BMC Bioinformatics

Research article

A web services choreography scenario for interoperating bioinformatics applications

Remko de Knikker[1], Youjun Guo[1], Jin-long Li[1], Albert KH Kwan[2], Kevin Y Yip[2], David W Cheung[2] and Kei-Hoi Cheung*[1,3]

Address: [1]Center for Medical Informatics, Department of Anesthesiology, Yale University School of Medicine, PO Box 208009, New Haven, CT 06520, USA, [2]Department of Computer Science and Information Systems, University of Hong Kong, Pokfulam Road, Hong Kong and [3]Department of Genetics, Yale University School of Medicine, PO Box 208005, New Haven, CT 06520, USA

Email: Remko de Knikker - remkodeknikker@hotmail.com; Youjun Guo - youjun.guo@yale.edu; Jin-long Li - jin-long.li@yale.edu; Albert KH Kwan - akhkwan@cecid.hku.hk; Kevin Y Yip - ylyip@csis.hku.hk; David W Cheung - dcheung@csis.hku.hk; Kei-Hoi Cheung* - kei.cheung@yale.edu

* Corresponding author

Published: 10 March 2004

BMC Bioinformatics 2004, **5**:25

Received: 03 November 2003
Accepted: 10 March 2004

This article is available from: http://www.biomedcentral.com/1471-2105/5/25

Abstract

Background: Very often genome-wide data analysis requires the interoperation of multiple databases and analytic tools. A large number of genome databases and bioinformatics applications are available through the web, but it is difficult to automate interoperation because: 1) the platforms on which the applications run are heterogeneous, 2) their web interface is not machine-friendly, 3) they use a non-standard format for data input and output, 4) they do not exploit standards to define application interface and message exchange, and 5) existing protocols for remote messaging are often not firewall-friendly. To overcome these issues, web services have emerged as a standard XML-based model for message exchange between heterogeneous applications. Web services engines have been developed to manage the configuration and execution of a web services workflow.

Results: To demonstrate the benefit of using web services over traditional web interfaces, we compare the two implementations of HAPI, a gene expression analysis utility developed by the University of California San Diego (UCSD) that allows visual characterization of groups or clusters of genes based on the biomedical literature. This utility takes a set of microarray spot IDs as input and outputs a hierarchy of MeSH Keywords that correlates to the input and is grouped by Medical Subject Heading (MeSH) category. While the HTML output is easy for humans to visualize, it is difficult for computer applications to interpret semantically. To facilitate the capability of machine processing, we have created a workflow of three web services that replicates the HAPI functionality. These web services use document-style messages, which means that messages are encoded in an XML-based format. We compared three approaches to the implementation of an XML-based workflow: a hard coded Java application, Collaxa BPEL Server and Taverna Workbench. The Java program functions as a web services engine and interoperates with these web services using a web services choreography language (BPEL4WS).

Conclusion: While it is relatively straightforward to implement and publish web services, the use of web services choreography engines is still in its infancy. However, industry-wide support and push for web services standards is quickly increasing the chance of success in using web services to unify heterogeneous bioinformatics applications. Due to the immaturity of currently available web services engines, it is still most practical to implement a simple, ad-hoc XML-based workflow by hard coding the workflow as a Java application. For advanced web service users the Collaxa BPEL engine facilitates a configuration and management environment that can fully handle XML-based workflow.

Background

The nature of genome-wide data analysis often requires the use of multiple databases and programs in some coordinated fashion. For example, microarray data analysis typically involves a sequence of analysis steps, which may include filtering, normalization, cluster analysis, access to a variety of genome annotation sources, etc. To compound the problem of interoperation, there are different ways of analyzing the same dataset. For example, some microarray analyses such as clustering are exploratory in nature while others are more specific to nature of the biological problem at hand (e.g., genetic network modeling [1]). Also, there are different ways to perform the same type of analysis (e.g., there are different clustering approaches such as hierarchical methods vs. non-hierarchical methods).

Despite the fact that a large number of genome databases and bioinformatics applications have been made available through the web, it is difficult to automate interoperation because:

1) the platforms on which applications reside are heterogeneous,

2) their web interface is human-friendly but not machine-friendly,

3) they use a non-standard format for input and output data,

4) they do not make use of standards to define the application interface and message exchange,

5) existing protocols for remote messaging are often not firewall-friendly.

Web services now offer a single uniform method for application integration through the Internet. They provide a model for accessing software systems over the web by pointing to their web address (URI), while their public interfaces and bindings are defined and described using an XML standard format. The potential of web services in bioinformatics database/tool unification has been recognized [2]. Examples of bioinformatics web service projects include: **BioMoby** is a project involving biological data hosts, biological data service providers, and coders whose aim is to explore various methodologies for biological data representation, distribution, and discovery [3]; **myGrid** aims to design, develop and demonstrate higher level functionalities over an existing Grid infrastructure that support scientists in making use of complex distributed resources [4,5]; **DDBJ** has published a number of biological tools as web services like Blast, ClustalW, DDBJ, Fasta, and others [6]; **Soaplab** developed at EBI provides a web service interface to a variety of analysis applications [7]; **Soap-HT-BLAST** is a BLAST web service implementation by the Bio Informatics Institute (BII) of the Agency for Science, Technology and Research in Singapore (A*STAR) [8]; and the **IBM Life Sciences** project implemented web services for PubMed, GenBank, BLAST, Phylogenic Tree and ClustalW [9].

Overview of the web service technology

Three standards have emerged that comprise the web services model. The Basic Profile of the Web Services Interoperability model (WS-I) [10] describes the web services model as follows:

1) the Web Service Description Language (WSDL) [11] is an XML language that describes a web service in an abstract manner by defining the web service interface and the exchange of messages between the provider and requester.

2) the Simple Object Access Protocol (SOAP) [12] is an XML-based protocol for stateless message exchange. It is not bound to any transport protocol, but is in general built on top of HTTP, which makes it firewall friendly in contrast to protocols used by CORBA for instance.

