Lowering the entry level: Lessons from the Web and the Semantic Web for the World-Wide-Mind

Ciarán O'Leary [1] and Mark Humphrys [2]

[1] School of Computing, Dublin Institute of Technology, Kevin St, Dublin 8, Ireland
coleary@maths.kst.dit.ie, comp.dit.ie/coleary
[2] School of Computer Applications, Dublin City University, Glasnevin, Dublin 9, Ireland
humphrys@computing.dcu.ie, computing.dcu.ie/~humphrys

Abstract. The "World-Wide-Mind" (WWM) is a scheme in *sub-symbolic AI* (numeric and behaviour-based AI, neural networks, animats, artificial life, etc.) for constructing complex agent minds (by which we just mean action-taking systems) through multiple authors. Authors put their (sub-symbolic) agent minds *online*, and other authors use these minds as components in larger minds.
This paper does *not* discuss in detail exactly what the WWM is *for* in sub-symbolic AI, for which see [8], [9]. Instead it will treat this as another problem domain in which to apply Semantic Web ideas. The substance of this paper is taken up with considering *how* Semantic Web ideas can be applied given some particular properties of this specific problem domain, namely: (1) We are trying to get existing sub-symbolic AI researchers to "publish" their algorithms online for remote re-use by others (as, essentially, *Web Services*). (2) The target audience *are* programmers, but not network programmers. (3) These algorithms are often *unique*, not commodities. As a result, we are likely to be very "forgiving" of whatever the researchers *do* put online.
The standard approach with the Semantic Web has been to aim the technology at network programmers, and assume that *tools* can hide this complexity from non-specialist users [7]. We argue that this will not work in this case. The Web itself showed a different approach, where the technology *itself* could be approached by the non-specialist, at least at the entry level. We adopt this approach. We construct an extremely low entry level, which rejects: (a) models of programs online that require network programming or complex APIs, and: (b) models of data that are *unforgiving* - where data must be well-formed or will be rejected. We explain why these ideas will not work in this problem domain.
This simple entry level does not compromise usage of advanced Web Services and Semantic Web concepts at higher levels. The discussion of design decisions here may have implications for other areas of the Semantic Web where the target audience may be programmers but not network programmers.

1 Introduction

The "World-Wide-Mind" (WWM) [9] is a scheme in *sub-symbolic AI* (numeric and behaviour-based AI, neural networks, animats, artificial life, etc.) for constructing complex agent "minds" (by which we just mean action-taking control systems) through multiple authors. Authors put their (sub-symbolic) agent minds online, and other authors use these minds as components in larger minds. This works in sub-symbolic AI because competition is resolved using schemes like competing *numeric weights*, rather than through explicit symbolic reasoning. So we can define a *sub-symbolic* communication protocol based on numbers which avoids (for the moment) the difficult problems of knowledge representation and "agent communication languages" [10] of the *symbolic* AI level. We still want to put these programs *online* in a communicating network though, and it is here that ideas from Web Services and the Semantic Web will still be useful.

This paper does *not* discuss in detail exactly what the WWM is *for* in sub-symbolic AI, for which see [8], [9]. Instead it will treat this as another problem domain in which to apply Semantic Web ideas - like other such domains of specialised research communities, such as chemistry (the Chemical Markup Language, www.xml-cml.org) or molecular biology (the RiboWeb Project, smi-web.stanford.edu/projects/helix/riboweb.html).

The substance of this paper is therefore taken up with considering *how* Semantic Web ideas can be applied given some unusual properties of this problem domain. The basic unusual feature of this problem domain is that it is aimed at *programmers who are not network programmers*.

1.1 Particular properties of the WWM problem domain

The WWM problem domain has the following properties:

1. We are trying to get existing sub-symbolic AI researchers to "publish" their algorithms online for remote re-use by others (as, essentially, *Web Services*).
2. The target audience *are* programmers, but not network programmers (and unlikely ever to become network programmers). Any scheme that requires them to become network programmers (or indeed learn any new concepts) will fail (will fail in the sense that most algorithms will remain offline).
3. These algorithms tend to be *unique*, not commodities. A business putting airline-ticket systems online will hire programmers with generic, "commodity" skills in databases, networks, etc. But sub-symbolic AI is driven by unique individuals whose work is often not easily replicable by anyone else. Often, few people may fully understand the algorithm. Indeed, one of the problems with this field is how few people try out each other's algorithms [2], [3], [5]. In general, if the author does not put his algorithm online then *no one will*.
4. Businesses will just *hire* network specialists to construct their web services. But individual sub-symbolic AI researchers will not or cannot. They must do it alone or not

at all.

