primitive data themselves to identify a desired match (ontology class and service node) which is no other than their string names. Then, they either compare their string names for a match or use other cognitive complicated mechanisms such as pure desire or prior knowledge about the domain ontology and/or the service. Hence, we concluded that the tool could at least provide the string matching feature, using primitive string matching algorithms.

Returning to the tool's interface, the Recommendation button is enabled if and only if both an ontology and a service file are open in the respective panels (at least one of each). Upon invocation, it shows a dialog such as on Figure 3, guiding the user to automatically match the active ontology concepts with the active service nodes. Three algorithms are provided for that purpose: the Levenshtein distance, Fuzzy String Search (also found as "Approximate Matching") and Common Words. Fuzzy String Search calculates the minimum Levenshtein distance for all substrings of the two strings. The latter is a custom algorithm introduced in this work. It is based on the observation that ontology and service node names traditionally are concatenated phrases in which each word begins with a capital letter e.g. Immune_System or ImmuneSystem. The algorithm splits these

phrases at capital letters and looks for common words among them. As a result, it is especially efficient for phrases that contain the same word but one is not the substring of the other (where the other two algorithms are just as efficient). For example, in the use case scenario described on the next section, both Fuzzy String Search and Common Words find the matches Temperature – GetRecentTemperatureResponse and Humidity - GetRecentHumidityResponse, PowerConsumption – ReadPowerResponse. However, Common Words only finds the matches SwitchAction - SwitcnOn and SwitchOff. All algorithms are normalized and exhaustive (match all concepts with all service nodes). Matching results, shown on a table, can be filtered using a rating threshold and/or a filtering keyword. Results can also be alphabetically sorted traditionally by clicking a column header. The user can select as many matches as he desires and commit them at once to the SAWSDL, facilitating batch operations.

The Service panel, likewise, hosts all service-related functions. The buttons on the top-right are used to open WSDL or SAWSDL files, from the web or local storage, and save the active file. Upon opening, the service file elements are loaded in a tree hierarchy on the right, showing WSDL elements in blue and SAWSDL elements in cyan, following the icon legend of Figure 1. To make its presence visible and manageable, the SAWSDL namespace is shown as a tree node too, although it is only an attribute instead of an element. On the middle pane, highlighted XML text of the selected service node is displayed. A button provides XML syntax validation. The list of open files, ontologies and services, is saved internally for the application to restore previous sessions at startup.

The buttons on the upper left of the service pane provide SAWSDL authoring operations. From left to right, they allow adding or removing the SAWSDL namespace, adding a Schema Mapping and adding a Model Reference, anywhere in the SAWSDL (target nodes are selected on a pop-up dialog). The "Add Namespace" button, changes to "Remove Namespace" when the namespace is present and changes back when it is not present. All changes are reversible since the user may even want to revert back to plain WSDL. The Schema Mapping and Model Reference buttons are enabled only if an Ontology is open (at least one) and refer to the active ontology and service. Considering the Schema Mappings, the user is solely to provide the URL where the transformation resides. However, there are thoughts to make the tool penetrate into this process and assist in the actual specification of the transformation itself. A hesitation for this is the upload process of the transformation (as it needs to be online) which raises architecture issues (it expands beyond a simple desktop application), and thus, is considered as future work. Continuing on the GUI, the menubar provides again some of these functions, plus a theme chooser (each theme is a different colour pattern), the icon legend, and the "About" dialog.

WSDL 1.1 and WSDL 2.0 are thoroughly supported according to the SAWSDL specification. Note, that the only difference between the two versions is that in 1.1, WSDL operations are annotated using the attrExtensions element to insert the model reference as a child element. All other 1.1 and 2.0 elements are annotated as usual using the sawsdl:modelReference attribute.

