2009 Third International Conference on Advances in Semantic Processing

# WISE - Workbench for Semantic Web Services

Keith Lia, Charlie Abela
*Department of Artificial Intelligence*
*University of Malta, Malta*
{klia0001, charlie.abela}@um.edu.mt

James Scicluna
*STI Innsbruck*
*Austria*
james.scicluna@sti2.at

***Abstract*—The onset of the Semantic Web has brought many innovations for enabling assisted interactions on the World Wide Web through document annotation. Many efforts have emerged throughout the past years, including RDF, OWL, OWL-S and WSMO, all seeking to improve the current web from a static one into a more dynamic one. Semantic Web Services are an effective way to facilitate the discovery, selection and composition of services and to integrate them into workflows. Presently, however, not many tools offer the functionality to facilitate the annotations of such services to ontology frameworks such as OWL-S or WSMO. Presented here is an Editing Suite for such a scenario. The aim of this work is to provide a tool that provides two core features. The first is a generic framework that allows the extraction of data from an SAWSDL document, as well as having an integrated visual editor for building composite business processes. The second is a set of *hooking* mechanisms, able to extend the above framework so as to provide mappings from SAWSDL to any other ontology framework.**

***Keywords*-semantic web; semantic web services; workbench;**

## I. Introduction and Background

The World Wide Web as we know it today was created primarily to be used and understood by humans. This resulted in a Web in which most of the documents were static pages. As the Web grew, the number of website grew exponentially, making it difficult to find relevant information manually. However, since the Web was originally designed for humans and not machines, it proved difficult to come up with an efficient way of searching the documents automatically.

The structure of the paper will be as follows. We first give a brief introduction on the Semantic Web and Semantic Web Services. We then move on to describe the motivation for the Workbench, as well as its design and implementation. The results are then presented, which compare the Workbench to other existing frameworks and editors, followed by a brief conclusion.

### A. The Semantic Web

One of the motivations behind the Semantic Web is to build on the existing Web and to give "meaning" to documents - thus enabling machines and agents to understand what a particular document is about. Furthermore, agents would be able to "understand" each other and communicate together to transfer information.

Throughout recent years, a number of Ontology Languages were created in order to better represent machine-readable content on the Web. Such languages are most often formal languages and are used mainly for knowledge representation. Throughout this paper, two Ontology Languages will be touched on, namely OWL and WSML.

The Web Ontology Language (OWL Guide, 2004) is a semantic markup language for sharing ontologies on the Web. It builds on RDF by adding new vocabulary for describing classes and their properties, thus enabling modelers to create taxonomies of real-life resources.

The Web Service Modeling Language (6) is part of the WSMO working group, and aims to formalize the Web Service Modeling Ontology. It offers a number of different variants on the language, which allow the modelers to either move to the *Logic Programming* aspect, or to the *Description Logic* aspect.

### B. Semantic Web Services

Traditional standards for describing Web Services, like the Web Service Description Language, put their focus mainly on the syntax of the message transfer while ignoring their semantic meaning completely. However, with the introduction of Semantic Web Services, this is no longer the case. The intention behind such services is to have an effective way to facilitate the automatic discovery, automatic invocation and automatic orchestration of services and to integrate them into work flows. Thus, this would eliminate the need of a human agent as was the case for traditional Web Services. A number of Ontology Frameworks have also arisen in recent years for the purpose of providing the semantics to Web Services, examples of which are OWL-S and WSMO.

The Web Ontology Language for Services, or OWL-S (7), is based on the OWL language, and is divided into four distinct ontologies - The Upper Ontology, the Service Profile, the Process Model and the Grounding. Each one of these ontologies describes a different aspect of the same Web Service and can be split into the *abstract* definition and *concrete* definition. The Profile and Process fall into the former, while the Grounding falls into the latter. The Upper Ontology serves as an entry-point to the Web Service, and points the agent to the other ontologies. The Service Profile tells the agent what the service does, and may

978-0-7695-3833-4/09 $26.00 © 2009 IEEE
DOI 10.1109/SEMAPRO.2009.18

also give additional information about the service provider or the service itself. The Process Model tells the agent how to interact with the service and can be viewed as a process - having a number of inputs and outputs, pre-conditions and results. The Grounding aspect usually hooks to a WSDL document and mapped to a concrete message transfer protocol, such as SOAP.

