Selecting the Best Web Service

Julian Day
University of Saskatchewan
Saskatoon, SK, Canada
julian.day@usask.ca

ABSTRACT

Web services are applications that communicate over open protocols such as HTTP using structured forms of XML such as the Simple Object Access Protocol (SOAP [17]) or Remote Procedure Calls for XML (XML-RPC [18]). There have been efforts to standardize a number of things: the Web Service Description Language (WSDL [5]) is a standard for describing web services' syntax, and the Universal Description, Discovery and Integration protocol (UDDI [1]) is often used as a discovery mechanism for dynamically finding new services. However, there have been few efforts to describe the interactions between clients and services. In this paper, a system is discussed and analyzed for using the Resource Description Framework (RDF [13]), the Java Expert Systems Shell (JESS), and the Web Ontology Language (OWL [15]) to augment web service clients. The clients can collect, report, and analyze data about their experiences with the quality of service (QoS) of web services, and are able to parse and use this information to dynamically select the best service for their needs.

General Terms

web services

Keywords

web services, semantic markup, quality of service

1. INTRODUCTION

XML Web services were introduced with the specification of XML-RPC by UserLand software in 1998 [18]. Web services use XML-RPC or SOAP [17] for encoding data over a transport layer, usually HTTP, so there are many that consider XML-RPC to be the start of web services. Popularized by their inclusion in software packages such as Microsoft Visual Studio and Sun's Web Services Developer Pack, web services have recently seen an explosion in popularity. Their design intention is to increase interoperability between applications running over the web through the use of open standards. SOAP is a W3C recommendation, and XML-RPC is essentially halted: development of the specification stopped in 1999, except for a small update in 2003 to allow for Unicode strings.

Web services communicate by sending requests over HTTP. These are standard HTTP requests with SOAP- or XML-RPC-encoded content appended to the end. The service processes the request, and then sends an HTTP message back. This process is shown in Figure 1.

Currently, there has been much focus on how to characterize the syntax of web services. The most popular of these is the Web Service Description Language, or WSDL. WSDL is "an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information." [5] However, while WSDL remains popular, there has been less work in standardizing *meaning* for web services: what they do, what they require, and what they provide, among other things. One area that has recently been explored for these purposes has been that of the semantic web.

The semantic web, as envisioned by Tim Berners-Lee, et al., is the markup of web data so that its semantics are machine-understandable [2]. It was conceived after the realization that despite the vast amounts of data stored online, computers could understand the semantics of virtually none of this information. One of the efforts produced by researchers in the semantic web community has been the creation of semantic markup languages [8]. The purpose of these languages is to give semantic structure to data, allowing systems to "understand" information in ways similar to how humans would. A variety of languages have been produced. Among the best-known are RDF [13], RDF Schema [4], DAML+OIL [9], OWL [15], and OWL-S [6]. These are each discussed in detail in Section 3.

The system described in this paper provides an automated web service client augmented with semantic models based on RDF and OWL, and a reasoning engine built in JESS, a rule-based expert system implementation in Java based on the RETE [7] algorithm. Both the semantic model and the reasoner will be discussed in detail in Section 4. These, combined with the publicly accessible forums to which the clients report their experiences, allow clients to reason about which of a number of syntactically identical web services will provide them with the best service. Once the client has