3) Universal Description, Discovery and Integration (UDDI) [13] is a standard protocol designed to publish details about an organization and the web services. It provides a description and definition of web services in a central repository, which functions as yellow pages for web services.

WSDL and SOAP are W3C standards, while UDDI is an OASIS standard. For a client to use a web service it only needs WSDL with SOAP being commonly used as the default protocol. SOAP implementations like Axis by the Apache Software Foundation, The Mind Electric's (TME) GLUE or Systinet's WASP Server for Java provide an easy-to-use method for publication and management of web services. These tools in general manage message exchange, register web services, generate WSDL files and implement functions like automatic error handling.

Web Services Choreography

While interlinking between web services is relatively simple, in reality applications need to work together in order to offer more advanced functionality. Thus to make full use of the advantages that the web services model offers, like advanced automation and application integration to group a set of applications together, we need more than a simple point-to-point connection. That is, we need a language that allows us to create a more complex composition of interdependent web services that easily operate together. This concept of interrelated web services that are linked together in a functionally coherent and repeatable process is called "Web Services Choreography".

There are more than a dozen languages that coordinate messaging and transaction for web services available but currently there are mainly two dominant open standards:

1) the Business Process Execution Language for Web Services (**BPEL4WS**) [14] was co-written by IBM, MicroSoft and BEA, and is (royalty-free) submitted as a standard to OASIS. SUN Microsystems and Oracle recently joined the WS-BPEL Technical Committee [15],

2) the Business Process Markup Language (**BPML**) [16] is defined by BPML.org and extends the Web Services Choreography Interface (WSCI) that was developed by SUN.

The W3C published in August 2003 a draft of its specification for Web Services Choreography language (WS-CHOR) [17]. **BPEL4WS** and **BPML** are the leading emerging standards. They are compatible languages and both offer a rich set of workflow options. **BPEL4WS** seems most promising at this moment to emerge as a widely supported standard for web services choreography since it has the broad industry support from companies like IBM, MicroSoft and BEA, while SUN recently also joined the Technical Committee. In addition, **BPEL4WS** is well documented and tools like editors and choreography engines are already available. Both **BPEL4WS** and **BPML** allow complex scenarios of choreography like concurrent processes, synchronous and asynchronous messaging, roll back mechanisms, data manipulation and error handling. These features benefit certain classes of biomedical informatics applications that involve sensitive data and the use of parallel programming to speed the complex analyses of large datasets.

Choreography engines

A choreography engine is the "smart center" of a web services choreography, which executes and manages workflow, process automation, data mapping, error handling and security. These engines significantly improve the management of the web services workflow. Currently there are a few web services choreography engines available. IBM offers its Business Process Execution Language for Web Services Java Run Time (BPWS4J), which implements a choreography as an RPC-style web service (for Document vs. RPC style see next section). Another project of interest is myGRID [3], which developed Taverna, a tool that integrates Choreography into a graphical workbench, for creating, editing and browsing workflow. Taverna uses a simple non-standard choreography language, called **XScufl**, and is an ongoing collaborative project between EBI, IT Innovation and the Human Genome Mapping Project (HGMP) of the Medical Resource Center (MRC). The **Collaxa BPEL Server** [18] implements the choreography as a document-oriented web service, is easy-to-manage with an advanced graphical interface (which works for IE6 only) but is available under a commercial license. The Mind Electric (TME) is to release **Gaia**, and other engines are being developed at the moment. Most of these engines support either **BPML** or **BPEL4WS**.

Document vs. Remote Procedure Call (RPC) styles

There are two ways of exchanging data between web services: document-oriented vs. remote procedure call (RPC). The document-style exchanges data as XML documents, while the RPC-style web service describes the interface in the format of a method-signature and takes input and output in a programming language specific data type. A data type-mapping interface (called a proxy in dot Net or a Serializer in Apache Axis) is used to converse data between a client and a web service.

While some vendors (e.g., Microsoft) adopt the document-oriented method, others (e.g., IBM, Apache Axis and SUN) use the RPC-oriented method. In practice, a mixture of document-style and RPC-style web services will exist. While it might be easier to wrap an RPC-style web service interface around existing applications, newly developed applications could be more inclined to use document-style web services. Currently however, most tools that are available initially have focused on RPC-style web services.

Results and discussion

In our web services choreography scenario [19], a user obtained a set of microarray spot IDs (GenBank accession numbers, Affymetrix chip probe set IDs, and Unigene Cluster IDs) corresponding to the genes (both known and unknown) being studied. The user then wants to create a hierarchy of Unified Medical Language System (UMLS) concepts formatted in XML and group the result per MeSH root category.

Three web services have been created to execute this scenario: one that maps a set of microarray spot IDs to their corresponding GenBank IDs, and if available their corresponding PubMed IDs and UMLS concept IDs (GetHAPI); the second web service builds an XML formatted hierarchy of UMLS concepts for a set of UMLS concept IDs (GetUMLS), while a third web service generates an XML-based report of UMLS concept ID occurrences per MeSH root category from a hierarchy of UMLS concept IDs (GetUMLSReport). A similar functionality as the choreography of GetHAPI, GetUMLS and GetUMLSReport is provided by the High-density Array Pattern Interpreter (HAPI) [20,21] developed at the University of California, San Diego through a web-based (HTML) interface.

When using the traditional web interface of HAPI [21], the user has to engage in a three-step process. In step 1, the user provides a name for the result set and uploads a file containing a list of spot IDs.

Figure 1 shows step 2 of HAPI in which the user needs to select a column that contains the GenBank IDs and can enter a name for the HTML summary page. In step 3 the result file is being generated and the user has to click the link to view the HTML summary page.

The result, shown in figure 2, is an HTML table containing a hierarchy of keywords from literature associated with the genes that were provided. The HTML table displays the occurrences per category and the split up for each category of the keywords hierarchy. To retrieve information from GenCard, Entrez or PubMed the user can click links to these databases per UMLS keyword.