WSMO, or Web Service Modeling Ontology (8), is another Ontology Framework for Semantic Web Services. It is made up of four top-level elements - Ontologies, Web Services, Goals and Mediators, which are responsible for the different aspects of the framework. Ontologies provide a terminology by which the other elements can describe the semantics, such as defining new "Concepts" (classes) and their properties. Goals specify the objectives that users may have when consulting a Web Service. They will usually have pre- and post-conditions to describe what the Web Service expects and returns respectively. Mediators are based on the concept of "Adapters". Their main aim is to resolve conversational mismatches between choreographies and orchestrations. Finally, the Web Service element describes the various aspects of a web service, from the functional, to non-functional and behavioral.

SAWSDL, which stands for Semantic Annotations for WSDL, builds on traditional WSDL in order to add semantics to the WSDL elements. This is done by linking the element to a concept in some ontology, be it OWL-S, WSMO or other. SAWSDL does not impose any restriction on the ontology formalism used, and also allows multiple annotations on the same element. Furthermore, SAWSDL can be easily integrated into legacy software, as it adds to the current WSDL but does not change the structure.

### C. Motivation

In order for the popularity of these technologies to grow, there was the need for a number of tools to enable the creation of such services in an easy manner. A number of editors have surfaced over recent years, including Protege for creating OWL Ontologies, OWL-S Editors and WSMO-Studio & WSMT for creating WSMO descriptions. These are discussed in further detail in Section V. The inspiration for this project is the work done by James Scicluna (Scicluna et al, 2004) - a visual OWL-S editor which enables the development of semantic web service descriptions without exposing the developer to the underlying OWL-S syntax. However, OWL-S has itself decreased in popularity in recent years, therefore the focus for this editor is abstracted away from mapping to a single Ontology Framework. Instead, it is geared towards the ability to define multiple mappings from between Web Service Description Languages to a number of different Ontology Frameworks.

The Web Service Workbench aims to provide a visual editor which enables the creation of semantic web service descriptions, without exposing the developer to the underlying syntax. In contrast to existing editors, our objective was to create a "generic" editor that is capable of being extended, hence allowing the generation of different Semantic Web Service frameworks.

## II. Design

The design of the Workbench was split into three components - an *SAWSDL semantics extractor*, a *graphical process editor* and a *transformation layer*. The SAWSDL extractor retrieves the WSDL elements and their annotations from the SAWSDL document to prepare a data structure to create the ontology skeletons. The graphical editor allows the user to visually model the service processes through the use of BPMN, which is a standard notation for modeling business processes. Finally, the Transformation Layer uses the data structures generated by the previous two components to create the full ontology descriptions. Figure 1 shows a high-level design for the Workbench.

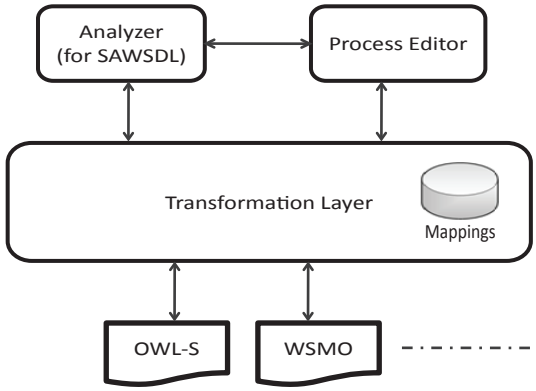


Figure 1. Web Service Workbench Design

The main entry point in the Workbench is the *SAWSDL Extractor*. This is presented to the user as a Wizard and serves as a guide to create the skeleton ontology. The *SAWSDL Extractor* ensures that the document provided is a valid WSDL2.0 document, and if valid, the data is extracted. This includes the various WSDL data types, the Interfaces, their Faults and Operations, and the Operation Elements. If any of these WSDL elements have an associated annotation, they are also extracted.

For the *Process Editor*, a ready-made BPMN modeler was used, called SOA Tools BPMN Modeler (9). This allows the users to create composite processes from other simpler processes, which were extracted from the SAWSDL document during the previous stage. Composite processes can be made up of several constructs, such as sequences, loops or conditionals.