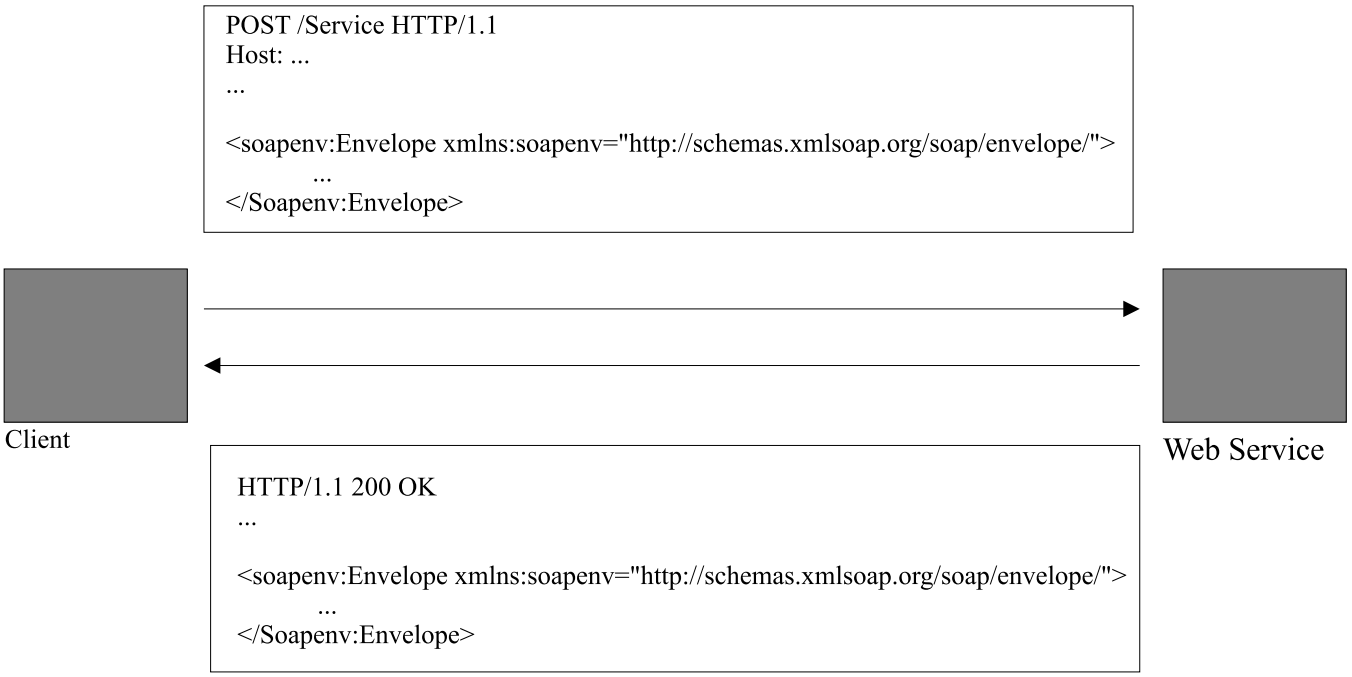

Figure 1: A Client Interacts With a Web Service

determined this, it will transparently switch its operations over to that service.

This paper is structured as follows: Section 2 gives a description of the problem of web service selection, and the approach taken in this paper. Section 3 is a survey of the literature on web services; the semantic web, semantic markup languages (including RDF, DAML+OIL, OWL, and OWL-S), and semantic web services; and a short description of rule-based expert systems. Section 4 describes the system built for dynamic web service selection, with experimental data from that system discussed in Section 5. Conclusions are presented in Section 6, with future directions for research in Section 7.

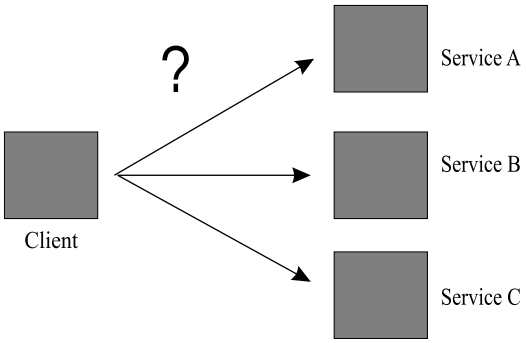

Figure 2: An illustration of the selection problem

2. PROBLEM DEFINITION

Selecting a web service for a client is typically a task performed by the designer of the client. The designer would know about one or more services, and generate the client based on the location and suitability of those services. In this paper, a solution is proposed whereby the designer is freed at least somewhat from the selection process, allowing both the user and the system to work together to find and use the best service available. Given a number of user-specified, syntactically identical web services that purport to provide the same services to the client, which should the client select? An example of this can be seen in Figure 2.

One approach to this problem could be through negotiation. If each web service could be seen as having a cost associated with it, is the cheapest service sufficient? Not necessarily. And neither is the most expensive service necessarily the best. Because cost is not a certain indicator of QoS, negotation is ignored and instead, the problem of selection is tackled by looking at the past experience of those who have used the service. The process used is to build an evaluation function, f, which guides the client as to which service to select. f takes in the raw data about a number of web services, processes it, and then returns the best web service for that particular client. The function f is currently fairly simple, applying a set of weights to the mean experienced reliability, availability, and execution time. These weights have default values, but can be easily changed by the user to reflect his or her preferences. The system could, through the use of more JESS rules, be extended to add weights to factors such as the IP addresses of the reporting clients, allowing for trust-based weightings or higher weights for clients within a particular subnet. The use of an expert system shell, JESS, for reasoning makes the system more open and scalable to more complex rules.

There were two approaches that were considered for getting the information required for sharing experiences: server-side or client-side augmentation. A server-side augmenta-

tion would allow the web service and its designers to talk about the guarantees that could be received from it. By contrast, a client-side approach treats the web service as a kind of black box. The client knows about the operations available on the web service (as a programmer has coded a client for their definition), but otherwise, the web service's semantics are ignored. Instead, the system captures the automated calls made by the client, and notes a number of things: whether or not the calls got through, whether or not the expected operation return type was returned, and how long, in seconds, the call took to execute.

The approach taken is to augment the client, rather than the server. There have been some efforts to use server-side information to make selection decisions [14], but these operate on the principle of objective publishing. However, if a service is not entirely forthright with the accuracy of its semantic information or service guarantees, then a client could be fooled into selecting an inferior service. It is for that reason that the client is augmented: assuming no tampering and database flooding, correct data should be compiled and made available to the clients. This might be a strong assumption, however, and it is discussed further in Section 7. The clients send their experiences to a central web service which stores this information inside an internal database. This web service can be thought of as a kind of forum system for QoS information. It can respond to requests about particular web services, sending all the data it knows about a particular service to a requesting client. Now when a client wants to pick a service, it gathers information from the QoS forums, and then reasons about which service is best. This is illustrated in Figure 2.

The four questions to be answered are as follows:

1. How can a web service be evaluated?
2. Which semantic markup language (RDF, DAML+OIL, OWL, OWL-S, etc) should be used for the representation?
3. What is needed to describe a web service quality of service ontology?
4. Can such a system be built?

3. LITERATURE REVIEW

3.1 Web Services

Web services were first discussed by the W3C in 2000. While XML-RPC had been finalized, and a W3C mailing list created for XML protocol discussion, in the previous year, it was not until 2000 that the W3C started to create plans for XML protocols. In February of 2000 they created an interim plan for XML protocols, and in September of that year, they began investigating using XML for a protocol to facilitate application-to-application messaging. A year and a half later, in January of 2002, they extended this by launching the Web Services Activity. Its aim was to extend the scope of the ongoing XML investigation into all aspects of web services. The goal of this research, as they put it, has been "to design a set of technologies fitting in the Web architecture in order to lead Web services to their full potential" [10].