While the traditional web approach provides the user with an easy-to-use interface and easy-to-read (HTML) output, it lacks a flexible programmatic interface, making automatic tool linking and interoperation difficult. The WSDL files of the choreography and of the web services used in the choreography on the other hand describe the interfaces and message format that provide the equivalent functionality of the HAPI application. This allows the client program to access and interpret the output data programmatically. In addition the user can choose to access additional web services in his choreography, like XEMBL [22] to retrieve GenBank information for genes of interest, based upon the outcome of the GetUMLSReport or choose to access only part of the web services choreography that comprises the HAPI functionality, for instance in order to retrieve the complete hierarchy of UMLS concepts associated with the submitted genes or to retrieve only the PubMed IDs for the associated genes. The result is that the remote functionality of the HAPI utility can easily be integrated in a local workflow, data can be programmatically processed and the individual modules can be utilized more dynamically and flexibly.

Hard coding the implementation of a BPEL workflow through a Java application allows for the easy implementation of a workflow and the full use of the power of the Java programming language with the ease of altering the workflow by simply configuring the BPEL file. Extending the BPEL workflow can be achieved by adding the *invoke* tag of the new web service to the BPEL sequence as long as the workflow consists of synchronous web services and a straightforward sequence of web services. Mapping of data can be achieved by XSL transformations implemented in Java. While it is fairly simple to implement a Java application and execute the BPEL Workflow, implementing the XML-based scenario in either Collaxa or Taverna is more arduous.

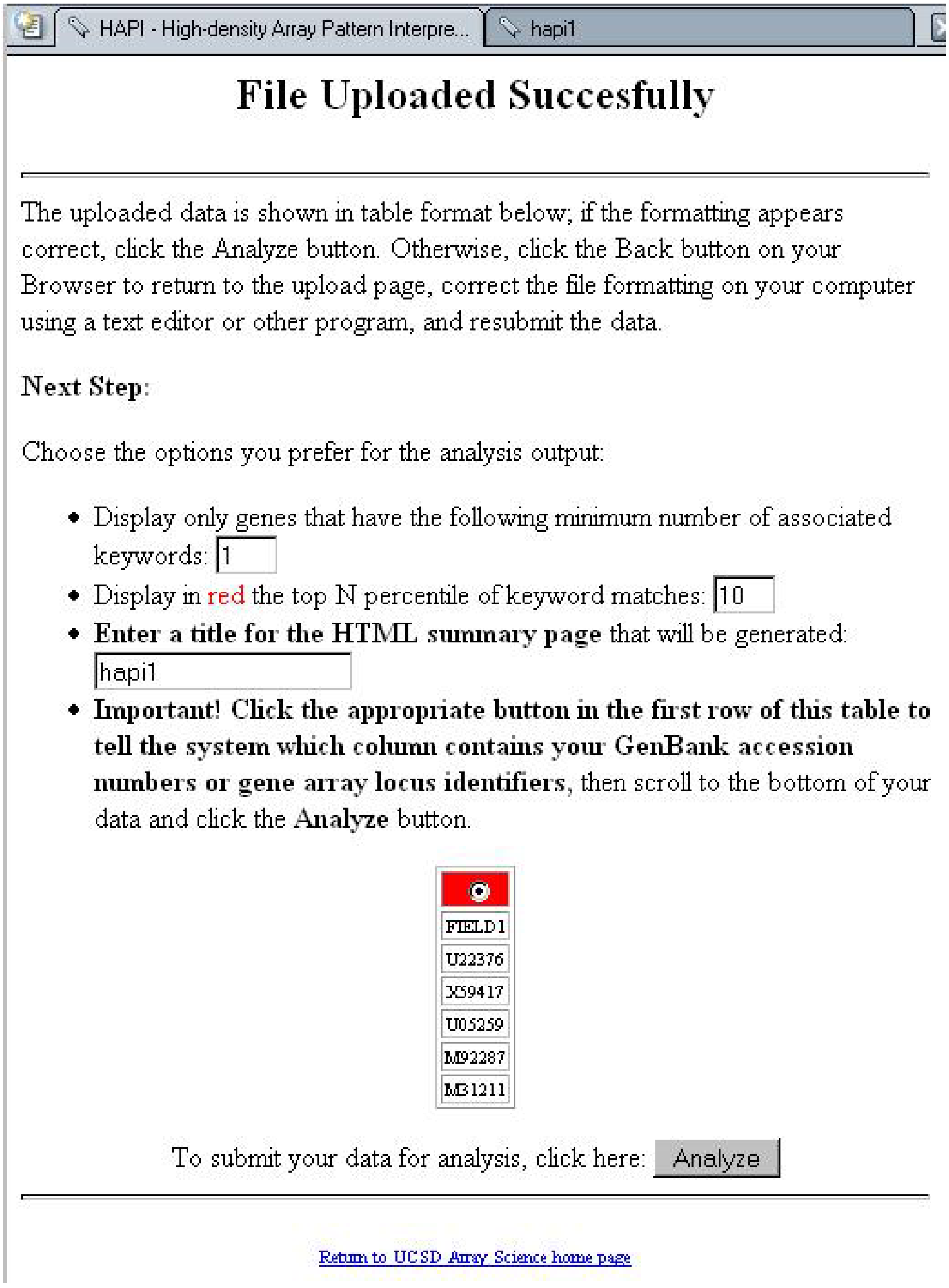
Taverna provides a user-friendly interface and RPC-style web services can quickly and easily be implemented in Taverna without advanced knowledge of web service technologies. It simply involves adding the URL location of the WSDL to the available services and adding the chosen operation or method of the web service to the model. By creating variables and data links the model is completed. It is very easy to quickly add services to the model, and reroute the workflow by altering the data links. To execute the workflow, switch to the Workflow Input Panel and press the run-button while the result can be viewed in the result tab (Figure 3). But as Taverna is an ongoing, open source effort not all Document-style WSDL implementations appeared to be supported at this moment, while mapping output to input options are still limited, restricting the applicability at this moment.

