

Using Meta-Data to Automatically Wrap BioInformatics Sources *

David Buttler
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
buttler@cc.gatech.edu

Terence Critchlow
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94551
critchlow1@llnl.gov

1 Introduction

A large amount of bioinformatics data is distributed over the Internet. Typically, this information is accessible only through custom, web-based query interfaces. These interfaces often include features uncommon in industrial applications such as multiple parameters, a variety of query options that invoke different support programs, indirection and delay pages, and complex results. If users require information from multiple sources, they must pose the appropriate queries at each source individually then explicitly integrate the results. This solution may be acceptable for a small number of sources, but it quickly becomes an overwhelming burden for users as the number of sources grows. Currently there are over 500 bioinformatics sources for biologists to choose from [2], making it infeasible to manually gather data from a non-trivial fraction of the available sources.

Our goal is to simplify access to bioinformatics data by providing a single access point to a large number of sources. There have been many similar efforts, but they have not succeeded due to the technical difficulty of integrating such a large number of complex sources. The fundamental problem is that each source is fiercely independent. As a result, they use different semantics, customized interfaces, and unique data formats. Furthermore, they are prone to having their interfaces and formats frequently updated without warning. In order to maintain access to a source, a specialized access program (wrapper) that keeps pace with the source's evolution is required.

This paper presents a general meta-data format capable of describing the complex, web-based interfaces often found in bioinformatics. Because this description provides sufficient information to automatically generate a wrapper for the associated site, it is an important first step in developing an infrastructure capable of interacting with a large number of dynamic, heterogeneous data sources. The next section provides a quick overview of the DARPA Agent Markup Language (DAML) [1], which formed the starting point for our work. We then present the extensions we made to describe these complex interfaces in Section 3, and conclude with an outline of how we intend to use this format in Section 4.

*This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

2 DAML as a Description Language for Web Services

Rather than create a new meta-data format from scratch, we started with the DAML specification. DAML is an extension of XML [8] and RDF [7] that allows the definition of machine understandable ontologies. Specifically, we have extended DAML's web service description language, DAML-S [6]. At this time DAML-S is still under active development, with version 0.5 released on May 30, 2001.

A service in DAML-S is essentially a web-based interface to data. The DAML-S description of a service has three parts: a service presents a *profile*, which indicates what information the service requires (input) and what the service does (output); it is described by a service *model*, which indicates how each step of the service works; and, it supports a service *grounding* which indicates how to access the service. For the purposes of our data source description, a profile is the external interface to a web service, the model represents the components of the service, and the grounding is the detailed information about how to properly interact with the service interface. Since we are interested primarily in describing how a service works, we focus on the model and the grounding.

The DAML-S model describes how components of a service interact. The most important characteristic of this description is the Process, which has inputs, outputs, and effects (real world consequences for invocation). Processes can represent atomic actions (e.g. find relevant protein structure identifiers given a disease name), or control flows (e.g. looped, conditional, sequenced [ordered], and parallel [unordered] execution). For example, the Process of retrieving protein structure information for a particular disease could be defined as a sequence of two atomic processes: looking up the protein structure identifiers related to the disease, and retrieving structure information given the structure identifier.

3 Extensions to the DAML Specifications for Wrapper Generation

To complete the interface descriptions of data sources, we need to extend DAML-S to include a detailed specification for groundings (the current release does not define them). Our definition of a grounding contains several elements:

- (1) A pointer to the Process it grounds
- (2) The binding
- (3) A specification of how to convert Process inputs into parameters the binding can use
- (4) A parser
- (5) Links

The definition of (1) is obvious; (2) and (3) are described in Section 3.1; (4) is described in Section 3.2; and (5) is described in Section 3.3.

3.1 Binding

Our grounding specification uses *bindings* to invoke different communication protocols for a service (e.g. HTTP, SOAP, a Java class, etc.). Since every grounding is associated with a particular process, its inputs and outputs are known. However, each binding must provide a conversion method to map the Process inputs to a usable format. In this section, we briefly outline three bindings and their default conversion methods: SOAP, HTTP, and Java. For the SOAP and HTTP bindings we follow the WSDL [4] specifications as closely as possible. SOAP is straight-forward: since we assume that a Process's inputs and outputs exactly match those of a SOAP accessible service, the conversion method simply returns the original inputs.

Because the HTTP binding is not a native XML protocol, the Process inputs need to be mapped to corresponding HTTP form input types. Typically, this requires a three step transformation. First, the names need to be translated because HTML form names are often abbreviated and somewhat cryptic (e.g. the field names in a Google search may include "q", "hl", "lr", and "safe") and thus are unlikely to match Process parameter names. Second, objects described at the Process level must be mapped into a corresponding text representation, because forms only accept strings as input. While this mapping may be trivial, we apply a transformation method to ensure generality. Depending on the type of object, this transformation may be performed by a local executable, Java class, web service, or XSLT script. Third, there are often values that are passed to the server that are not explicitly set by a user, such as hidden values or selection boxes that have reasonable defaults. By incorporating these values into the HTTP binding, we avoid polluting the Process description with a large number of trivial inputs.