These activities have produced a number of results. One of the most important was a common protocol on which most web services are based: SOAP. The first version of SOAP, 1.1, was detailed in a W3C note from May 8, 2000 [3]. The specification was authored by a number of people from a number of large companies such as IBM and Microsoft. The most recent version of SOAP, 1.2, is a W3C recommendation as of June 24, 2003.

Also part of the W3C are the Web Services Description Working Group, and the Web Services Choreography Working Group. The former group is largely responsible for WSDL, while the latter seeks to create a language to describe the relationship between web services (hence the "choreography"). The Description Working Group, as of the time of writing, has not yet produced the long-expected WSDL 2.0. It is expected to be ready some time around May of 2004. The Choreography Working Group is expected to public a working first draft by the end of 2004 [10].

While much research has been put into the standardization process, there has recently been more academic research in the area of web services. One area that has been receiving more attention lately is that of web service selection: how can we determine the best web service for a particular need? Often, as in this paper, the proposed solutions are dynamic, allowing for on-the-fly service selection and invokation.

Liu, Ngu, and Zeng [12] approach this problem, detailing a dynamic selection process for web services based on QoS computation and policing. Their system uses an "open, fair, and dynamic QoS computation model for web services selection" by means of a central QoS registry. They describe an "extensible QoS model", arguing that web services are so diverse that a single, static model cannot capture all of the relevant QoS parameters, and that domain-specific parameters for one service may be completely inapplicable to others. They define a number of generic quality criteria, including execution price, execution duration, and reputation. Execution price is the monetary cost the service requestor must pay the service provider to use the service. Execution duration is simply the time it takes, in seconds, to call the service and get the result back. Reputation is a parameter that can be specified by each user for any particular web service he or she uses.

Their QoS registry is similar to the approach taken in this paper, explained in detail in Section 4. Their registry takes in data collected from the clients, stores it in a matrix of web service data in which each row represents a web service and each column a QoS parameter, and then performs a number of computations on the data, such as normalization. Clients can then access the registry, getting rankings of web services based on their individual preferences. [12]

Much of the material drawn upon by the industry for web services has been standardized by the W3C, including most of the core underlying technologies: XML, upon which the encoding mechanisms of SOAP and XMLRPC are based; SOAP, and WSDL. However, one area that the W3C has been pursuing since the late 1990s has been the semantic web, which holds much promise for the future of web services.

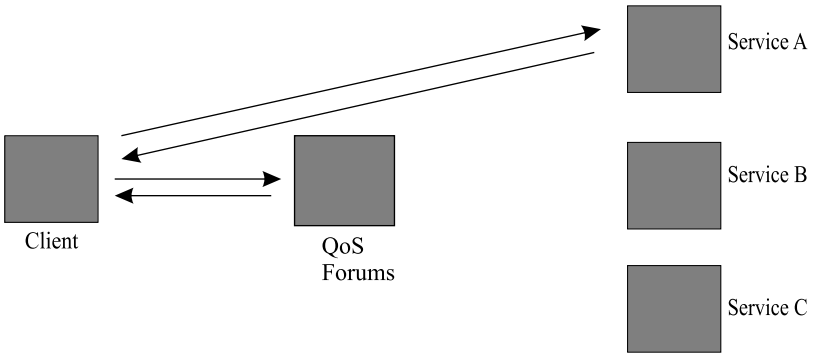

Figure 3: Selecting the Best Web Service through Reasoning and the QoS Forum

3.2 The Semantic Web and Semantic Web Services

The W3C published its semantic web roadmap [2] in 1998. Since then, there has been much activity in this area. The promise is an approach that "instead develops languages for expressing information in a machine processable form" [2], rather than the current web which is geared more to human-to-human interaction.

Machine understandable information, according to the roadmap, is presented in a basic assertion model: the Resource Description Format, or RDF. The basic model contains assertions (Subject-Verb-Object statements), and quotations, which are assertions about assertions [2]. An example mentioned earlier is the fragment of RDF code about the creator of a resource. In that example, one such S-V-O statement would be "Eric Smith is-full-name-of-creator-of resource x". Because of this very basic model, the language is limited in what it can do. The authors point out that it does not contain any concept of negation or implication; these and other limitations are addressed in the languages that have been built on top of RDF: DAML+OIL [9], OWL [15], and OWL-S [6]. RDF is built on top of XML. An example of RDF markup can be found in Figure 4.

RDF Schema is a language for describing RDF vocabularies, and it provides a way to talk about RDF resources in known terms [4]. Though it provides a mechanism for writing ontologies, it is rather minimal. It allows for classes, subclasses, and inheritance, but does not allow for element cardinality (necessary if talking about booleans, real numbers, and so on).

DAML, the DARPA Agent Markup Language, is a language built on top of RDF. In addition to the advantages provided by RDF, DAML+OIL (the DAML language plus the Ontology Inference Layer) provides a number of useful constructs. At a basic language level, it contains things such as bounded lists, basic datatypes (through the use of XMLSchema), and enumeration of data values. At a logical level, it provides negation (through the use of the <daml:ComplementOf> tag), disjunctive and conjunctive classes, as well as necessary and sufficient conditions for membership, and inverse and transitive properties. RDF and RDF Schema have none of these things. RDF and DAML are well-supported in packages such as HP Labs' JENA library, which provides classes for RDF, DAML, and other languages. A good analysis of the features of DAML+OIL, as well as how it compares to other semantic markup languages, has been done by Gil and Ratnakar [8].