The *Transformation Layer* is the 'generic' part of the editor. The idea behind this layer is the mapping from SAWSDL to a specific ontology language, without tying the developers down to only one framework. During the design

stage, is was noted that although Ontology Frameworks use different syntax, there are some aspects that are common amongst them. The four aspects defined are:

1) The Functional Aspect - This describes the functions offered to the requesting agent, and can also aid in the discovery of the service. In OWL-S, this corresponds to the Service Profile, whereas in WSMO it would roughly correspond to a Capability.
2) The Behavioral Aspect - This describes the way the requesting agent must interact with the service to make use of its functionalities. In OWL-S, this is done through the Service Model, whereas in WSMO this is done through the Interfaces, particularly Choreographies and Orchestration.
3) The Grounding Aspect - The grounding usually describes the concrete aspect of a web service. It tells the requesting service what protocols and formats are needed in the message transfer so as to ensure good communication between the agent and the service. In most cases, the service is grounded into a WSDL document. In OWL-S, the Service Grounding takes care of this by providing mechanisms of hooking the Atomic Processes to WSDL Operations. In WSMO, a grounding can be established by mapping concepts and relations in the Choreography descriptions to WSDL constructs.
4) The Upper Ontology - This is the simplest aspect of the service as all it does it point to all the necessary constructs that are needed in its execution. The purpose of this is to connect all the other aspects of the service together once it has been discovered by an agent. In OWL-S, this is done by the Service Ontology, while in WSMO, the Web Service takes care of this.

These aspects are important to abstract away the implementation details and only keep the ones common throughout all ontologies. The Transformation Layer is invoked twice during the life-cycle: the first time at the end of the ontology creation wizard at which stage the partial ontology is formed, and another time after the composite process is created. Developers can easily integrate new mappings to the Workbench through this layer.

The separation of the three components is very important. Both the SAWSDL Analyzer and Process Editor will be needed regardless of the mapping. However, since our primary aim is to make the editor extensible, the Transformation Layer has to be made in a way to be easily swapped when necessary. New mappings can be defined and hooked onto the Transformation Layer and, in this sense, the editor becomes fully generic to any ontology framework.

## III. Implementation

The Workbench was developed as a plug-in to the widely used Eclipse IDE. The idea to move away from a stand-alone application stems from the fact that Eclipse has become the de-facto platform for such applications. Since the main aim for the editor is for it to be extensible, building the Workbench on top of Eclipse allow it to be extensible automatically while leaving the actual mechanics to the Eclipse Framework.

The Implementation is split into four major packages. The Wizard package is concerned with extending the Eclipse wizards in order to customize them for the Workbench. It also provides basic functionality to create an empty Eclipse project in which the ontology descriptions are saved, as well as a simple wizard page to enable the user to select a valid SAWSDL document. Furthermore, the Wizard can also be extended in the Transformation Layer to include more wizard pages than is available by default. Should the Ontology Framework being mapped contain other aspects outside of the Functional, Behavioural or Grounding, the Wizard could be extended in order to accommodate them.

The SAWSDL Extractor package is directly related to the Extractor component. The Wizard transfers the selected SAWSDL document to this package to check for its validity and to extract the important data. An *Extractor* class first checks whether the document is a valid WSDL 2.0 document, while all other documents will cause the wizard to throw an exception. Once validated, a loop will traverse all the WSDL Interfaces and extract their *Faults* and *Operations*. The class also contains a recursive function that retrieves the Complex and Simple Types in the WSDL document. Since SimpleTypes cannot be decomposed further, they are used as the base case. If a ComplexType is encountered, the elements that make it up are passed through this method again until a SimpleType is encountered instead. A tree of Elements and their respective Types is then created. Ultimately, another tree data-structure is created, containing an array of Interfaces as the root, with each Interface branching into its constituent Faults and Operations, which in turn branch into their respective Types.

The Ontology package contains classes that are used to transfer data to and from the Transformation Layer and the Wizard package. This package contains a Factory class and an Interface, which together make use of the Factory method pattern to enable the Transformation Layer to easily create and integrate new mappings with the rest of the Workbench. The implementation of the Interface is left to the developer who is defining the mappings. The Workbench is only concerned about receiving the final output, that is, the generated ontology descriptions. However, if the Transformation Layer does not implement this interface, the Workbench will not run successfully.