To sum up, there are many alternative ways to accomplish a single task in Iridescent. That ensures that even novice computer users can find a way to perform an operation without even reading the manual or long exploring. More specifically:

- To add the SAWSDL namespace:
  use the service panel toolbar button, the WSDL menu or right-click the WSDL tree root node (description/definition).
- Add Schema Mapping (Lifting or Lowering):
  use the service panel toolbar button, the WSDL menu or right-click the target WSDL tree node. Select the appropriate option on the context menu.
- Add Model Reference:
  use the service panel toolbar button, the WSDL menu or right-click the desired node. Alternatively, Drag 'n' Drop the desired source ontology concept i.e. Class on the target Service element, or use the Recommendations function.
- Remove any SAWSDL element (i.e. the namespace, a Model Reference or a Schema Mapping):
  right-click the target element tree-node and select Remove. Use the toolbar button for the namespace.

4. AMBIENT INTELLIGENCE USE CASE

The purpose of the use case scenarios presented here is twofold: to demonstrate Iridescent's functionality in practice and how both the tool and the SAWSDL standard enable intelligence on the service client side. For the latter, the documents created during these scenarios are used directly to interface with existing implemented services in an Ambient Intelligence setting. All scenarios are available online in the form of a step-by-step presentation and were also followed by testers during the beta testing cycle of the application.

The web services are deployed at a Smart Building environment by the Smart IHU project, which targets energy savings and quality of life through Ambient Intelligence [10]. A web service middleware exposes the functions and data of various smart devices (sensors and actuators) scattered around the Smart Building. As similar Web Service-middleware setups are common in such environments, the use of semantic web services is a common practice as well. The majority of approaches prefer the more sophisticated top-down approach, OWL-S, such as in [11], up to 2008 [12]. As lightweight annotations emerged, a more recent approach supports SAWSDL as well [13], mapping from SAWSDL to the system's own XML-description language. We investigate enabling intelligence with SAWSDL and no additional layer of complexity, as it already suffices to enhance all the stages of a service's lifecycle (Service Matching, Discovery, Selection and Composition).

The ontology used is BOnSAI (Smart Building Ontology for Ambient Intelligence) [14]. BOnSAI extends existing ontologies with concepts for Smart Buildings in general (e.g. "Environmental Parameter", "Location" etc.). The concepts added for the Smart IHU case in particular suit the devices found in this deployment e.g. "SmartPlug", "SmartClamper" etc. The available operations are grouped in five services, according to the Device type that they expose: SmartClamperService, SmartPlugService, SensorBoardService, Z-WaveService and ITService.

The aim of the scenarios is to semantically describe the input and output types of operations and specify transformations. BOnSAI, SmartPlugService and SensorBoardService are opened either from URL or from local files. As the services were developed in Netbeans, their complementary xsd schema files are also opened automatically. Concepts are located either manually or using the search box and used to annotate the service Complex Types. As an example, the "PowerConsumption" concept is assigned as the return type of the ReadPower operation (i.e. ReadPowerResponse Complex Type). The Recommendation function is used for two annotations: "Temperature" at GetRecentTemperatureResponse and "Humidity" at GetRecentHumidityResponse. The exact Recommendation frame for