OWL-S, a language that has been getting much attention lately, is the Ontology Web Language for Services, and at the time of writing is at version 1.0. For all previous versions, it was known as DAML-S. The aim of OWL-S is to define an ontology for web service discovery, composition (using multiple services together in such a way that it appears as only one to the user) and interoperation, invokation, and execution monitoring [6]. The authors admit that no work has yet been made on the last point, but that it should be included anyway because they felt it was important. Their structuring of the ontology is motivatated by the need to provide what the service requires of the user, and what it provides for them; how the service works; and how it is used [6].

The service *profile* describes what the service does. This is interesting because it provides the sort of information needed by, as the authors put it, "a service-seeking agent" [6]. Traditionally, this sort of information would be semantic-less and stored somewhere in a UDDI registry: perhaps a fragment of text like, "a dictionary service for Irish Gaelic." OWL-S provides authors of web services a way of describing their services semantically, so that search-agents or the like don't have to guess by looking for keywords.

The service *model* describes what happens when the service is carried out. As the authors point out, it could potentially be used by service-seeking agents for a second tier of decision making [6]. Should a service's profile match that which the service-seeking agent is looking for, the agent could examine several candidates' models as a consideration for which one to choose.

Finally, the service *grounding* specifies exactly how a web service may be accessed. A grounding will typically specify "a communication protocol, message formats, and other service-specific details such as port numbers used in contacting the service" [6]. This seems at first glance to be similar to WSDL, and the authors later confirm that "a OWL-S/WSDL grounding uses OWL classes as the abstract types of message parts declared in WSDL, and then relies on WSDL binding constructs to specify the formatting of