5. An important consequence of this is that if, after lengthy persuasion, the researcher Bloggs is finally persuaded to put the "Bloggs neural architecture" online, then we may use his server even if it deviates badly from official standards (e.g. outputs badly-formed XML). We will have to be extremely tolerant of whatever he does publish, because we can't *get* this service anywhere else. If Bloggs is important enough, then *we will adapt to his server*.

This problem domain is clearly a long way from e-commerce. But many similar domains exist, especially within science, that are planned to be addressed by Semantic Web models. We now see how ideas from the Web and the Semantic Web apply to this problem domain.

2 Lowering the entry level

One of the defining features of the Internet over the last 15 years can be seen as the attempt *to entice large numbers of people to put their data and programs online*. This attempt is ongoing, and much data and programs remain offline.

A protocol or scheme is by definition *successful* if large numbers adopt it. This is the only definition of success that the WWM is interested in. There may, of course, be a *tension* between designing a rich, functional protocol and a protocol that will be adopted (the Hypertext 91 conference famously came down on the first side of these two when rejecting the paper describing the Web). But, in the case of the WWM at least, if adoption is not widespread then most of the functionality is wasted.

2.1 The Web

The Web has clearly been the paradigmatic success story in terms of adoption, persuading millions of people to put their data online. Most of these use publishing tools to do so, but one can also publish on the Web using just a text editor and a couple of markup tags. One can also "publish" a program on CGI with a text editor and knowledge of just one HTTP header line (Content-type) and a couple of markup tags. No new programming language need be learnt. In summary, the entry level is very low.

2.2 The Semantic Web

The Semantic Web has made a deliberate decision to raise the entry level. The standard approach has been to aim the technology at network programmers, Semantic Web experts, and other specialists, and assume that *tools* can hide this complexity from non-specialist users. For instance, users who want semantic markup in their pages are not expected to insert it by hand. They are expected to use web authoring tools [7]. As a result, the technology is forbidding for the non-specialist, even for programmers.

2.3 The WWM

But the world is not divided into network programmers and *non-programmers* who just want to mark up pages. The WWM problem domain illustrates a class of applications that has been given little attention by Semantic Web researchers - programmers, who are *not* network programmers, yet whose programs could usefully go online. Such programmers may be willing to learn a few XML tags in order to get their programs online, but the idea that they would be willing to learn about well-formedness, DTD's and Schemas (let alone more advanced concepts like ontologies) is totally unrealistic. For most of them, this (putting their algorithms online) is incidental to their main careers, and will not happen if it requires a big investment. This aspect - of appealing to an audience of programmers, yet one that will *resist* learning new programming tools or standards - is what makes this audience unusual. (Though we suspect there may be other audiences like this that the Semantic Web will need to address.)

Our approach, therefore, is to take the "Web" approach and lower the entry level so these programmers can approach the technology directly, even if they do not use any of the advanced concepts. This simple entry level should not *preclude* the use of more advanced concepts at higher levels of the system.

2.4 WWM Mind servers and World servers

A word needs to be said about what we envisage by putting sub-symbolic agent minds "online". We envisage authors putting (sub-symbolic) minds online as "Mind servers" to which messages can be sent from remote clients. A typical message for a Mind server would be *"Given that the input state is x, what output state do you generate?"*. The typical "input state" here will be, say, an **n-dimensional vector of real numbers**, such as the input to a neural network (e.g. a visual pattern, or the combined vector of sensory input of a robot). The "output state" could represent a classification (e.g. a decision on the class to which the visual or auditory pattern belongs) or an "action" (e.g. commands sent to robot joints or motors). This will also typically be an **n-dimensional vector of real numbers**, as in the output layer of a neural network. This will cover many problems and applications in sub-symbolic AI, but we will still leave the definition of input and output open and extensible, so that other definitions may be used.

We envisage that the *problem* that the mind is attempting to solve can also be put online, separate from any *specific* attempt to solve it. We call this a "World server". This may receive messages such as *"What is the current state x of the world?"* (the data structure that will be sent as the input state to the Mind server).

Again, for detailed discussion of how this can be applied to existing problems and architectures in sub-symbolic AI (including how multiple-Mind-server systems can be built) see [9], or in considerable detail, [8]. What this paper is concerned with is *given* we have such a requirement for a system of remote invocation, and given the specific nature of the audience we are aiming at, what technology to use?