Collaxa is committed to Document-style web services and it has created a BPEL designer called **bpelz** and a browser-based (IE6 only) workflow management console that allows instance inspection amongst others (Figure 4). The designer's interface may require still some hand coding of the source files to make corrections, while most WSDL files will need to be extended by a partnerLinkType definition. But Collaxa offers full support of BPEL with advanced support for extensions to the model. Data sources can be mapped using XPath and BPEL extension for Java coding is supported.

Conclusions

Web services facilitate bioinformatics software interoperation based on a standard XML-based protocol. While the development of web service technologies is as much ongoing as rapid. Many bioinformatics applications have been published as web services and more initiatives are being developed. With more web services for bioinformatics becoming available, there will be an increasing need to use metadata and ontologies to facilitate description, tracking, classification, and constraining of web services based upon UDDI or domain-specific standards as developed by BioMoby.

While it is relatively straightforward to implement and publish web services, the interoperation of bioinformatics web services through the use of choreography engines is in the early stage. Documentation and examples for most choreography engines are scarce. For example, IBM's choreography engine BPWS4J can only be deployed as an RPC style web service. MyGrid's Taverna offers an advanced GUI and can process most WSDL files, but support for document-oriented web services is still limited. Collaxa's orchestration engine offers full **BPEL4WS** support, fully



File Uploaded Succesfully

The uploaded data is shown in table format below; if the formatting appears correct, click the Analyze button. Otherwise, click the Back button on your Browser to return to the upload page, correct the file formatting on your computer using a text editor or other program, and resubmit the data.

Next Step:

Choose the options you prefer for the analysis output:

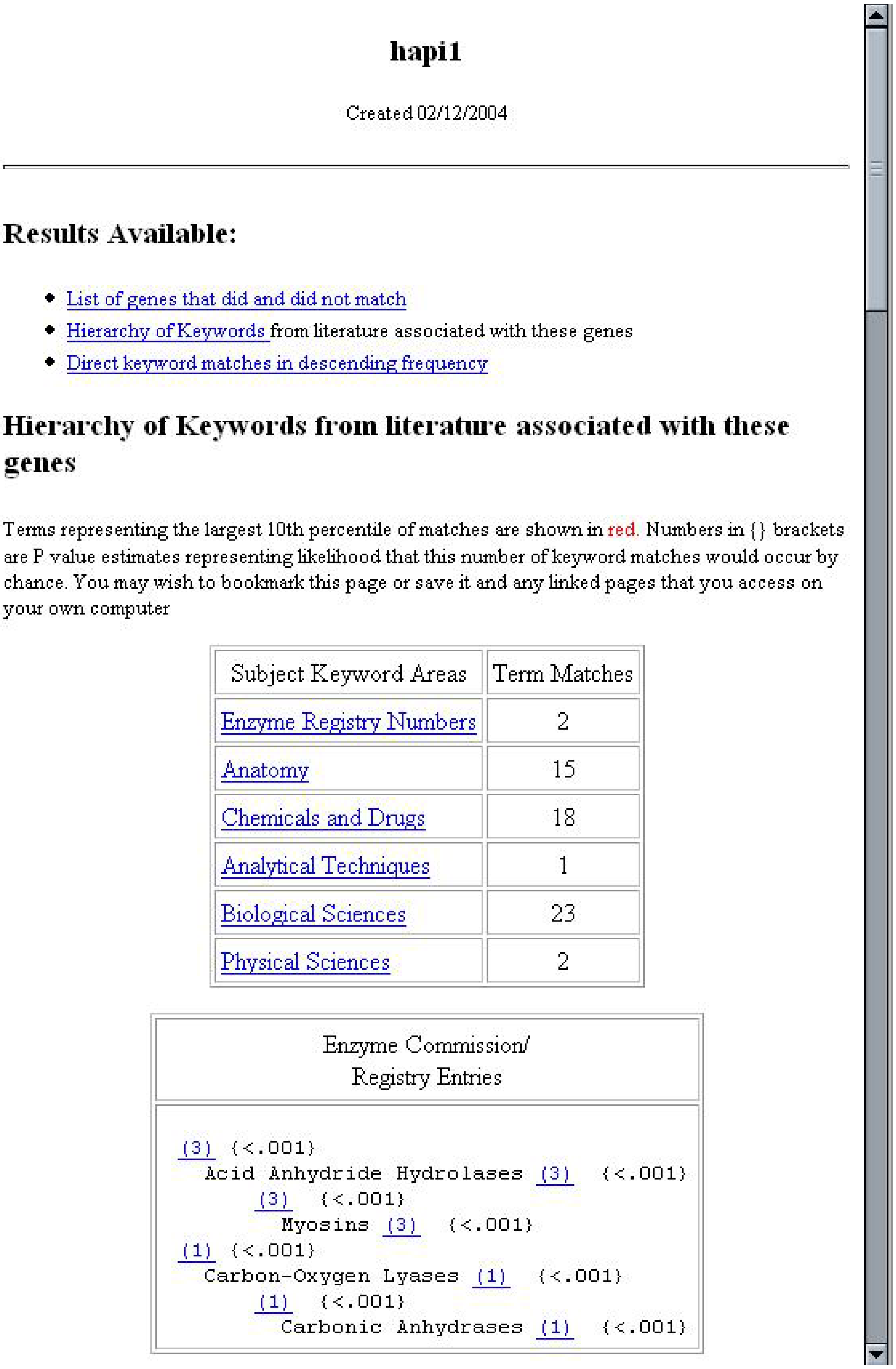
- Display only genes that have the following minimum number of associated keywords: 1
- Display in red the top N percentile of keyword matches: 10
- **Enter a title for the HTML summary page** that will be generated: hapi1
- **Important! Click the appropriate button in the first row of this table to tell the system which column contains your GenBank accession numbers or gene array locus identifiers**, then scroll to the bottom of your data and click the **Analyze** button.