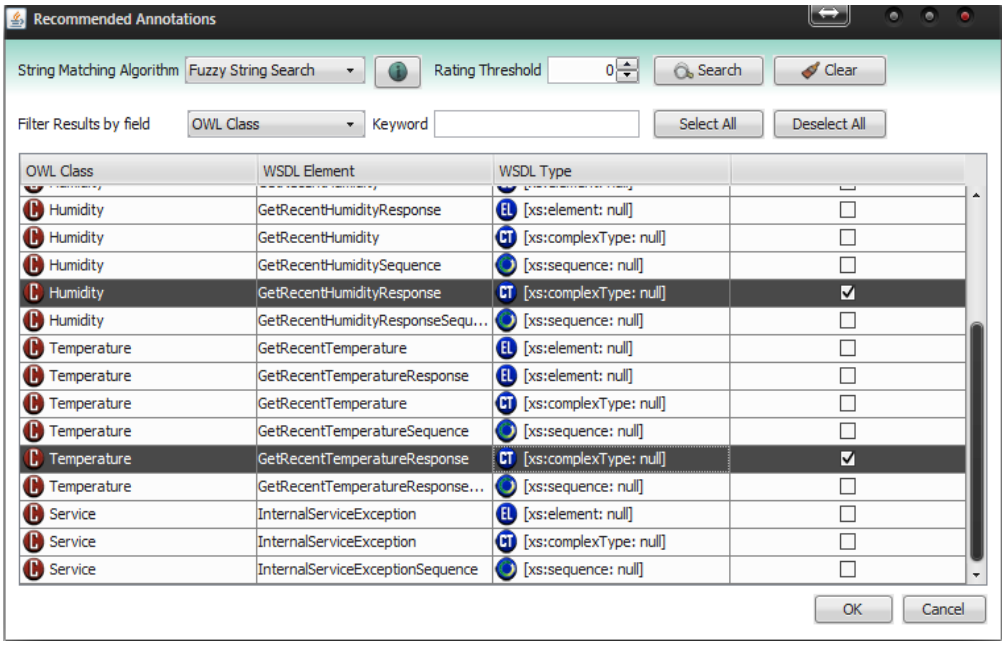

Figure 3. Annotation recommendations in Iridescent, sensor board scenario

this case is shown on Figure 3. At all times, the user observes changes on both tree and code. One scenario includes inserting a Lifting Schema Mapping transformation to RDF on a W3C service, to demonstrate the tool's generality.

There are many ways to exploit the resulting semantically-annotated services e.g. for Discovery, Matching, Selection and Composition. Thoroughly presenting client applications is outside the scope of this work. However, a brief summary is given here to demonstrate the kind of possibilities that present themselves. First of all, a web application was implemented to allow specifying queries for web service descriptions based on semantic inputs and outputs. Subsequently, it matches them with services in a repository (new services can be loaded on the repository). The matching algorithm is weighted and also incorporates reasoning to find substitute classes. The results can be exploited by any client in order to operate the devices of the Ambient Intelligence system.

More generally, the annotations can be exploited in any kind of A.I. application. A JADE agent was developed to maintain a knowledge base about the state of the Ambient Intelligence environment. A GUI is used to insert rules that operate the smart building's semantic services based on SAWSDL information. Due to their semantic nature, services can be added or removed, supporting the dynamicity of such environments. The agent uses JESS to find rules that are triggered or fired up and invokes the proper semantic services.

5. USER EVALUATION

A sample of 20 evaluators was engaged in a two-phase usability test and evaluation of Iridescent. Specifically, 17 evaluators were undergraduate computer science students, from which 2 had developing experience with WSDL and one with SAWSDL. The rest of the evaluators were: a PhD student, a MSc student, both specialized in A.I., and one with no computer science background whatsoever. It was hard to find semantic web experts evaluators at this stage of development but on the other hand, this less-skilled set of users can demonstrate how usable the application is for non-expert users.

The first part of the evaluation concerned only the representation model adopted in Iridescent; the tree-and-code layout. To measure the effectiveness of the representation, users were asked to count various elements of the WSDL/SAWSDL files (after giving them sufficient explanations of what each element is and what it looks like). Although, developers never actually need to count elements in real use, this process is a measure of how rapidly and correctly a user perceives data through the representation. The same tasks were given again on the plain text form of the files (code) for comparison. We believe this process to be extremely hard and time-demanding for non-expert users on plain text editors, so we used a syntax-highlighting one. The five tasks assigned were to count various elements of a WSDL : A) its message nodes, B) its element nodes, C) its ComplexType nodes, D) all of its SAWSDL attributes and E) how many of these are Model References. The two views were selected in random order for each user, to generalize the result. Also, random WSDLs of about the same size were chosen for each user and each view, to eliminate learning.

The times recorded on code and tree views are shown on Figure 4 and Figure 5 respectively. A conclusion drawn is that all users perform similarly. A few outliers were the very fast #11 and the slow ones, #4 and #10. However, contrary to expectation, the outliers are not the ones with different background. We would expect the SAWSDL developer, who is #3, to be the fastest one, but his times are average. Additionally, the non-computer science student, #17, performs the same or even better than the rest. The same goes for the others: WSDL developers, #1 and #2, and PhD, #9.