The BPMN package corresponds to the Process Editor component. Because the Workbench was developed as an Eclipse plug-in, this allowed us to make use of other Eclipse plug-ins, such as the SOA Tools BPMN Modeler. This package implements the Modeler, as well as extends and customizes it to better fit the Workbench. One of the most

important customization is the inclusion of the "Process Annotation" functionality, which annotates a BPMN Task with an atomic process from the SAWSDL document. This makes sure that the composite process is indeed made up of the simple processes and facilitates the mapping from model to ontology description.

Once the composite process is modeled, the user can generate the description by clicking on the "Generate" button in the Eclipse Framework. This parses the diagram and creates a tree of Tasks, which is sent to the Transformation Layer, where the mappings defined by the developer transforms it into the full ontology description. If the model contains a conditional branch, an additional pop-up dialog will prompt the user to enter the condition to be satisfied, based on the outputs of the previous task. Currently, the outputs can only be compared to literal values, however, work is being done to allow the comparison between outputs of different Tasks.

The Workbench also makes use of Eclipse Extension Points in order to extend its functionality. The first extension point defined allows the developers to extend the Wizard by defining new pages to be appended. Four different interfaces correspond to the four aspects of the Ontology Frameworks, namely the Functional, Behavioural, Grounding and Upper Description. Each interface that is implemented allows the developer to transfer properties about the respective aspect from the wizard to the Transformation Layer. These can then be used by the developer, as well as added on or modified, and passed back to the Wizard class. The Workbench contains a number of in-built properties which allows the developer to manipulate the ontology description filenames, as well as to access important data such as the *Base URI* and the source WSDL file.

The other extension point defined is the Ontology extension. Unlike the previous extension point, this mandates that the developer implements the Ontology Interface from the Ontology package. The interface contains methods that must be implemented, such as the creation of the ontology descriptions. During runtime, an instance of the implementation is retrieved, and is used as a "gateway" between the Workbench and the Transformation Layer. If no implementation is found, the Workbench will not be able to run successfully and will throw and exception instead.

Developers who are looking into extending the Web Service Workbench must first create a new Eclipse plug-in, and include the Workbench as a dependency. This will cause all the exposed APIs and third party plug-ins to also be exposed to the developer, as well as the defined Extension Points.

## IV. Results

A number of tests were performed on the final prototype, which are outlined in this section.

### A. OWL-S Use Case

In order to test the Workbench, a use case for OWL-S was implemented. Table I shows the mappings used to transform from SAWSDL to OWL-S based on work by Martin (2007). The entire Profile, Grounding and Upper description can be generated from the SAWSDL document. The Process Model can be generated in two parts - a basic description outlining the atomic processes can be generated from the SAWSDL document, whereas the complete description is generated from the BPMN Modeler.

| WSDL 2.0 Component | OWL-S Description |
|---|---|
| Interface | Service Profile |
| Operation (without Faults) | Atomic Process |
| Operation (with Faults) | Composite Process (with Conditional Effects) |
| Element (Simple, Complex Type) | OWL Class |
| Interface Fault Reference / Interface Message Reference | Input / Output |

Table I
MAPPING FROM WSDL 2.0 TO OWL-S

An `AtomicProcess` is created for every `Operation` in the `Interface`, and the name of the `Operation` itself is used as the name of the process. Each `Element` used from within the `Operation` is then mapped to an OWL-S Parameter (i.e. input or output) to reflect its role in the SAWSDL document. The OWL-S inputs and outputs require a `parameterType` to identify their type. Ideally, this would refer to an OWL class, however the mapping of `Elements` to OWL Classes was not implemented due to the time constraints. Instead, the `parameterType` of an OWL-S Parameter contains a reference to the XML data type used in the SAWSDL document directly. If an `Element` has a `ComplexType`, a recursive function in this class will "break" it down into its simple constituents. However, to make use of the Semantic Annotations offered by SAWSDL, if any of the `Operation` Inputs or Outputs, or any of the `Elements` contain a `modelReference` attribute, this are used as the `parameterType` instead. If multiple annotations are present for the same construct, then only the first one is used.