●
FIELD1
U22376
X59417
U05259
M92287
M31211

To submit your data for analysis, click here: Analyze

Return to UCSD Array Science home page

Figure 1
HAPI, the High-density Array Pattern Interpreter. Step 2 of 3 when using HAPI utility's web-based (HTML) interface provided by UCSD



hapi1

Created 02/12/2004

Results Available:

- List of genes that did and did not match
- Hierarchy of Keywords from literature associated with these genes
- Direct keyword matches in descending frequency

Hierarchy of Keywords from literature associated with these genes

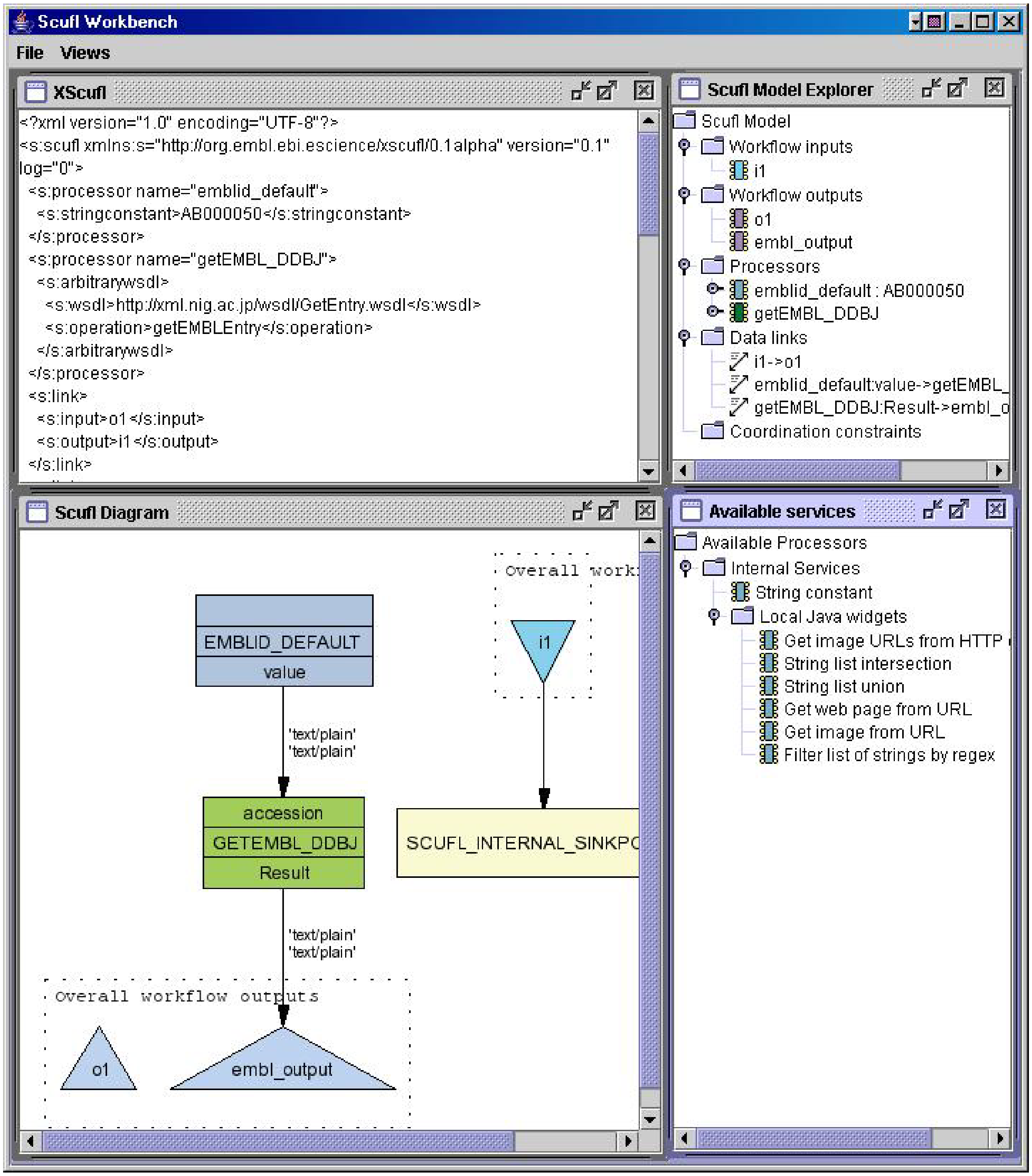
Terms representing the largest 10th percentile of matches are shown in red. Numbers in {} brackets are P value estimates representing likelihood that this number of keyword matches would occur by chance. You may wish to bookmark this page or save it and any linked pages that you access on your own computer

Subject Keyword Areas	Term Matches
Enzyme Registry Numbers	2
Anatomy	15
Chemicals and Drugs	18
Analytical Techniques	1
Biological Sciences	23
Physical Sciences	2