The average times for each task and for each view, shown on Figure 6 confirm what is already apparent in the previous figures. First of all, some tasks, i.e. A, C and D in code, are more time-demanding than others. That is due to the fact that code essentially takes a long time, except for very easy tasks e.g. B and E. In the examples, elements (task B) were very compact on code and Model References (task E) were few (3-5) and already counted along with other SAWSDL attributes on the previous step (D). However, on tree view, all average

times are lower and even approximately the same for any task. Indeed code view is much slower on any task: the average of average task times is 15s on code view and 7s on tree view. Also, any task is expected to require around average time on tree view, while on code, some tasks can range widely: the standard deviation of average task times is 7s on code and 3s on tree view.

Meanwhile, answers are not only faster, but also more correct on tree view. Figure 7 shows the percentage of correct answers for each task

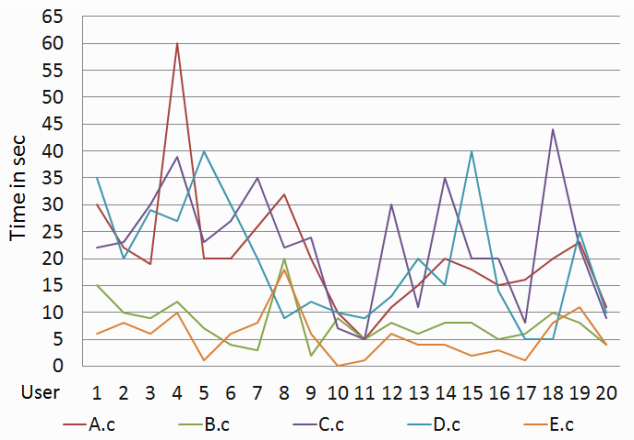

Figure 4. User answer times for each task on code view

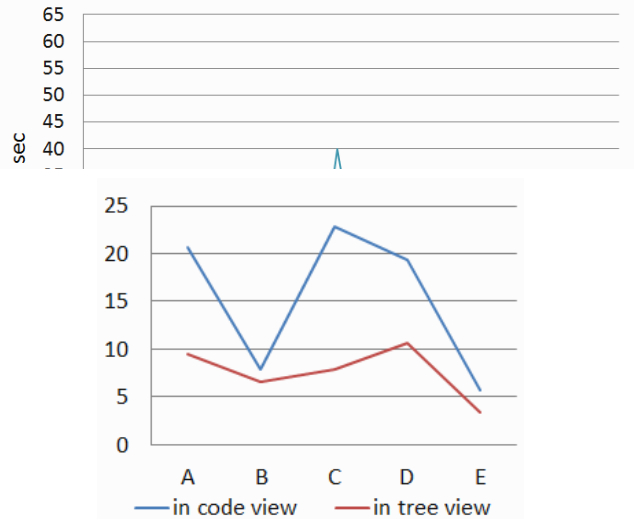

Figure 6. Averages of times recorded for each task

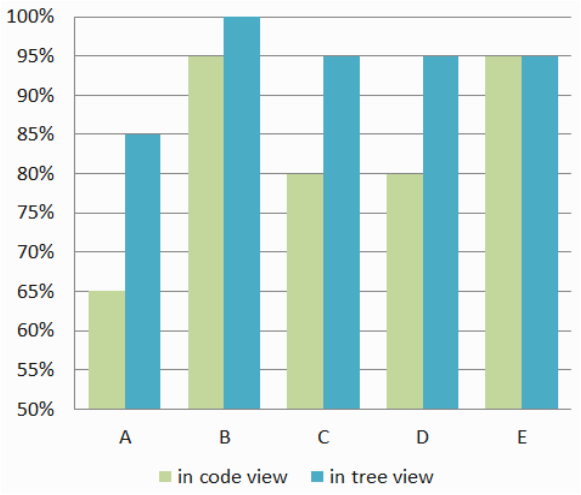

Figure 7. Percentages of correct answers for each task.

and view. The tree view percentage is always above or equal to the one in code view. We could also possibly deduct the fact that the more time the user takes, the more probable it is to answer incorrectly, as A,C and D in code are the most time consuming and poorly answered tasks at the same time. To conclude this part, users were asked for suggestions and ratings. They were asked whether they prefer code, tree or both simultaneously and all of them (100%) picked the latter. They rated the representation with 4.7/5 (30% 4/5, 70% 5/5) on a Likert scale (1 to 5). Two of them suggested a different representation where the tree structure is abandoned and nodes are categorized in groups.