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>
</rdf:RDF>
```

Figure 4: RDF Code Describing the Creator of a Web Resource

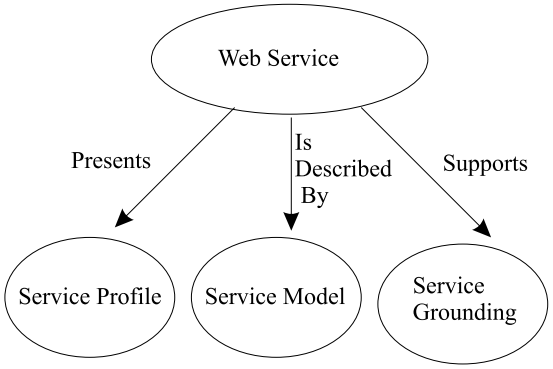

Figure 5: The Service Profile, Model, and Grounding of OWL-S

the messages." The OWL-S service grounding is not meant to replace WSDL, but rather to complement it, as "the two languages do not cover the same conceptual space" [6]. Both OWL-S and WSDL are languages built on top of XML, so it is easy to build some OWL-S constructs on top of WSDL. The three core features of OWL-S, the service profile, model, and grounding, are shown in Figure 5. The diagram is similar to one found in "OWL-S: Semantic Markup for Web Services" [6].

OWL, the Web Ontology Language provides three sublanguages. The first, *OWL Full*, provides no guarantees as to the computability of any conclusions within the language. The second, *OWL DL*, provides less expressiveness than OWL Full, but is guaranteed to be decidable. The third, *OWL Lite*, is a minimal sublanguage designed for, as McGuinness and van Harmelen put it, "those users primarily needing a classification hierarchy and simple constraints" [15].

That there are several sublanguages to select from is indicative of the fact that different needs are served by different languages. This is not only true of the OWL sublanguages, but also of the semantic markup languages in general. The choice of which language to use is an important application-specific decision to be made by the application designer.

McIlraith et al. address this issue in "Semantic Web Services" [16]. In it, the authors describe their system, which uses semantic markup in DAML+OIL to mark up a number of existing commercial web services: Yahoo's driving direction information service and United Airlines' flight booking service. Their markup provides declarative advertisements of the service, like the service profile of OWL-S; declarative APIs, built on WSDL, to allow for automatic service execution; and finally, specifications of the pre-requisites and consequences of individual service use necessary for automatic discovery, composition, and interoperation [16].

The authors make a strong case for the use of DAML+OIL for web services. RDF, they argue, is insufficient as a general semantic markup language due to its lack of expressive power and an underspecified semantic [16]. Since the paper was written in 2001, however, RDF's semantics have been undergoing constant extension and revision. The current W3C Recommendation on RDF Semantics dates from February 10, 2004. Prior to that revision, the previous recommendation dated from December 15, 2003. RDF is one of the central building blocks of the semantic web, and the W3C remains committed to making it better. That several semantic markup languages use it as its base is testament to the strength and elegance of RDF's basic representation model, and of its extensibility. That said, it is probably not suitable for all applications. OWL-S and DAML+OIL are built on top of RDF for a reason. They offer more functionality than RDF. However, there is an increased overhead, and in some cases, no guarantees as to computability. The choice of which markup language to use should be made on a case-by-case basis, taking into account the needs of the system.

Semantic web languages and ideas can be used to tackle the selection problem, as mentioned earlier. Maximilien and Singh write of a system whereby agents serve as proxies for web services [14]. The agents select services based on reputation, making their choices by talking with other agents about reputations of services, with unknown services being selected if they are recommended by trusted third-party agents. Discovery of these services occurs through UDDI registries that have been augmented to allow ratings based on QoS attributes.

The languages of the semantic web provide the ability to both reason about marked-up data, as well as simply to classify, allowing for multiple implementations to talk about the same qualities through use of data ontologies. They build on each other. XML is the foundation, with RDF built on top of that. RDF Schema is built on RDF. OWL and DAML build on RDF and RDF Schema, and OWL-S builds on OWL. The use of these languages can be of great benefit, allowing heterogenous systems to talk about resources using the same set of terms and facilitating intercommunication.

3.3 Reasoning with Expert Systems

An expert system, according to Peter Jackson, is "a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or solving advice." [11] An expert system "*simulates human reasoning* about a problem domain, rather than simulating the domain itself," and solves problems by "*heuristic or approximate methods* which, unlike algorithmic solutions, are not guaranteed to succeed." [11]

Most rule-based expert systems now use the RETE algorithm [11]. This algorithm is efficient for pattern-matching due to a tree-structured sorting network that reduces the number of iterations over productions. The algorithm, explained by Jackson, works as follows: patterns on the left-hand side of the production are compiled into the network. Then, the match algorithm computes a *conflict set* (a set of elements that, given the current conditions, cannot be all working correctly) for the current cycle by processing the network. The iteration on the working memory between cycles is eliminated due to the processing of a set of tokens which indicate which patterns match various elements of the working memory. This set of tokens is updated when the working memory changes [11].

4. APPROACH

As described in Section 2, my approach to the selection problem is to build on to an existing web service client an augmentation that allows for reporting of, and reasoning on, the user's experience in using particular web services. These web services currently must be syntactically identical, though this could be changed in future versions of the system; see Section 7.

The two main parts of the system are the augmented client and the QoS forums. The web services are not modified at all, and can be seen to be separate from the rest of the system. The representation of QoS data, the semantic models, bear detailed explanation.

4.1 The Semantic Models