Enzyme Commission/
Registry Entries

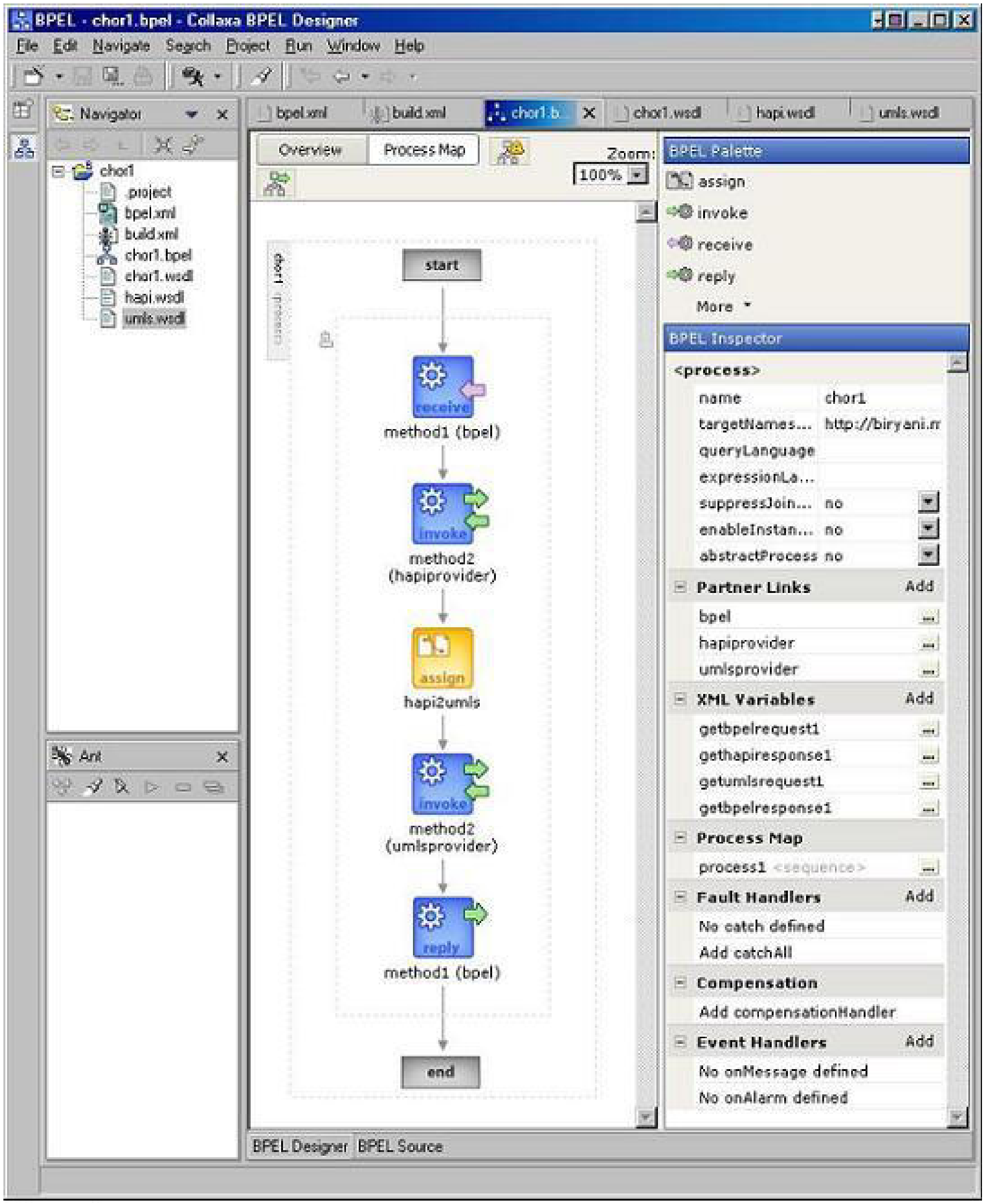
```
(3) {<.001}
  Acid Anhydride Hydrolases (3)  {<.001}
      (3)  {<.001}
        Myosins (3)  {<.001}
(1) {<.001}
  Carbon-Oxygen Lyases (1)  {<.001}
      (1)  {<.001}
        Carbonic Anhydrases (1)  {<.001}
```

Figure 2
HAPI, the High-density Array Pattern Interpreter. Summary page of HAPI utility provided by UCSD



```
<?xml version="1.0" encoding="UTF-8"?>
<s:scufl xmlns:s="http://org.embl.ebi.escience/xscufl/0.1alpha" version="0.1"
log="0">
 <s:processor name="emblid_default">
  <s:stringconstant>AB000050</s:stringconstant>
 </s:processor>
 <s:processor name="getEMBL_DDBJ">
  <s:arbitrarywsdl>
   <s:wsdl>http://xml.nig.ac.jp/wsdl/GetEntry.wsdl</s:wsdl>
   <s:operation>getEMBLEntry</s:operation>
  </s:arbitrarywsdl>
 </s:processor>
 <s:link>
  <s:input>o1</s:input>
  <s:output>i1</s:output>
 </s:link>
```

Figure 3
Taverna, the Scufl Workbench Screenshot of Taverna implementation of RPC based workflow retrieving EMBL information from DDBJ.



<process>

name	chor1
targetNames...	http://biryani.r
queryLanguage	
expressionLa...	
suppressJoin...	no
enableInstan...	no
abstractProcess	no

Partner Links Add
- bpel
- hapiprovider
- umlsprovider

XML Variables Add
- getbpelrequest1
- gethapiresponse1
- getumlsrequest1
- getbpelresponse1

Process Map
- process1 <sequence>

Fault Handlers Add
- No catch defined
- Add catchAll

Compensation
- Add compensationHandler

Event Handlers Add
- No onMessage defined
- No onAlarm defined

Figure 4
Collaxa BPEL Server Screenshot of Collaxa's bpelz design tool, building a choreography of GetHAPI and GetUMLS web services

supports document-oriented web services and provides an easy-to-use and advanced GUI, but it is only available under a commercial license and its console runs only under IE6. To sum up, there is a need for further development of user-friendly choreography engines that support choreography standards like **BPEL4WS** and both document-style and RPC-style choreographies, as well as broader manipulation of data. Of the two web service engines, only Collaxa seemed to support our example XML-based workflow, providing XPath and full BPEL support. Collaxa requires working knowledge of all web services technologies, whereas Taverna is transparent to use. For an RPC-based web service and some XMLbased workflow that does not require complex mapping, Taverna can be used. In addition, it offers advanced support for additional services, which we were not able to fully explore, and is more than Collaxa focused on the scientific community. While transparent engines fully supporting BPEL might be available soon, for most simple workflow implementations it is currently still most convenient and flexible to use hard coded implementations. For more advanced workflow Collaxa can be used while workflow requiring specific services or RPC based workflow Taverna is available.