During the second part, the users evaluated Iridescent's functionality by performing five tasks. First the users were asked to open an ontology and search for a random concept (task I). Then the search box was used for the same purpose (II). The users also entered two annotations on a WSDL, one by Drag 'n' Drop (III) and one via the Menu function (IV). The same two annotations were then committed via the recommendation function (V). Statistics for times recorded are shown on Table 5. Judging from the standard deviation, times are stable for all tasks except task I. For this task, some users used logic while searching (they followed the inheritance hierarchy) while some searched blindly and exhaustively. Approximately, for a single annotation one has to either manually or via search-box find a class (process of task I or II) and use Drag 'n' Drop (III) or the Menu (IV). Selecting the minimum in both stages, the complex task requires 9s on average. On the other hand, a recommendation requires 13s on average for two annotations, which is significantly faster (two manual annotations require 9s times two). Based on experience, selecting more annotations during the recommendation process requires insignificant amount of time, because there are all listed simultaneously. Hence, the difference between recommendation and manual annotations grows exponentially with the number of annotations.

Again, suggestions and ratings concluded the second part of the evaluation. When asked whether they would use a. Drag 'n' Drop only, b. Menu only or c. both, the users replied: 35%-a, 5%-b and 60%-c. They also rated the following functions on Likert scale (1 to 5): File management 4,75/5 (25%-4/5, 75%-5/5), Drag 'n' Drop and Menus 4,55/5 (5%-3/5, 35%-4/5, 60%-5/5), Recommendation 4,85/5 (15%-4/5, 85%-5/5), Namespace and Schema Mapping menus (they were shortly demonstrated) 4,65/5 (25%-4/5, 70%-5/5). When asked about Iridescent's usefulness, assuming they were SAWSDL authors, they answered positively 4,8/5 (20%-4/5, 80%-5/5). Finally, interesting suggestions include buttons to expand/collapse all tree nodes, keyboard shortcuts, optionally hide code, tree or split view and more functions for code.

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

6. FUTURE WORK

Future work on Iridescent focuses on two directions: improving its form and extending its functionality. Suggestions from evaluators such as the code/design/split view and more code functions, e.g. search, are going to be investigated. Functionality extensions include more string matching algorithms for recommendations. Another major functionality extension would be to provide actual authoring assistance for the Schema Mapping transformations e.g. XSLT, SPARQL. Feedback from the community underlines that the trans-

formation authoring process for every annotated element is often a very tiresome task. In that case the application has to be redesigned to not only insert the Schema Mappings (pointers to such files), but initially help author and upload the transformations to the cloud.

An additional important goal is to keep the tool compliant to future developments and adjustments of the specifications. We are eager to implement any suggestion in both directions, by the members of the community. For that reason there is a sample package with examples online as well as a form for suggestions. Other than that, we plan to also evolve and fully present the applications based on Iridescent annotations, on the client side, and possibly refine our methodologies and the tool itself.

7. CONCLUSIONS

This work presents the Iridescent tool for creating and editing Semantic Web Service descriptions following the SAWSDL standard. A review of current state-of-the-art has been carried out, identifying drawbacks and possible improvements. The proposed tool was designed and implemented to fulfill these critical requirements such as maximum availability, usability and functionality for both expert and non-expert users. Additionally, a novel way to produce batch automatic recommendations is proposed in this tool, using primitive data (string matching) to produce semantic information. The tool's aim is to promote the use of the promising Semantic Web technologies to enable intelligent client applications. Use case scenarios for enabling an Ambient Intelligence system, through matching and intelligent agents, have been demonstrated to prove the tool's and SAWSDL's capabilities. Evaluations carried out by a variety of users show optimistic results. Users seem eager to adopt a graphic tool and are especially fond of intuitive functions and automatic recommendations. Timings during the testing also show that the graphic tool's representation gives a significantly better overview of the code as the users answered more promptly and correctly than in code. Ratings seem optimistic while suggestions from evaluators and the community are bound to be implemented. A major suggestion was to extend the tool to actually penetrate the transformation authoring process (i.e. beyond the SAWSDL protocol and into XSLT and SPARQL), providing even more automation.