### B. WSMO Use-Case

The WSMO Use-Case is still a works in progress, but contains a number of elements similar to the OWL-S implementation. Focus is being given exclusively to the "Web-Services" aspect of the Ontology, and allows users to automatically generate this WSMO element directly from

the Workbench. Table II shows the mappings currently being used.

| WSDL 2.0 Component | WSMO Web Service |
|---|---|
| Interface | Capability |
| Operation | WSMO Interfaces particularly Choreographies |

Table II
MAPPING FROM WSDL 2.0 TO WSMO WEB SERVICES

### C. Limitations

In its present version, the Workbench does have some limitations, particularly in the mapping generation of the Process Model. While various ontology frameworks contain a number of constructs for their composite processes, the present mapping mechanism only allows for sequence, parallel processes and conditionals.

Another limitation involves the semantic annotations in the SAWSDL document. Although SAWSDL itself does not pose any constraints on the ontology language used for the elements, in order for the OWL-S use case to make use of the annotations, the language used must be constrained to OWL. There is no direct work-around to this as OWL-S is based on the OWL Ontology Language. Furthermore, other use cases would require different ontology languages altogether.

Although the transformation for the OWL-S Editor was not entirely complete it suffices to provide a proof of concept and is in fact capable of generating valid OWL-S descriptions albeit limited to only certain scenarios.

### D. Testing

With the implementation of the OWL-S Use Case, it was shown that:

- new functionality can be extended to the Workbench,
- the Workbench works with new mapping definitions, and
- the Workbench accomplishes its main task, that is, to be a generic editor.

A number of SAWSDL documents were tested from [1], in order to check the Workbench stability under differing types of annotations. The two extremes were tested, namely a SAWSDL document with no annotations (that is, a normal WSDL 2.0 document), and another document having all elements annotated with multiple concepts. Other tests included using annotations only on the Simple and Complex Element Types, and only having annotations on the Interfaces. These tests were repeated several times with other SAWSDL documents to ensure the correctness of the Workbench, and all results were positive. As expected, all the relevant WSDL constructs and annotations were successfully extracted regardless of the WSDL 2.0 document used. On the other hand, because WSDL 1.1 documents, or other non-WSDL documents, are not supported, these are caught by the wizard before any extraction can take place.

A SAWSDL document, found at {*http://www.w3.org/2002/ws/sawsdl/CR/wsdl2.0/00-all.wsdl*}, was used as a major test case for the Workbench. An additional three Operations - `rentRooms`, `sendInvites` and `sendComplaint` - were added in order to test the *Composite Process* generation. Figure 2 shows a BPMN diagram that uses a Conditional Branch. Upon clicking the "Generate" button, a pop-up similar to Figure 3 is shown to the user. In this case, the output of `opCheckAvailability` was of type *checkAvailabilityResponse*.

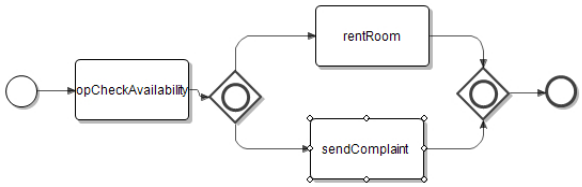


Figure 2. BPMN Model created in the Workbench from the SAWSDL document, showing Conditional Branching

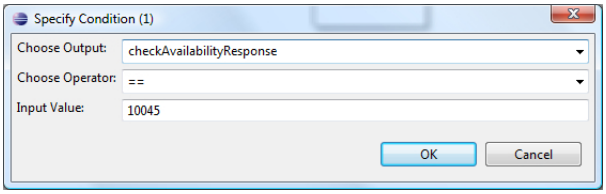


Figure 3. Pop-Up dialog for conditional values

## V. COMPARISON WITH EXISTING PLATFORMS

During the evaluation process, the final prototype of the Web Service Workbench was also compared to the other Editors. The first comparison was made with WSMO Studio (10). This is an Eclipse plug-in for creating complete WSMO Ontology descriptions, and can be extended further to include new functionality to the Eclipse plug-in. Although WSMO Studio would be a better choice when creating full WSMO Services, the Workbench together with a WSMO mapping would be useful for users who would want to create a WSMO Web Service element automatically in a quick manner. For a more comprehensive guide on WSMO Studio, the reader is referred to (Dimitrov et al, 2007).