3 Remote Invocation technologies

3.1 Rejecting local installation

We are setting up a system whereby researchers can re-use each other's sub-symbolic minds (and problem worlds). First note that we have rejected the solution of *local installation*. Given the huge diversity of, and incompatibility of, operating systems, platforms, files, libraries, versions, environments, programming methodologies and programming languages in use in sub-symbolic AI, we view it as highly unlikely that local installation could lead to widespread re-use. We clearly reject the idea, for example, of asking all sub-symbolic AI researchers to use a certain programming language (e.g. Java) or platform. How can one avoid these compatibility problems and allow researchers use whatever platform they want? By *remote invocation*. We look here at some of the major schemes for remote invocation.

3.2 Java RMI and Jini

Java RMI [java.sun.com/products/jdk/rmi] is a Java-only mechanism for locating and invoking services across a network. It allows for a Java object, hosted on any node on the Internet, to be the subject of remote invocations from clients located elsewhere. A client contacts a registry which provides a remote reference to the server object. It then downloads a *stub* object which represents the server object. From that point on the stub handles all communication with the server object - this is completely transparent to the programmer. Jini [sun.com/jini] is an architecture built on top of Java RMI which allows for publication of services and automatic notification of events. Jini applications are grouped together into a *federation*, where they make *services* available to other members of the federation. Clients wishing to use a service provide a serialised Java service object to a central lookup service, requesting similar Java objects, which are then provided. Java provides two simple solutions to the problem of remote objects and services. The majority of the requirements for a distributed service architecture are made transparent to Java programmers. It is however, a pure Java solution, and as such is not available for use by non-Java applications.

3.3 CORBA

CORBA (Common Object Request Broker Architecture) [corba.org] is a language-independent and platform-independent standard for remote invocation that predates either of the Java technologies. This is the OMG (Object Management Group) [omg.org] standard for remote method invocation. Since it is not tied to any platform, it cannot provide the luxury of complete transparency, thus making the creation of a CORBA application somewhat more complex than a distributed Java application. CORBA provides a new set of standards for communication (IIOP), interface description (IDL) and marshalling of data (XDR). The creation of a CORBA application requires that the client and server be fitted

out with an installation of a CORBA product, such as Iona's Orbix [iona.com/products].

3.4 Web Services

Web services are a rapidly emerging standard for remote invocation over the existing network of HTTP servers (rather than requiring that a new network be set up). Web services by their nature are completely open (since they work over the Web), and available to all programmers. The three main protocols which form the core of Web Services are independently regulated by the World Wide Web Consortium and their members. These protocols fulfill the main requirements for a distributed object or service architecture, such as the ones we have seen above. SOAP (Simple Object Access Protocol) [w3.org/TR/SOAP] is the communication protocol, providing the message format and the encoding of complex data types. It is a lightweight protocol built on top of XML, and makes use of several components of XML, such as namespacing, and the data typing of XML schema. UDDI (Universal Description, Discovery and Integration) [uddi.org] provides a means for discovery of services, analogous to discovery in Jini. Since it comes from the e-commerce community, it is primarily a classification and inquiry scheme for business web services. Services are registered with a set of information divided between white pages (address and contact information), yellow pages (business category) and green pages (technical information). Users can send queries to the registry for the service that they require using the *inquiry* API with *find* and *get* messages. WSDL (Web Service Description Language) [w3.org/TR/wsdl] is an XML based language used to describe web services. It describes services at an abstract level, as a connection between two abstract network end points. A WSDL operation ties a request and response together, and provides the various pieces of information required to describe the service. The service is then bound to a particular implementation, by indicating the transport mechanism to be used (most typically SOAP), as well as the actual location of the service.

The design of web services was inspired by a desire to make distributed computing as simple as possible, prompting the following quote from *Byte* magazine: *"Does distributed computing have to be any harder than this? I don't think so."* [16]. And indeed they are correct, it does not have to be any harder than this, unless the designer of a distributed system wants to have distributed garbage collection, activation of objects, objects by reference or other more advanced features. What web services give us is the lowest common denominator. We can write an object using the language of our choice. Anyone in the world can locate and invoke methods on that object once we put it on a web server that is SOAP enabled, and publish its location with a UDDI registry using WSDL. Even the final two steps are not mandatory. Any object accessible through SOAP can be a part of a distributed object system. While it is simple to create a service using SOAP, one must still use an installation of a SOAP interface such as Apache SOAP [jakarta.apache.org/SOAP]. The protocol is based on XML and as such will always need some third party APIs or software in order to process and generate it correctly. This is what made the web service protocols and standards unattractive for our purposes. While

recognising the achievement of the web service community, we still feel that for constructing the simplest possible entry level for programmers for the WWM, it is better to create a cut down version of the web service protocols than to insist upon adoption of them by people who may have no interest in the Web or network programming.