8. ACKNOWLEDGEMENTS

The authors would like to thank the undergraduate student Theodoros Mylonides for his contribution. The Smart IHU project is funded by Operational Program Education and Lifelong Learning, OPS200056 (International Hellenic University, Thessaloniki, Greece).

9. REFERENCES

[1] Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S. 2001. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl

[2] Tran, V. X., Puntheeranurak, S. and Tsuji, H. 2009, June. A new service matching definition and algorithm with SAWSDL. In *Digital Ecosystems and Technologies, 2009. DEST'09. 3rd IEEE International Conference on* 371-376. IEEE.

[3] Iqbal, K., Sbodio, M. L., Peristeras, V. and Giuliani, G. 2008, December. Semantic service discovery using SAWSDL and SPARQL. In *Semantics, Knowledge and Grid, 2008. SKG'08. Fourth International Conference on* 205-212. IEEE.

[4] Lécué, F., Gorronogoitia, Y., Gonzalez, R., Radzimski, M. and Villa, M. 2010, July. SOA4All: An innovative integrated approach to services composition. In *Web Services (ICWS), 2010 IEEE International Conference on* 58-67. IEEE.

[5] Gomadam, K., Verma, K., Brewer, D., Sheth, A. P. and Miller, J. A. 2005. Radiant: A tool for semantic annotation of Web Services. In *Proceedings of the 4th International Semantic Web Conference* (ISWC 2005) Galway, Ireland - Demo Paper.

[6] Dimitrov, M., Simov, A., Momtchev, V. and Konstantinov, M., 2007. WSMO Studio – a Semantic Web Services Modelling Environment for WSMO, *The Semantic Web: Research and Applications*, LNCS, vol. 4519, 749-758. DOI=http://dx.doi.org/10.1007/978-3-540-72667-8_53

[7] Valle, E. D., Niro, G. and Mancas C. 2011. Results of a Survey on Improving the Art of Semantic Web Application Development, *The 7th International Workshop on Semantic Web Enabled Software Engineering Co-located with ISWC2011*

[8] Jaeger, M., Engel, L. and Geihs, K. 2005. A methodology for developing owl-s descriptions, in *Proceedings of the 1st Int. Conf. on Interoperability of Enterprise Software and Applications. Workshop on Web Services and Interoperability*. February 2005

[9] Gannod, G. C., Timm, J. T. and Brodie, R. J. 2006. Facilitating the specification of semantic web services using model-driven development. *International Journal of Web Services Research (IJWSR)*, 3(3), 61-81.

[10] Stavropoulos, T. G., Tsioliaridou, A., Koutitas, G., Vrakas, D. and Vlahavas, I. 2010. System architecture for a smart university building. *Artificial Neural Networks – ICANN 2010*, 477-482.

[11] Qiu, L., Chang, L., Lin, F. and Shi, Z. 2007. Context optimization of AI planning for semantic Web services composition. *Service Oriented Computing and Applications*, 1(2), 117-128.

[12] Thomson, G., Bianco, S., Mokhtar, S. B., Georgantas, N., and Issarny, V. 2008. Amigo Aware Services. *Constructing Ambient Intelligence*, 385-390.

[13] Georgantas, N., Issarny, V., Mokhtar, S. B., Bromberg, Y. D., Bianco, S., Thomson, G., ... and Cardoso, R. S. 2010. Middleware architecture for ambient intelligence in the networked home. In *Nakashima, H., Augusto, J. C. & Aghajan, H. (Eds.), Handbook of Ambient Intelligence and Smart Environments. Springer*.1139-1169.

[14] Stavropoulos, T. G., Vrakas, D., Vlachava, D. and Bassiliades, N. 2012, June. BOnSAI: A Smart Building Ontology for Ambient Intelligence. In *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics* 30. ACM.