The Java binding lets us use an arbitrary Java class as a communication protocol. While this is an excellent opportunity for extensibility, it can easily defeat the purpose of a meta-data description by hiding how the service works. It should be used only for simple operations that do not require external communication or for instances where Java defines the communication protocol (e.g. RMI). For example, it is often desirable to pause the execution of a Process for a period of time, such as when waiting for remote results to be computed. An obvious way to represent this delay state is as a binding to a Java method that sleeps for a specified time (taken as input) before returning.

3.2 Parser

The output of a service can be viewed as a raw stream. While no action is required when the service and the Process outputs are the same, in general some data post-processing must be performed. Our parser description specifies how to invoke another service that takes the raw output and returns the appropriate objects in XML format. We list several types of common parsers in Table 1, along with how each type is referenced and what input it expects.

Parser Type	Referenced by	Input
XSLT	URL	XML
HTTP	URL	string
Local executable	location of the program in the local file system	string
Java	URL of jar file; class name	string

Table 1: Parser types

3.3 Links

While the current description of DAML-S allows Processes to be defined in a sequence, it does not provide a mechanism for describing where the inputs of a Process are obtained. We use links to define the connections between outputs of one Process and inputs of another, allowing us to logically connect Processes. Links are also used to provide Process level default and constant values similar to those used in the HTTP binding, but available to all bindings for a Process.

A simple example of a web service that uses links in its description is the NIH BLAST search [3]. The communication protocol between a wrapper and the search service is shown in Figure 1. A state machine that implements this protocol is shown in Figure 2. This service can be described with three

Processes, corresponding to the states of the state machine: the initial query (A), a wait process that pauses while the results are computed (B), and a result retrieval process (C). The initial query returns a delay time and a result identification (RID) string. The wait process immediately following the query process uses only the delay time, while the retrieval process requires the RID. Note that after the wait process completes, the results may still not be available. If this happens, the service returns a delay page rather than the actual results when queried by the result retrieval process.

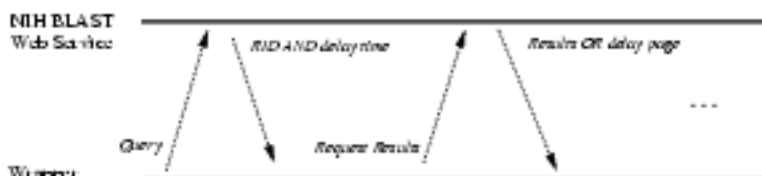


Figure 1: Communication Protocol for NIH BLAST Search

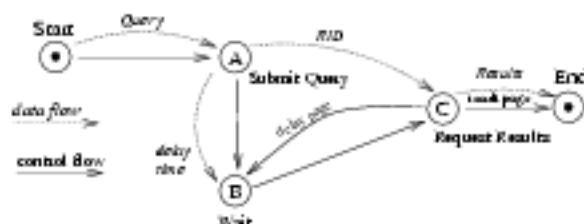


Figure 2: Control Flow for NIH BLAST Query

4 Conclusion and Future Work

This paper has presented a meta-data format based on DAML-S that allows us to describe complex interfaces to Web accessible data sources. We hope that this work will speed the evolution of the DAML-S standard and help establish a common format for complex interface descriptions. However, our primary motivation for this work has been to make the first steps towards an infrastructure that can support access to a large number of dynamic, heterogeneous scientific data sources. As such, we are continuing to develop this infrastructure by extending the XWrap Elite [5] wrapper generator to use our meta-data descriptions, instead of human interaction, to generate a wrapper for the associated source. Once completed this program will generate full featured wrappers that conform to a common query interface and return objects in XML. Providing this uniform source interface will enable different integration scenarios, from simple multi-source result fusion to techniques that reconcile semantic information such as data warehouses or federated systems, to be explored. Thus, it is an important first step in providing a single access point for bioinformatics researchers.

References

- [1] DAML+OIL (December 2000). <http://www.daml.org/2000/12/daml+oil-index>, December 2000.

- [2] DBCAT, The Public Catalog of Databases. <http://www.infobiogen.fr/services/dbcat/>, August 2001.
- [3] QBlast's URL API. User's Guide. <http://www.ncbi.nlm.nih.gov/BLAST/Doc/urlapi.html>, June 2001.
- [4] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.1. Technical report, W3C, March 2001.
- [5] W. Han, D. Buttler, and C. Pu. Wrapping Web Data into XML. *SIGMOD Record*, June 2001.
- [6] D. Martin. DAML-S: Semantic Markup for Web Services. <http://www.daml.org/services/daml-s/2001/05/daml-s.html>, June 2001.
- [7] Resource description framework (RDF) model and syntax specification. Technical report, W3C, February 1999.
- [8] Extensible markup language (XML) 1.0. Technical report, W3C, 1998.