Methods

Although it is possible to invoke a web service using any other protocol, we use WSDL and SOAP. Since we already know the location of the target web service and its functionality, publication and discovery with UDDI is not our focus here, although it is part of the WS-I Basic Profile. We refer to web services projects such as BioMoby that extensively use UDDI for more information.

In exchanging messages between web services, we use the document-oriented method since it has the following advantages.

1) It uses a language independent and self-describing data format.

2) It allows for validation based on DTD or XML Schema.

3) It provides a better separation of data and functional logic.

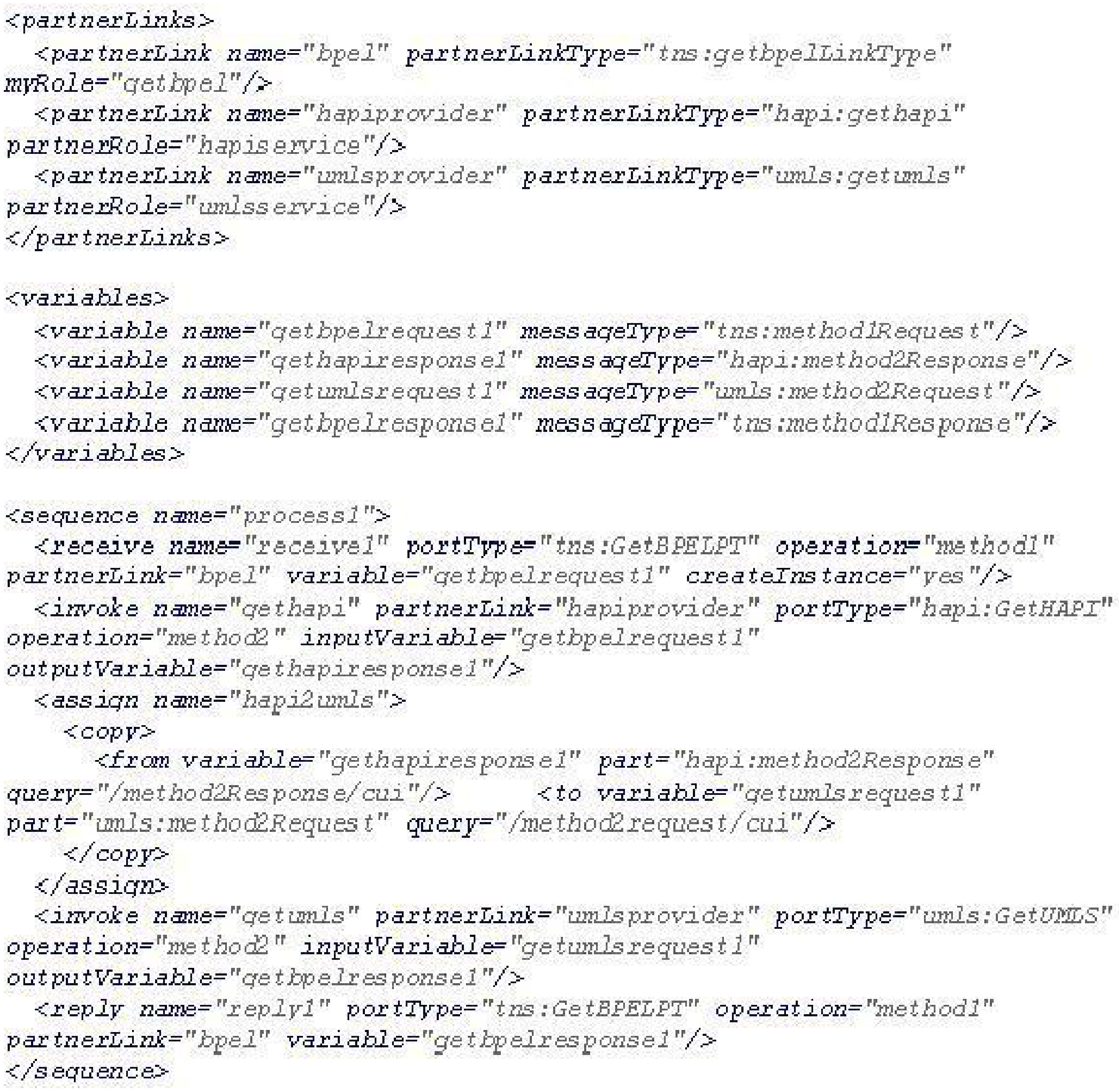
It allows us to exploit the various bioinformatics-related XML standards (including BSML[23], BIOML [24,25], MAGE-ML [26,27], etc) to further facilitate interoperability.

Our web services choreography scenario involves the use of **BPEL4WS** to define a workflow that consists of several document-oriented web services. To execute the choreography, we have created a client that functions as a choreography engine that is implemented as a document-oriented web service. For demonstration purposes the choreography engine can be accessed through a web interface that calls the client engine. The user configures a **BPEL** file describing the web services choreography and a WSDL file that describes the choreography engine's web service interface and message format. The client executes the preferred workflow of web services as described in these two files. Figure 5 illustrates the central elements of the choreography code that integrates the GetHAPI, GetUMLS, and GetUMLSReport web services.

This web services choreography returns an XML document that represents a report of UMLS concept occurrences per category from the hierarchy of UMLS keywords that correlate to the submitted set of microarray spot IDs.