Similar to WSMO-Studio, WSMT (11), or the Web Service Modeling Toolkit, is a framework for creating and managing WSMO, WSML and WSMX resources. It is an

[1] http://www.w3.org/2002/ws/sawsdl/CR/testsuite.html

application built on the Eclipse framework and includes the functionality for additional plug-ins to be linked with it. The toolkit is made up for three tiers: The *launcher* builds the dynamic class loader for the application to run, and allows plug-ins to be deployed. The *core* contains the GUI of the application, as well as a set of components that provide reusable functionality to all the plug-ins. Finally, the last tier is made up of the individual plug-ins that hook in to the main application. WSMT contains several tools, including a WSML Editor and a WSMX Invoker. Like the Eclipse Framework, WSMT is capable of being extended further, allowing developers to add extra functionality. Despite this however, the main goal of WSMT is to provide in-built functionalities to the WSMO family only, and lacks "support" to other ontology frameworks. In this aspect therefore, our Web Service Workbench has a leading edge as the prime objective was to make it generic to different ontologies.

The Protege OWL-S Editor (12) and Scicluna's OWL-S Editor (13) are both editors specifically designed to create OWL-S descriptions. The strengths that the Web Service Workbench has over these is that the Workbench was designed and built to be generic. Furthermore, because of the extensible nature of the Workbench, additional mappings can be defined to further increase the different ontologies supported. The reader is kindly requested to refer to (Elenius et al, 2004) for more information on the Protege Editor's features, and to (Scicluna et al, 2004) for more information about the OWL-S Editor.

## VI. Conclusions

The work presented in this report reflects the recent work that has been underway in the Semantic Web scene, particularly Semantic Web Services, with efforts such as SAWSDL, OWL-S and WSMO being the major influences. The end result is the Web Service Workbench, an Eclipse plug-in and a generic framework which provides an easy-to-use framework for creating Semantic Web Services.

A similar use case for the WSMO Ontology Framework is currently also underway. The focus is being put on the WSMO Web Service Element aspect, and not on the whole framework. Like the OWL-S Use Case, the WSMO extension will use the wizards to create the functional, grounding and upper services, while using the SAWSDL document and BPMN Modeler for the behavioural description.

Other minor future work involves work done on customizing the BPMN Modeler. Currently, the Workbench only allows the creation of one composite process per service. Because of this limitation, the composite processes can only be composed of atomic processes. Finding a work-around to the one composite process problem would mean that composite processes would be able to incorporate other composite processes in their business model.

## References


1. Scicluna J., Abela C., Montebello M. 2004. *Visual Modelling of OWL-S Services*. IADIS International Conference WWW/Internet.

2. Martin D., Paolucci M., Wagner M. 2007. *Toward Semantic Annotations of Web Services: OWL-S from the SAWSDL Perspective*. OWL-S: Experiences and Directions, 4th European Semantic Web Conference (ESWC) 2007.

3. Dimitrov M., Simov A., Konstantinov M., Momtchev V., Cekov L. 2007. *WSMO Studio Users Guide*. *http://www.wsmostudio.org/doc/wsmo-studio -ug.pdf* Last Access Date June 2009

4. Elenius D., Denker G., Martin D., Gilham F., Khouri J., Sadaati S., Senanayake R. 2005 *The OWL-S Editor Development Tool for Semantic Web Services*. Second European Semantic Web Conference. ESWC 2005, Heraklion, Crete, Greece. Vol. 3532. pp. 78-92.

5. Smith M. K., Welty C., McGuinness D. L. 2004 OWL - Web Ontology Language Guide. http://www.w3.org/TR/owl-guide/ Last Access Date July 2009

6. The Web Service Modeling Language - WSML. http://www.wsmo.org/wsml/wsml-syntax Last Access Date June 2009

7. OWL-S: Semantic Markup for Web Services. http://www.w3.org/Submission/OWL-S/ Last Access Date July 2009

8. ESSI WSMO working group. http://www.wsmo.org/ Last Access Date July 2009

9. SOA Tools BPMN Modeler. http://www.eclipse.org/bpmn/ Last Access Date July 2009

10. WSMO Studio. http://www.wsmostudio.org/ Last Access Date July 2009

11. Web Service Modeling Toolkit. http://sourceforge.net/projects/wsmt Last Access Date July 2009

12. Protege OWL-S Editor. http://owlseditor.semwebcentral.org/ Last Access Date July 2009

13. OWL-S Editor. http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/OwlSEdit.html Last Access Date July 2009