3.5 DAML-S

While proposing a simpler entry level for the WWM, we intend this to co-exist with the incorporation of advanced concepts at higher levels. One of these concepts is description and discovery of services. Entities within the WWM will need to be aware of all the services that other entities are making available. Our protocol, that we will describe below, allows for various (sub-symbolic AI) entities to query each other in order to see what they can do. If one entity says that it can do X, then all other entities will need to share the same understanding of what X means. Web services and WSDL don't provide us with this. A client can be told that a service will accept two strings and return a single integer, but there are no semantics involved in the definition of the service. The closest we get to this is in UDDI where the yellow pages insist upon a business categorisation of the service.

DAML-S [1] is an initiative from the DAML Services Coalition (see below) that intends to create an ontology for web services. It is quite different from the standard web service protocols. It intends to include semantics in the definition of services, so that clients can be sure that the service they are invoking is providing the functionality that they require. DAML-S describes a service using a *Service Profile* (for saying what it does), a *Service Model* (for saying how it works) and a *Service Grounding* (for saying how to communicate with it). In defining the ontology for services DAML-S makes use of a number of the protocols used on the Semantic Web. DAML (DARPA Agent Markup Language) [daml.org] is a language used for describe resources on the semantic web in order to allow better understanding by agents. It is built on top of RDF (Resource Description Framework) [w3.org/RDF/], which in turn is an application of XML. Various other protocols exist for creating ontologies, and marking resources up with semantics. Some of these (such as SHOE [6]) are a great deal simpler than RDF, and as such make themselves more attractive to us for our requirements. We do not intend to include *any* semantic description of services at the lowest layer of the WWM, but we will do so at layers above that. This layered approach will be explained in detail below.

4 Layered Architecture of the WWM

The WWM web services (the Mind and World servers) will have to support a fixed set of messages. Examples of these messages are *new run*, *end run*, *get state*, etc. These form the minimal set of operations that a service must provide. However, services will typically have extra messages that they can support. These messages will be non standard, and will vary across servers, and will be decided upon by the creator of the service. An example of

such a message could be *get neural architecture*, which will return information about the architecture (e.g. the number of hidden layers) of a neural network. This message will be supported by a service which provides access to a neural network, but would have no reason to be supported by other services. It would be impossible for us to define a full set of messages that capture all possible behaviours of all services in sub-symbolic AI, so we decided instead to standardise only those messages that must be supported by everyone. It was necessary to standardize these messages so that *client software* provided by the creators of the WWM could interact with the services. All other (non-standard) messages will be used as the various services communicate with each other.

Our requirements for the WWM are three-fold. We need a way for the various services to be able to find out what (non-standard) messages are supported by other services. Also, the various services must be able to interpret the data produced by each other. Most importantly, all standards and rules for communication must be kept as simple as possible so as not to discourage involvement.