Authors' contributions

KC initiated and directed the research project. RdK performed the technical research and documentation, and the overall design and implementation of the project. YG provided XSLT Transformations and assisted in implementation. JL assisted in the data analysis and database optimization. AK, KY and DC collaborated with the YCMI team on the web services interoperation. All authors read and agreed with the final manuscript.



```
<partnerLinks>
  <partnerLink name="bpel" partnerLinkType="tns:getbpelLinkType"
myRole="getbpel"/>
  <partnerLink name="hapiprovider" partnerLinkType="hapi:gethapi"
partnerRole="hapiservice"/>
  <partnerLink name="umlsprovider" partnerLinkType="umls:getumls"
partnerRole="umlsservice"/>
</partnerLinks>

<variables>
  <variable name="getbpelrequest1" messageType="tns:method1Request"/>
  <variable name="gethapiresponse1" messageType="hapi:method2Response"/>
  <variable name="getumlsrequest1" messageType="umls:method2Request"/>
  <variable name="getbpelresponse1" messageType="tns:method1Response"/>
</variables>

<sequence name="process1">
  <receive name="receive1" portType="tns:GetBPELPT" operation="method1"
partnerLink="bpel" variable="getbpelrequest1" createInstance="yes"/>
  <invoke name="gethapi" partnerLink="hapiprovider" portType="hapi:GetHAPI"
operation="method2" inputVariable="getbpelrequest1"
outputVariable="gethapiresponse1"/>
  <assign name="hapi2umls">
    <copy>
      <from variable="gethapiresponse1" part="hapi:method2Response"
query="/method2Response/cui"/>      <to variable="getumlsrequest1"
part="umls:method2Request" query="/method2request/cui"/>
    </copy>
  </assign>
  <invoke name="getumls" partnerLink="umlsprovider" portType="umls:GetUMLS"
operation="method2" inputVariable="getumlsrequest1"
outputVariable="getbpelresponse1"/>
  <reply name="reply1" portType="tns:GetBPELPT" operation="method1"
partnerLink="bpel" variable="getbpelresponse1"/>
</sequence>
```

Figure 5
BPEL Choreography An extract from the BPEL file describing the choreography of GetHAPI and GetUMLS web services.

Acknowledgements
This research is supported in part by NIH grant K25 HG02378 and by NSF grant DBI-0135442. The authors are grateful to Dr. Dan Masys (at UCSD) for allowing us to download the current HAPI dataset. We would like to acknowledge Tom Oinn at the European Bioinformatics Institute and head of the Taverna project, for his feedback with regard to Taverna.

References
1. van Someren EP, Wessels LF, Backer E, Reinders MJ: **Genetic network modeling.** *Pharmacogenomics* 2002, **3(4):**507-25.
2. Stein L: **Creating a bioinformatics nation.** *Nature* 2002, **417:**119-20.
3. Wilkinson M, Links M: **BioMoby: An open source biological web services proposal.** *Brief Bioinform* 2002, **3(4):**331-341.
4. Stevens R, Robinson A, Goble C: **myGrid: personalized bioinformatics on the information grid.** *Bioinformatics* 2003, **19(Suppl 1):**I302-I304.
5. **myGrid BioServices** [http://www.mygrid.org.uk/myGrid/web/components/BioServices/]
6. **DNA DataBase of Japan (DDBJ) Web Services** [http://xml.ddbj.nig.ac.jp/wsdl/index.jsp]
7. **European Bioinformatics Institute (EBI) Soaplab** [http://industry.ebi.ac.uk/soaplab/]
8. Wang J, Mu Q: **Soap-HT-BLAST: high throughput BLAST based on Web services.** *Bioinformatics* 2003, **19(14):**1863-4.
9. **IBM alpha Works, Web Services for LifeSciences** [http://www.alphaworks.ibm.com/tech/ws4LS]
10. **Basic Profile of the Web Services Interoperability model** [http://www.ws-i.org/Profiles/Basic/2003-05/BasicProfile-1.0-WGAD.htm]
11. **Web Service Description Language (WSDL)** [http://www.w3.org/TR/2003/WD-wsdl12-20030611/]
12. **Simple Object Access Protocol(SOAP)** [http://www.w3.org/TR/SOAP/]
13. **Universal Description, Discovery and Integration (UDDI)** [http://www.uddi.org/specification.html]
14. **Business Process Execution Language for Web Services (BPEL4WS)** [http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/]
15. **OASIS Web Services Business Process Execution Language Technical Committee (WS-BPEL TC)** [http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel]
16. **Business Process Markup Language (BPML)** [http://www.bpmi.org/bpml.esp]
17. **Web Service Choreography Working Group** [http://www.w3.org/2002/ws/chor/]
18. **The Collaxa BPEL Server** [http://www.collaxa.com]
19. **Example URL** [http://biryani.med.yale.edu:8081/axis/index.jsp?which=6-4-3]
20. Masys D, Welsh J, Fink JL, Gribskov M, Klacansky I, Corbeil J: **Use of keyword hierarchies to interpret gene expression patterns.** *Bioinformatics* 2001, **17(4):**319-26.
21. **High-density Array Pattern Interpreter (HAPI)** [http://array.ucsd.edu/hapi]
22. **The XEMBL Project** [http://www.ebi.ac.uk/xembl/]
23. **Bioinformatic Sequence Markup Language (BSML)** [http://www.bsml.org/]
24. Fenyo D: **The Biopolymer Markup Language.** *Bioinformatics* 1999, **15(4):**339-40.
25. **Genomic Solutions, The BIOpolymer Markup Language (BIOML)** [http://65.219.84.5/BioML.html]
26. Spellman PT, Miller M, Stewart J, Troup C, Sarkans U, Chervitz S, Bernhardt D, Sherlock G, Ball C, Lepage M, Swiatek M, Marks WL, Goncalves J, Markel S, Iordan D, Shojatalab M, Pizarro A, White J, Hubley R, Deutsch E, Senger M, Aronow BJ, Robinson A, Bassett D, Stoeckert Jr CJ, Brazma A: **Design and implementation of microarray gene expression markup language (MAGE-ML).** *Genome Biol* 2002, **3(9):**RESEARCH0046. 2002 Aug 23
27. **MicroArray and Gene Expression – MAGE Group, MAGE-ML Working Group** [http://www.mged.org/Workgroups/MAGE/mage-ml.html]