The semantic models model interactions between the client and a web service. Each client may have zero or more models; in fact, if the client updates far less frequently than it interacts (these are discussed further in Section 4.2), then it may have many semantic models.

Each semantic model contains a number of properties. For the purposes of the current implementation of the system, generic QoS parameters were chosen so that they can be applied to any web service:

- *Availability* measures whether or not the client can connect to the web service. It takes a value of 0 (cannot connect) or 1 (able to connect).
- *Reliability* refers to whether the operation the client wishes to perform can be performed, and whether or not it returns the type it was specified to. It takes a value of 0 (unable to perform the operation, or received a bad return type) or 1 (able to perform the operation and got the specified type in return). If a service is not reachable, the reliability is assumed to be 0 for that interaction.
- *Execution Time* is the time, in seconds, that it takes to *attempt* to access the service, perform the requested operation, and get a value in return.

For reliability, the specified return type is the one specified in the WSDL file used to build the client. The client uses WSDL2Java in the Apache Axis libraries to create the classes used to access the web services. Access methods are created based on the WSDL file available at generation-time. It is certainly possible that operations on the web service could change signatures, or even be removed, after the access objects have been generated.

It should be noted that execution time may be lower when failure occurs than when accessing a reliable service has occurred. Because of this, the default weights on execution time tend to be much lower than that of availability and reliability. These weights are specified in an external file editable by the user.

These parameters were chosen because they were generic enough that they could be applied to any web service. Because they are so generic, future versions of this system could use the same basic model, but extend it to add more generic parameters, or include domain-specific ones.

From the beginning, my goal was to use semantic markup languages for the semantic models. The basic premise of my work, web service selection by using QoS parameters, has been explored at least once before (though in a different way), by Liu et al. [12]. They do not, however, make use of semantic markup, nor of client-side reasoning, which is discussed in Section 4.2. Because web services are web-based – that is, they communicate over HTTP using structured XML – and because much work has already been done in developing languages to describe web-based resources, the choice was made to represent the models with one of the popular semantic markup languages.

RDF was chosen to act as the underlying representation for the data, with an ontology describing the data written in the OWL Lite sublanguage. Because all that was needed was a simple classification hierarchy and simple datatype restrictions, and because the external reasoning was done with the JESS engine, OWL Lite seemed more appropriate than the bulkier OWL DL or OWL Full. RDF Schema was a possibility for writing the ontology. However, it does not allow for cardinality constraints on values, which is required when talking about various parameters. At the moment, these parameters are only boolean and real numbers, which have specified mappings within XML Schema. However, for complex datatypes, this is not the case, and it would be useful to specify constraints if future versions of the system were to progress in that way. The clients could talk about various parameters and not have to worry about checking to see if the value of certain parameters falls within the required range. As such, OWL was chosen over RDF Schema as the ontology language for the semantic models.

4.2 The Augmented Client

The augmentation currently consists of three parts:

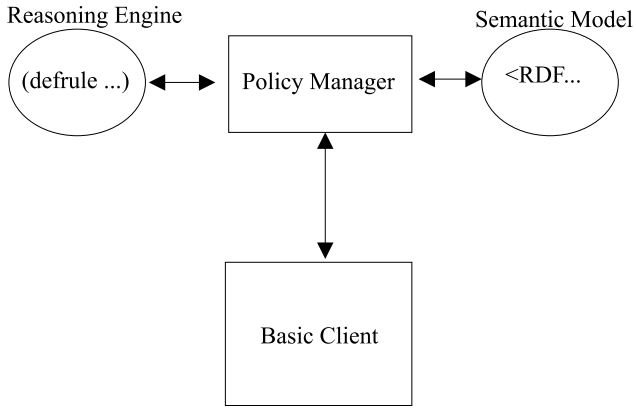

Figure 6: The Automated I-Help Client with Augmentations

1. A *policy manager* acts as a coordinator of the client's calls with updates to the QoS system.

2. A number of stored *semantic models* each capture one interaction with the client's web service.

3. A *reasoning engine* is capable of taking in a list of semantic models, extracting information from them, and based on its analysis, selecting the service best for the user.

The policy manager controls such aspects as how often the client makes calls to the web service, and how often updates are sent to the central QoS forums. In a system where a user would be using the client (rather than my current automated client), the policy manager would only control how often updates were sent to the QoS forums, leaving the use of the web service to the user.

When the client is first started, the policy manager connects to the QoS forum, and queries them for all the information stored about a number of user-supplied web service locations. The QoS forum returns a vector of semantic models, each of which represents one interaction with a web service by a particular client.

This information is then passed to the reasoning engine. The engine is an expert system written in JESS, the Java Expert Systems Shell. JESS was chosen for a number of reasons. First, expert systems are a well-known and well-studied area of AI, and are suitable for reasoning over a number of rules and facts. Second, the RETE algorithm has been shown to be fast and efficient for large data sets [7]. Scalability of rule- and fact-sets was an important consideration in choosing a reasoning method. While the current rule- and fact-sets are relatively small, the ability to scale up to dozens or hundreds of rules without a significant increase in cost was important.

The first thing the reasoning engine does is take in the information about the web services' availability, reliability, and execution time. It processes these into separate data structures, and then based on user-specified weights, creates a weighted sum for each:

$$\sum_{i=1}^{n} w_x x_i$$

In this equation, w_x is the weight for QoS parameter x. x_i is the i^{th} reported parameter x for the web service whose data we are currently processing. From there, we have a simple evaluation: take the highest weighted sum as best.

The current simple evaluation rule is as follows:

```
(defrule updateBestWS
   (and (ws ?x ?y) (best-time ?z)) =>
   (try-update ?x ?y ?z))

(deffunction try-update (?x ?y ?z)
   (if (> ?y ?z) then
      (and (retract (fact-id ?bt-id))
      (retract (fact-id ?bs-id))
      (bind ?bs-id (assert (best-service ?x)))
      (bind ?bt-id (assert  (best-time ?y)))
)))
```

The facts for the JESS reasoning engine are generated at runtime. Because the augmented client does not know beforehand what services it may select from, the reasoning engine generates the facts based on the information provided to it from the QoS forums. The facts have the following basic form:

```
(assert (ws http://ws.location... 3.78))
```

The first part of the triple is the indication that this fact is about a web service. The second part is the location of the web service. The third is the weighted sum as computed by the reasoning engine earlier.

The system also keeps track of the best service as follows:

```
(bind ?bs-id (assert (best-service http://...)))
(bind ?bt-id (assert (best-time 0)))
```

These allow us to keep track of the best service (with bs-id) and the best weighted sum (bt-id). The clients have built into them an initial web service location, which is initially taken to be the best service. If our information shows that another service is better than the default weighted sum (taken to be 0 in the current implementation), then the RETE algorithm will match the facts against the rules, and update the best-service and best-time tuples. Once JESS has been allowed to run on these facts and rules, the reasoner extracts best service location from the best-service tuple. This is the service that the reasoner has computed to be best for the client. This value is returned to the client, and if it is different from the current web service, the client is switched over to the new service.

From there, execution continues as normal on the best web service. Semantic models are stored each time an interaction is made, and when the policy manager indicates that

the system is ready to update, it contacts the QoS forums to upload the semantic models. This process is more fully described in Section 4.3.

4.3 The QoS Forums

The QoS forums are a web service designed to store information about clients' experience with particular web services. A web service is wrapped around a database which stores semantic models reported to the service.

The semantic models, as explained in Section 4.1, are a combination of RDF and OWL. Besides these, the forums also keep track of the date, the sender's IP address, and the web service's location. The first two are tracked because if the system is expanded in the future, they could provide useful information in addition to what's already in the semantic model. The reasoner could find out, for example, all clients reported that a web service x was not available on a certain day, and perhaps conclude that this is an exception rather than the rule. Alternatively, if one IP range keeps reporting experiences that are wildly different from every other client, and at a much higher rate, it might bias its calculations against that IP range, concluding that it is flooding the forums. At the moment, however, the reasoner is not capable of making realizations such as this. The last feature kept, the web service's location, is just so that the database can more easily return ranges of semantic models, rather than all of them.

Currently, the forums are centralized: they are a single web service wrapped around a database. A future extension might be to have decentralized storage, with each client maintaining a database of its own experiences. That way, a single point of failure would be eliminated, though at the cost of additional overhead in retrieving data.

5. EXPERIMENTATION

5.1 Dynamic Service Selection

To prove that the system implemented all the concepts discussed earlier, clients were written for the I-Help web services. I-Help is a system designed by the ARIES laboratory at the University of Saskatchewan. Its purpose is to allow students to find help for problems relating to their computer science courses. Recently, it has been extended into web services, allowing for posting, reading, and other operations without having to use the traditional web-page-based interface.

In addition to the main I-Help web service, two replicates were created. These replicates vary from the main service in the amount of time they take to process requests. Clients were generated to use each service, and 500 semantic models for each service were submitted to the QoS forums through the clients' interactions with the I-Help services.

The statistics for the main service and its two replicates are found in Table 5.1. $\bar{a}$, $\bar{r}$, and $\bar{e}$ refer to the mean availability, reliability, and execution time, respectively. σ_a, σ_r, and σ_e refer to the standard deviation of the availability, reliability, and execution time.

Each client gathered data over the course of about six hours. As can be seen from the data, no service had 100% uptime.

	Main Service	**Replicate A**	**Replicate B**
$\bar{\mathbf{a}}$	.814	.964	.978
$\bar{\mathbf{r}}$	.814	.964	.978
$\bar{\mathbf{e}}$	4.672	7.93	9.774
σ_a	.389	.187	.147
σ_r	.389	.187	.147
σ_e	2.46	9.970	2.143

Table 1: Data Gathered About The Web Services

There were a number of crashes. Whenever this happened, the services were promptly rebooted. Thus, there was imperfect availability and reliability.

Code was used on the client side to simulate network delays and server-side slowness. The first service was the quickest, followed by the first and second replicates. However, replicate A varied greatly in its execution times, as can be seen by its standard deviation.

When the default weights were defined, execution time was deemed to be most important; however, availability and reliability weren't to be sacrified either. The default weight for both availability and reliability is 10; the default weight for execution time is -2.

With these weights defined, the service the client should have reasoned to be best, given the reasoning rules from Section 4, is the original I-Help web service, as the weighted sums for the original service and its replicates are 6.935, 3.410, and 0.0111, respectively. When the client ran, gathered this data, and ran it through the expert system, it found that the original I-Help web service provided the best service, and transparently switched over to that service. The client was originally generated to run on one of the replicates.

5.2 Scalability of the Clients

As seen in the previous section, the proof-of-concept works as intended. The next question, though, is to determine how well these clients could scale up in usage.

To judge the scalability of the clients, there are four identified parameters to be studied in terms of resource consumption. The first is the *query cost*. The query cost is the cost incurred when the system queries the QoS forums for information on a particular web service. The second is the *analysis cost*, which is the cost of taking the data provided by a query, parsing it in such a way that the JESS reasoner can make sense of it, and running the JESS-ready facts and rules. The third is the *monitoring cost*, which describes the cost of monitoring calls to the web service, and preparing a semantic model based on the findings. The fourth is the *reporting cost*, which specifies what is required to report the monitored data to the QoS forums.

To judge these parameters, two factors were considered: first, how often they are used by the client; and second, where the main bottleneck lies. The result can be seen in Table 5.2.

The number of times these parameters are used is due to

Cost	Times Used	Bottlenecks
query	constant	memory, bandwidth
analysis	constant	memory
monitor	varies by calls	memory
report	varies by policy	bandwidth

Table 2: Bottlenecks of the Identified Parameters

the design of the system. Currently, selecting the best web service is a one-time event done just after the client starts up, and the analysis, done after the data arrives, also occurs only once. However, the bandwidth and memory requirements can be rather steep. In our test cases, 500 semantic models for each service were stored. The bare XML representation of the models (not the objects in the system, which would have a greater size) had a mean size of 3.7KB. 500 such objects would then take at least 1.81MB if only their XML structure were stored. But when each is wrapped in a Java object, and all of those objects are stored within a vector, the memory costs suddenly jump by 7.1MB. This was not a huge amount on the main development machine; however, transferring that amount of data over the network is not insignificant. As the amount of interactions increases, the amount of data sent over the network will only increase, as will the memory consumption by the client. This is something that will need to be addressed in future versions of the system.