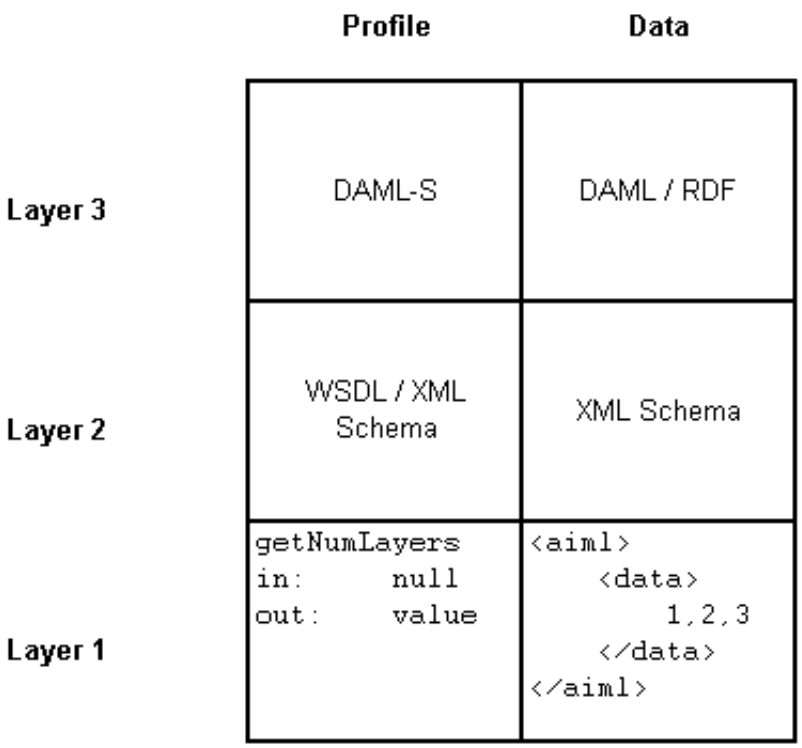

Fig.1. Layered Architecture of the WWM

As a result of our third requirement it was decided that the WWM will be implemented as three separate layers, as shown in Fig.1. The lowest layer (layer 1) provides a simple entry level for anyone interested in creating a service. The middle layer (layer 2) provides a facility for those who wish to describe their services in a more structured fashion. XML Schema can be used here to ensure that services are described correctly, and that the data being shared is consistent among services. The upper layer (layer 3) allows for the full integration of semantic web technologies. Ontologies will be created to describe services, in line with DAML-S as described above. Layer 1 is the only layer as yet implemented. This layer will be the subject of the next section. Layers 2 and 3 will be discussed in the section following that.

5 The WWM entry level

The very basic requirements for a distributed service architecture are transport protocol, mechanism for invocation of service, message format and discovery protocol. We require as simple as possible a solution for each of these, as well as a simple data representation scheme, for the lowest layer of the WWM. Each of these will be described in turn below, but it is imperative to remember that for each decision we made, our fundamental concern was keeping the entry level as simple as possible, so that we are in a position to encourage sub-symbolic AI researchers to make use of our service architecture.

5.1 Simple Transport Protocol

Web services use HTTP for transport. All other communication protocols such as SOAP, are built on top of this. We avoid SOAP because of complexity, favouring AIML, as discussed below, but retain HTTP, because of its obvious benefits, such as simplicity and ubiquity.

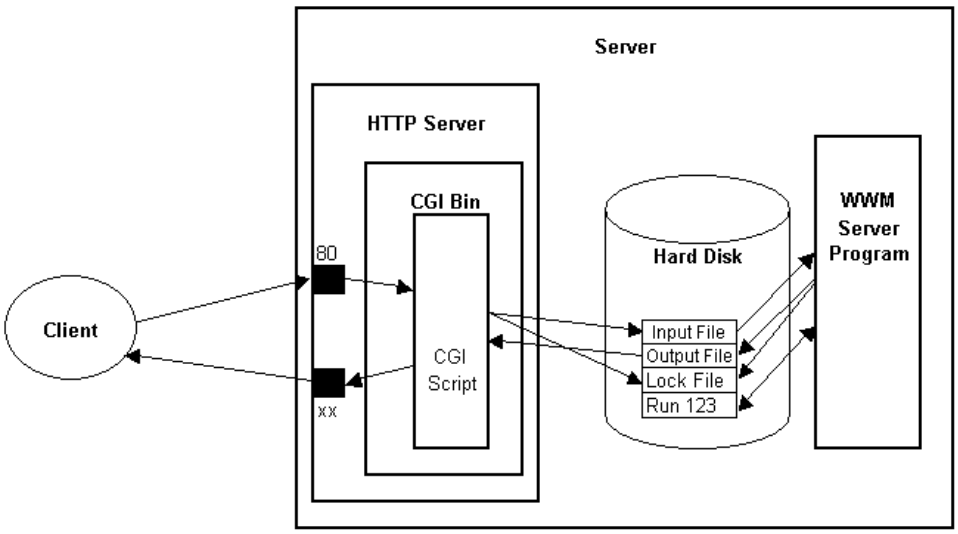

Fig.2. Requirements for a service. Create CGI Script to listen for requests, which are handed on to the server, using (in this example) a series of simple file operations. Server here maintains state using a file for each runid.

5.2 Simple Invocation of Service

Fig.2 shows how a WWM service is designed and invoked. Because of our choice of HTTP as a transport protocol, we can use the CGI facilities of any web server as an interface. A service author simply writes a CGI script which reads in messages and makes the appropriate call to their application (the WWM server). Synchronisation and communication between the CGI script and the application is probably the most complicated part of the architecture. There are obviously many different solutions, depending on time, level of expertise etc. A simple example using lock files is given in [14]. The serviceauthor should also ensure that their service can interact with several different clients (many runs) at the same time. This obviously is only the concern of authors who create stateful services, i.e. ones that need to keep information about the run across several invocations (the obvious analogy is with cookies in web applications). Once again, there are a number of different ways that this could be handled. The protocol messages (to be discussed below) will include a value called *runid*, which is intended to uniquely identify a particular run as started by a client. The server could manage persistence of information using simple files named using the runid.

