AN XML VIEW OF THE "WORLD"

Ezio Bartocci and Emanuela Merelli
Dipartimento di Matematica e Informatica, Università di Camerino
via Madonna delle Carceri, I-62032 Camerino, Italia
Email: ezio.bartocci, emanuela.merelli@unicam.it

Leonardo Mariani
Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano Bicocca
via Bicocca degli Arcimboldi, 8, I-20126 Milano, Italia
Email: mariani@disco.unimib.it

Key words: Coupling and Integrating Heterogeneous Data Sources, Wrapper, XML.

Abstract: The paper presents "Any Input XML Output" (AIXO), a general and flexible software architecture for wrappers. The architecture has been designed to present data sources as collections of XML documents. The use of XSLT as extraction language permits extensive reuse of standards, tools and knowledge. A prototype developed in Java has been effectively proven in several case studies. The tool has also been successfully integrated as a wrapper service into BioAgent, a mobile agent middleware specialized for use in the molecular biology domain.

1 Introduction

The World Wide Web (WWW) was essentially developed as a vehicle for the exchange of documents and data between human users distributed among Internet hosts. Today Internet applications are being designed and developed to access Internet resources as documents, data and services. Since the WWW was originally conceived for human users, it is not machine readable (Berners-Lee et al., 2001), and thus unsuitable to support the issues raised by the present scenario, at least until its conversion into a Semantic WWW.

Generally, Internet applications are complex and designed to support users' activities in special application domains (ecommerce, medicine, tourism, ...). To support the coordinated execution of multiple distributed activities in an open scenario like Internet, suitable abstractions and tools to model and enable effective world wide access to Internet data and services are required. Typically, the execution tasks, associated to activities, include data research and data extraction from distributed heterogeneous data repositories.

In recent years, different technologies and standards have been developed to automate the research, integration and elaboration phases, but a uniform framework has yet to be developed. Notwithstanding standards definitions like XML (W3C, 2000a), XSL (W3C, 2001) and XSLT (W3C, 1999) for data description and portability, and UDDI (Bellwood et al., 2002) and WSDL (W3C, 2002b) for service access and discovery, machines still cannot automatically access Internet resources because of semantic and syntactic heterogeneity. The semantic gap remains an unsolved problem. Services or documents, which in theory are integrable, in practice cannot be fully processed because the same piece of information is stored using a different structure or different terms. We believe that the definition of shared ontologies (Gruber, 1993) would simplify the semantic integration of heterogeneous data sources. While the semi-structured nature of XML and the use of metalanguages such as DTD or XML-Schema might make documents machine-readable (W3C, 2001), the problem would still remains, since for the most part Internet resources continue to be non-XML. At present only web service developers provide XML documents as exchange format.

In the scenario presented here, wrapper technology plays an important role. A wrapper adds a layer of abstraction to a resources access system, allowing for previously impossible interactions. For example, wrappers to present HTML pages as XML documents can be constructed. The wrapper is not a stand-alone solution, but is often used with other tools such as multi-agent systems (Müller, 1999; Sudmann and Johansen, 2001; Kolp et al., 2001) and Web Services (Pierce et al., 2002; Kreger, 2001). Many wrappers have been proposed in the literature, but each is

specialized for a particular type of resource. In the context of heterogeneity we believe that a wrapper customized for a specific application domain helps to achieve an "XML view of the world". By customizable wrapper, we mean a software component that can adapt to the different data sources that characterize the application domain.

In this paper we propose AIXO, a wrapper architecture suitable for the presentation of any data source as a collection of XML documents. AIXO's flexibility and modularity allow for management of many input data sources ranging from HTML to XML, from databases to flat file, from CGI to command line programs. XML has been chosen as the output format to enhance portability and flexibility; XML permits a broad reuse of existing standards and technologies such as data models, parsers, and tools. AIXO's output can also be forced to comply with a well defined XML structure, and can provide the final user with a syntactic and semantic homogeneous "view of the world". The proposed AIXO architecture has been experimentally proven on different resources in different contexts and successfully integrated as *wrapper service* in BioAgent (Merelli et al., 2002) - a mobile agent-based middleware. We emphasize that the integration of AIXO in an agent-based middleware represents, in the context of the present paper, one of many contexts where a generalized wrapper could be proved. Therefore, we will not enter into details of MAS (Multi-Agent Systems).