Monitoring and reporting occur variably. Monitoring is handled by the policy manager, and occurs each time the client attempts to make a call on the target web service. Despite this, though, they tend to require fewer resources than do query and analysis. Monitoring simply creates a new semantic model after a call is made, and queues it in the unsent list of semantic models. These models are each roughly the same size, and take a fixed amount of time to create. Reporting occurs whenever the policy manager says to (at the moment, this occurs at the same time interval as calling the service). It takes no more memory than does monitoring, but bandwidth is a potential bottleneck, especially if large numbers of semantic models have been stored but not sent.

While the concept implemented in the system could be scalable, there are a number of issues that would need to be addressed. There is a large memory and bandwidth overhead when reasoning about the best service that will only grow with increased numbers of interaction snapshots within the QoS forums. However, the rest of the interactions should go smoothly, as the costs of monitoring and reporting are fairly negligable.

6. CONCLUSIONS

The process of selecting a web service does not have to be a static, design-time decision. Instead, it can occur dynamically, with web service clients deciding which service to use. In this implementation, web service clients reasoned over the experiences of others clients as to which service to select. The experiences concerned generic QoS parameters: availability, reliability, and execution time. To represent a model of its knowledge, the clients in the implementation used a combination of RDF and OWL. These experiences were stored in a central forum system available as a web service to any interested party. To reason about the best service, an expert system, written in JESS, was used to analyze the data received from the QoS forums. When done, the client was updated to call the new web service.

This approach, while generally good, only suffered from a few issues relating to scalability. The clients operated on an "all knowledge is potentially useful" assumption, which meant that they requested all of the interaction snapshots from the QoS forums. If the forums had hundreds or more interactions stored about a particular web service, the analysis process would take a noticeable amount of time as the data from the semantic models had to be first parsed by the reasoner, and then reasoned over. However, this is a one-time cost incurred when the client starts up, and the variable-time occurances of both monitoring and reporting had far fewer costs associated with them. Overall, the approach of reasoning over experiences seems a promising one for the problem of web service selection.

7. FUTURE WORK

In this paper, the web service selection problem was tackled purely from a past-experience standpoint. There were other approaches that, for the purposes of time, were neglected: for instance, negotiation, or semantic suitability. In the future, the system could be extended to have full OWL-S descriptions for the web services to fully talk about what they offer, and have the clients take that into consideration, too. In the case where there is a cost associated with a web service, the clients could perhaps also apply negotiation strategies.

As for the forums themselves, they are currently a standalone web service. There are currently no mechanisms in place to prevent flooding of the database with bogus semantic models, as there are in Liu, et al [12]. One possibility would be to set up a peer-to-peer-like forum system in which each client contains its own database and stores its own experiences, but no-one else's. This, combined with trust-based rules in the expert system, would allow for the clients to build trust-based networks of various clients in which clients that offer advice radically different from the others could be placed under closer scrutiny. As for the semantic models, future versions of the ontology could be extended to allow for more generic, or possibly domain-specific, parameters.

A major assumption in my work is that all clients are operating under the same circumstances. Information about the clients' experiences are kept and stored, but not information about the context of the clients themselves. Information such as available memory, CPU load, number of running processes, and network connection speed, among other things, would be extremely valuable in the reasoning process, but are not included in the current version of the system.

Currently, the user specifies his or her preferences by way of weights. Another way of doing this would be by using a set of constraints. At the moment, the user places higher weights (or in the case of execution time, lower) on parameters that he or she believes to be most important. It might be more intuitive for the user to instead express his or her preferences through constraints: for example, "find me a web service

which has 95% availability, 100% reliability, and a mean execution time of less than eight seconds". In cases where no constraints are satisfied, the service which violates the fewest number of the user's constraints could be selected.

The approach taken, that being evaluation of experiences, relies on the user supplying choices of web services. This approach could be combined with conventional UDDI or OWL-S technology to abstract away one more level from the programmer. The client could get a list of prospective web services from a UDDI registry or OWL-S profile server. In this way, the client could get not just others' experiences with web service locations, but what those web services actually purport to provide. Also, combining this approach with client examination of a service's OWL-S service grounding could tell the client whether the service is syntactically identical to the others considered, which is a current requirement of the system.

Finally, there is the matter of bandwidth. XML-based languages are an excellent basis for description, but suffer from all the extra space that the tags take up. A possible extension of the QoS forums and the clients could be to send their experiences in a compressed format, if it were found that it would be faster to compress/send data/decompress than simply to send data. The QoS forums could also be modified to not send all the data known about a particular web service. This would alleviate strains on the client's memory usage, as well as bandwidth.

8. ACKNOWLEDGEMENTS

I am indebted to Chris Brooks for setting up, and providing the interface to, the I-Help web services. I would also like to thank fellow graduate students whose discussions on web services and my research has been extremely valuable: Chris Brooks, Kamal Elbashir, Tay Hock Keong, Mike Winters, and Chris Wormon.

9. REFERENCES

[1] Bellwood, T., Clement, L., and Claus von Riegen, E. Uddi version 3.0.1. http://uddi.org/pubs/uddi_v3.htm.

[2] Berners-Lee, T. Semantic web road map. http://www.w3.org/DesignIssues/Semantic.html, September 1998.

[3] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D. Simple object access protocol (soap) 1.1. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[4] Brickley, D., and Guha, R. Rdf vocabulary language description 1.0: Rdf schema. http://www.w3.org/TR/rdf-schema/.

[5] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl.

[6] Coalition, T. O. S. Owl-s: Semantic markup for web services. http://www.daml.org/services/owl-s/1.0/owl-s.pdf.

[7] Forgy, C. Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence 19*, 1 (1982), 17 – 37.

[8] Gil, Y., and Ratnakar, V. A comparison of (semantic) markup languages). In *Proceedings of the* 15^{th} *International FLAIRS Conference* (2002).

[9] Horrocks, I., van Harmelen, F., and Peter Patel-Schneider, e. Daml+oil. http://www.daml.org/language/.

[10] Hugo Haas, A. L. Web services activity statement. http://www.w3.org/2002/ws/Activity.

[11] Jackson, P. *Introduction to Expert Systems, Second Edition*. Addison-Wesley Publishing Company, 1990.

[12] Liu, Y., Ngu, A., and Zheng, L. Qos computation and policing in dynamic web service selection (to appear). In *Proceedings of the WWW 2004* (May 2004).

[13] Manola, F., and Miller, E. Rdf primer. http://www.w3.org/TR/rdf-primer.

[14] Maximilien, E. M., and Singh, M. P. Agent-based architecture for autonomic web service selection. In *Workshop on Web Services and Agent-based Engineering at Autonomous Agents and Multi-Agent Systems* (2003).

[15] McGuinness, D. L., and van Harmelen, F. Owl web ontology language overview. http://www.w3.org/TR/owl-features/.

[16] McIlraith, S., Song, T. C., and Zeng, H. Semantic web services. *IEEE Intelligent Systems 16*, 2 (March/April 2001), 46 – 53.

[17] Mitra, N. Soap version 1.2 part 0: Primer. http://www.w3.org/TR/soap12-part0/.

[18] Winer, D. Xml-rpc specification. http://www.xmlrpc.com/spec.