5.3 Simple Message Format

We created a protocol called AIML (Artificial Intelligence Markup Language) which consists of a core of eight simple messages and a mechanism for each service to add extra messages based on the operations they provide.

AIML is an XML-like language, however it is not XML. The XML specification dictates that any document which does not obey the rules of well-formedness cannot be called XML. While this is perfectly acceptable for situations where documents are generated by network professionals (or generated by programs written by network professionals), as is the case in e-commerce, we feel that for the purposes of the WWM this is unnecessarily strict and could easily be fatal to the level of uptake. Instead we aim to tolerate situations where the messages being exchanged are incorrectly generated and even cater for situations where the data being generated is entirely incompatible with the protocol. The aim is to get the people involved in sub-symbolic AI research to publish their software in order to allow others experiment with it. At the entry level we do not want to impose great restrictions on those wishing to get involved. We do not want to *encourage* our generic researcher Bloggs to break standards, but realistically, *whatever* Bloggs puts online we will adapt to.

We have subsetted XML to get rid of everything except the tags. The tags we use do not have to be grouped together in a well formed fashion, rather they should be grouped together to give a sufficient indication to the user of the intended meaning of the message. This is not a unique endeavour. Common XML [15] is a stripped down version of XML that identifies the most useful parts of XML for certain applications. Similarly, MinML [17] subsetted XML leaving out everything but elements and text, in order to create a language suitable for embedded systems. Others who use XML have found difficulty with its strictness [12], but seem satisfied with applications which inform the author of where the problem exists.

5.3.1 Basic AIML messages

The eight basic AIML messages that form the AIML protocol are as follows: NewRun, EndRun, GetState, GetAction, TakeAction, TunnelMessage, Success and Error. The actual meaning of these messages is not important here. Rather we focus on the generation and exchange of the messages using our loosely defined set of rules. We will show one example. An AIML NewRun message should be represented in the following way:

```
<aiml version=1.1>
    <request type="NewRun">
    </request>
</aiml>
```

When this message is sent to a service the following response should be returned, indicating that the request was successful, and a new run was started:

```
<aiml version=1.1>
    <response type="Success" runid="1234">
        <param name="id" value="0001"/>
        <param name="alttext" value="New Run Started"/>
    </response>
</aiml>
```

AIML responses should include two parameters indicating the status code for the response and a natural language explanation of this (e.g. "everything worked fine"). The message should not be rejected if this was omitted however, as the message still contains valuable information i.e. the run id. Similarly, all AIML messages should be wrapped up with AIML tags indicating the version of AIML being used, but once again, if these are omitted it would still be wise to attempt to extract the meaningful information. Finally, if the document is not well formed, and say the closing tag for the response tag is omitted, the message will still contain the data we require, so we do not reject the message outright, but attempt instead to extract the useful information. In fact, the minimum amount of data that we would expect back from such a request would be:

```
<response type="Success" runid="1234">
```

The only absolute requirement for the response is that the runid be supplied, so we may even be satisfied with a message being returned containing the following text:

```
runid="1234"
```

If this data is returned then we can be satisfied that the request was successful since a runid has been allocated. While it would be more desirable to have the whole message, and it would definitely prevent any misinterpretation due to assumptions made in the absence of certain tags, we believe that simply rejecting messages for reasons like this could fatally damage the uptake of the WWM project.

5.3.2 Forgiving v. Unforgiving data parsing

In order to handle these incomplete messages we must provide client software which attempts to extract as much information as possible from messages. This information can then be used to generate correct messages to pass on to other services. Similarly servers which communicate with other servers must parse incoming messages as loosely as possible, probably using simple string searches. So, if a server received one or other of the incomplete responses above, it is still likely to search through the text only looking for the word runid and the number that is associated with it. Also, if a server is created which makes some very interesting implementation of an algorithm available online, but does

not conform in any way to the AIML standard, it is likely that someone who wishes to use this server will write a *wrapper* (another server that relays messages to and from the first server) for this server, which does obey AIML.

We do not wish to make it appear as if it is pointless for service authors to create complete and accurate messages. If messages are generated correctly, then this will ease interoperation with other entities on the WWM, and also with upper layers. Nonetheless, it is far more important for the WWM community to have access to an algorithm, than it is for the information to be structured in a 100 percent correct fashion.