The paper is organized as follow. Section 2 describes the background of the work. Section 3 introduces the AIXO architecture and Section 4 discusses experimental results. Section 5 describes integration of AIXO with BioAgent. Section 6 presents related works, and Section 7 outlines future research directions and presents conclusions.

2 Background of the Work

XML XML (W3C, 2000a) – the eXtensible Markup Language – has recently emerged as a new standard for data representation and exchange on the Internet. XML supports electronic exchange of machine-readable data on the Web. The basic ideas underlying XML are very simple: tags on data elements identify the meaning of the data, rather than specifying, for example, how data should be formatted. The relationships between data elements are provided through simple nesting and references. XML data files can be presented through specifications in XSL, the eXtensible Stylesheet Language (W3C, 2001). Any XML document can contain an optional description of its grammar for use by applications that require structural validation. The grammar portion is called XML-schema (W3C, 2001).

XML has been applied with success in several contexts such as data description and integration (IONA, 2002), communication protocols (W3C, 2000b) and provision of services (W3C, 2002a).

XSLT XSLT is part of the XSL language and defines the syntax and semantics of a language capable of transforming XML documents into other XML documents. An XSLT transformation defines the requirements for transformation of an input document to an output document.

Since XSLT is a well-known language, the use of AIXO would be relatively simple and easily learned. The wrapper architecture has been developed using XML standards, therefore we will take advantage of new standards and new XML tools, integrating them in the proposed architecture.

There are different XSLT processors compliant with XSLT standards, such as DGXT (http://www.datapower.com/products/), iXSLT (http://www.infoteria.com/), Xalan (http://xml.apache.org/xalan-j/) and XT (http://www.blnz.com/xt/). We have decided to use Xalan from Apache Software Foundation, because it was developed in Java as AIXO. Xalan is compliant with the lastest recommendations and provides good performance.

JDOM JDOM (http://www.jdom.org) is a Java implementation of Document Object Model (DOM) (W3C, 1998) a standard model defined to manage XML documents. It provides a robust, lightweight framework for reading and writing XML documents. JDOM interacts with the Simple API for XML (SAX) parser and the Document Object Model (DOM) parser, therefore allowing for the reception of XML elements from both of the above-mentioned parsers.

MAS An agent is an active autonomous entity which can communicate with other agents and human users. Agents often possess other capabilities such as benevolence and mobility. Agents are grouped together to form a Multi-Agent System (MAS) (R.Jennings et al., 1998). Complex tasks can be successfully executed by agents through the coordination of their actions. An autonomous agent must know the environment in which it operates or intends to operate. In an open Internet scenario, a mobile agent must be capable of accessing remote resources.

Previously (Merelli et al., 2002), we developed a multi-agent system populated with mobile agents. These agents move among hosts of a network and can access local resources and services. We will discuss the integration of AIXO in BioAgent in Section 5.

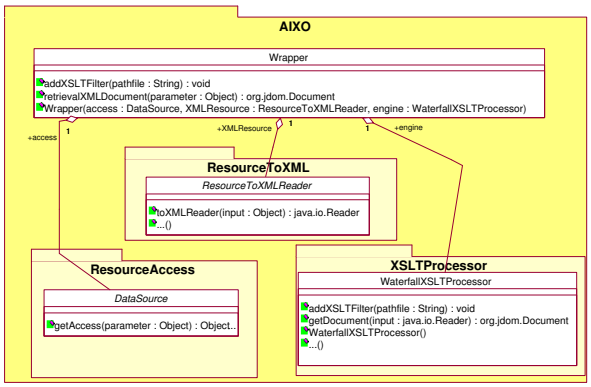

Figure 1: AIXO architecture

3 AIXO Architecture

The AIXO architecture is not for a specific resource or data type; rather, it is general and suitable for a wide range of resources. An AIXO implementation offers a wrapper that provides an "XML view of the resource". In the context of this paper, a resource is defined in terms of schema and value. In some cases the schema may be absent, e.g., flat file. Resources are grouped into families: two resources belong to the same family if they share the same schema extraction rules. The AIXO architecture, shown in Figure 1, is composed of three main components: *ResourceAccess*, *ResourceToXML*, *XSLTProcessor*.