5.4 Simple Discovery Protocol

As well as the messages listed above, services may wish to make a number of other methods available. The example mentioned briefly above was of a neural network service which provided a method so that others could inquire about the internal structure of the neural network. Obviously this message is only appropriate for that particular service, so it could not be a part of the standard. There needs to be some way, however, that all other entities can become aware of what messages (requests really) are supported by a service.

At the entry level for the WWM we provide the following solution. Service authors publish on their web site what messages they are willing to accept. Other services which wish to use this service must be hard coded to interact with that particular service. This should not pose too much of a difficulty if one service is built to only interact with one other service. But it will immediately begin to cause problems if this is not the case. However, at the entry level for the WWM we see a situation where one service is made public, e.g. a service to interact with a robot. This immediately proves popular so several people create services to choose actions for this robot service. The methods this service exposes become well known and people just build services to co-operate with the robot server. This obviously hinders interoperability, and will cause possible incompatibilities between different versions of AIML used in different sub-communities. What we propose as a second solution to this is a simple entry-level discovery protocol, where all services can optionally provide an extra method called getProfile. This method will return an AIML document listing all the methods the service makes available, the parameters it expects, and the return values. There will however be no categorisation of services and no semantics. We feel this is sufficient for the entry level.

5.5 Simple Data Representation

The various entities exist only to exchange data. This data will represent states and actions in sub-symbolic AI, and as such is perhaps simply a vector of numbers. At the entry level of the WWM we do not control the format of this data in any shape or form. Data is exchanged between entities between data tags in AIML messages such as those shown below.

```
<aiml version="1.1">
 <response type="Success" runid="99">
  <param name="id" value="0001"/>
  <param name="alttext" value="State"/>
   <piggyback type="GetState">
 <data name="x">
  (0,0,322,0,1556,0,0,0)
 </data>
   </piggyback>
 </response>
</aiml>
```

```
<aiml version="1.1">
 <request type="GetAction" runid="99">
  <data name="x">
   (0,0,322,0,1556,0,0,0)
  </data>
 </request>
</aiml>
```

The first of these messages gets the state from one service, and the second passes this state to another service, requesting that it decide upon an action. For this to work, the two services must be configured to interpret the same types of data in the same way, without any help from the protocol. At the entry level, we deem this to be the correct approach. It would be impossible to provide simple markup for all possible types of data, so we provide none. Once again, we foresee situations where services would be deliberately built to be compatible with the data format of other services.

6 The WWM higher levels

The following is essentially the Future Work section. This section will analyse the requirements for layer 2 and 3 of the WWM. While we have provided a simple architecture which will allow anyone with any amount of programming experience create a WWM service, we still feel that the complexity we have discarded at the lowest level should still be available to authors should they wish to use it. Integration with the upper layers should not pose great difficulties for WWM Layer 1 authors. The eight protocol mesages remain the same. All that changes is the fomat of the data provided, and the profile description mechanism.

6.1 Layer 2

At layer 2 we introduce XML and XML Schema to allow for more structured representation of data. A layer 2 service could describe itself using a WSDL description. This WSDL description would be delivered to any other entity using the getProfile message integrated into the AIML protocol, as discussed above. Any other service could then examine the description of the service in WSDL and decide upon what messages it wished to send to it. WSDL still provides a great deal more than what is required at layer 2. For instance it specifies what transport method should be used to contact a service. In the WWM the transport protocol will always be HTTP. A simpler way to describe a service is to simply create an XML Schema which describes all the methods that could be provided by a certain type of service. All services which fall into a particular classification could use this shared schema to describe the methods they expose. If all layer 2 entities within a particular sub-community used this shared schema then they could be certain

that they shared the same interpretation of the service methods. Similarly, an XML schema could be used to describe the data structure used within a certain sub-community. Data would not have to be configured to match the data from all other services, as is the case at the lower layers, but could instead be marked up to represent the type of data used within the sub-community.

6.2 Layer 3

At the upper layer we intend to integrate the WWM with the Semantic Web. Technologies such as DAML, RDF and DAML-S could be used to describe both data and profiles with proper semantics. DAML-S provides a way to describe web services. It proposes an ontology for describing a web service's profile and its model. We propose an upper ontology for describing services in a manner more sophisticated than the XML Schema at layer 2. HTML^{A} [4] and SHOE [6] provide simple semantic markup for data according to a given ontology. A number of tools may assist in the creation of ontologies, these include Protégé-2000 [protege.stanford.edu] and OilEd [img.cs.man.ac.uk/oil]. Candidates for the first ontologies to be defined on the WWM would include a *neural network* ontology and a *reinforcement learning* ontology.

6.3 Alignment

Since we are creating a completely open system where we expect independently developed services to interact we are faced with the problem of shared understanding. At layer 2 we first attempt to face this issue by providing for XML schema that express a shared markup for description and data. Layer 3 goes further by integrating Semantic Web technologies. However, we will still have to deal with situations where data or profiles are described using different ontologies or schema. In situations like there we propose that an attempt be made to align the understanding of certain concepts. Efforts in this area include Prompt [13] and Chimaera [11], both of which rely on human intervention.

7 Conclusion

We have described a scheme for getting sub-symbolic AI researchers to put their programs online as services that can be used remotely by others. This project is in progress. [For the latest information see the portal site w2mind.org.] Sub-symbolic Mind servers and World servers by multiple authors with layer 1 functionality are currently available online. Software clients are available to "run" a mind repeatedly in a world. Future work will involve the implementation of layers 2 and 3.

In designing this system we analysed and rejected a number of available technologies, including Java, CORBA and SOAP. We also rejected semantic web technologies for our *entry level* due to their complexity. We provide a simple entry point for anyone who wishes to make a service available on the WWM. We favour the "Web" model, where all interested parties can become involved, albeit at different levels. While the entry level

rejects advanced protocols due to their complexity, upper layers allow for their inclusion.

Most technology for programmers in Web Services and the Semantic Web is aimed at specialist network programmers. We have argued the need to design an entry level for non-specialist programmers in the problem domain we considered here. The discussion of design decisions here may have implications for other areas of the Semantic Web where the target audience may be programmers but not network programmers.

References

1. Ankolekar, A. et al. (2001), DAML-S: Semantic Markup for Web Services, *Semantic Web Workshop (SWWS)*, Stanford, 2001.
2. Bryson, J. (2000), Cross-Paradigm Analysis of Autonomous Agent Architecture, *JETAI* 12(2):165-89.
3. Bryson, J.; Lowe, W. and Stein, L.A. (2000), Hypothesis Testing for Complex Agents, *NIST Workshop on Performance Metrics for Intelligent Systems*.
4. Decker, S.; Erdmann, M.; Fensel, D. and Studer, R. (1999), Ontobroker: Ontology based Access to Distributed and Semi-Structured Information, *Semantic Issues in Multimedia Systems*, Kluwer.
5. Guillot, A. and Meyer, J.-A. (2000), From SAB94 to SAB2000: What's New, Animat?, *Proc. 6th Int. Conf. on Simulation of Adaptive Behavior (SAB-00)*.
6. Heflin, J.; Hendler, J. and Luke, S. (1998), Reading Between the Lines: Using SHOE to Discover Implicit Knowledge from the Web, *AAAI-98 Workshop on AI and Information Integration*, 1998.
7. Hendler, J. (2001), Agents and the Semantic Web, *IEEE Intelligent Systems Journal*, March/April 2001.
8. Humphrys, M. (2001), *The World-Wide-Mind: Draft Proposal*, Dublin City University, School of Computer Applications, Technical Report CA-0301, February 2001. computing.dcu.ie/~humphrys/WWM
9. Humphrys, M. (2001), Distributing a Mind on the Internet: The World-Wide-Mind, *Proc. 6th European Conf. on Artificial Life (ECAL-01)*, September 2001.
10. Martin, F.J.; Plaza, E. and Rodriguez-Aguilar, J.A. (2000), An Infrastructure for Agent-Based Systems: an Interagent Approach, *Int. Journal of Intelligent Systems* 15(3):217-240.
11. McGuinness, D.L.; Fikes, R.; Rice, J. and Wilder, S. (2000), An Environment for Merging and Testing Large Ontologies, *Proc. 7th Int. Conf. on Knowledge Representation and Reasoning (KR-2000)*, Morgan Kaufmann.
12. Nelson, M. (2002), XML Schema Validation in the Microsoft Universe, *Windows Developers Journal*, January 2002.
13. Noy, N. and Musen, M. (2000), PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment, *Proc. 17th National Conf. on Artificial Intelligence (AAAI-2000)*.
14. O'Connor, D.; Humphrys, M. and Walshe, R. (2001), First Implementation of the World-Wide Mind. w2mind.org
15. St. Laurent, S. (2000), Common XML - Final Draft Specification. groups.yahoo.com/group/sml-dev/files/Work/cxmlspec.txt
16. Udell, J. (1999), Exploring XML-RPC, Byte Magazine, August 1999.
17. Wilson, J. (ongoing), MinML: a minimal XML parser. wilson.co.uk/xml/minml.htm