ResourceAccess manages access to the resource to be wrapped. Its implementation depends upon the communication protocol, permissions, and access policies. By using the *ResourceAccess*'s interface, data can be gathered from the resource in its native format; there is no transformation. For example, in the case of a Relational DataBases (RDB), the data obtained is contained in a "recordset".

ResourceToXML transforms data, provided by the *ResourceAccess* component, into XML. The transformation is *canonical* and independent of the data's semantics. Mapping from the original format to XML is performed considering only the data's structure. For example, in transforming a recordset to XML, the output conforms exactly to the schema of the table; in the case of a flat file, the transformation will derive its structure taking into account special characters such as tabular and white spaces. For an HTML text, the transformation extracts the document schema from the tags.

Finally, the *XSLTProcessor* applies a set of XSLT filters to the raw XML, provided by the *ResourceToXML*, to obtain the effective XML view of the resource. In this phase, the semantics of data plays an important role.

To create a concrete wrapper the *DataSource* and *ResourceToXMLReader* Java classes must be implemented and the *XSLTProcessor* must be configured using the appropriate set of XSL Transformations. Each wrapper is defined by an XML configuration file. The system automatically loads classes and initializes attributes.

Thanks to inheritance of OOP languages, in our case Java, we obtain a flexible architecture and families of reusable components, allowing for the definition of new wrappers by reusing existing components. Below, we describe implementation details for each component.

ResourceAccess

The *ResourceAccess* component provides physical access to a resource. Whenever we need to read the data contained in a resource we must comply to the interaction rules required by the resource type. For example, if we need to read the data contained in a database, we will probably interact with a DBMS, while if we need to read data from a flat file we will probably use API. The *DataSource* abstract class provides a uniform way to access a resource by an activating method invoked each time a request is issued. The parameter of this method is a query that selects the data that must be extracted. The query format depends upon the target resource and can be expressed in well-know language, e.g. SQL, or in a proprietary language. The XML data model suffers with large resources (a resource containing a large amount of data), but the query avoids the transformation of the whole resource by selecting only a small portion of data. The access to a resource may require several parameters affecting the extraction phase, e.g., username and password in the case of a secure resource or

the URL, and the access method in the case of a web source. Each *DataSource* implementation is equipped with an XML configuration file storing all useful information. The final result provided by this class is data stored in a native format, for which no transformation is performed.

Similarly to the Tambis mediator architecture (Baker et al., 1998), the activating method can be invoked by a mediator that receives requests expressed in a standard language, and maps them in a resource specific format. In AIXO, the requests are performed by executing the access method with the query as parameter. Then the *ResourceToXML* and the *XSLT Processor* components receive and transform each result to a common XML schema. Finally, the mediator collects and merges all responses, providing the requestor with a unified result. We aim extending AIXO by developing a mediator to play this role in the system.

When the *DataSource* abstract class is implemented for a specific resource type, the *getAccess* method (shown in Figure 1) that selects a portion of the data based upon the query, must be implemented. If the target resource is a service, the implementation of the method will stimulate the service according to the query. In the case of an access to a static HTML page the parameter is composed of two strings *(key, value)* used to construct the query string or the body of the message depending upon the format of the request (HTTP Get or HTTP Post).

We have implemented classes allowing access to RDB, command line programs and web pages. This family of classes need not be implemented each time from scratch; in fact for similar resources it is often sufficient to update the XML configuration file.

ResourceToXML

The *ResourceToXML* component transforms data obtained from the *ResourceAccess* component to raw XML by looking exclusively at the syntax, so that the same transformation is used for all data in the same family. The goal of this step is to obtain a first XML representation of the data, whether it differs considerably from the intended structure or not. For example, the mapping from relational data extracted from a database to XML is performed by representing each record and all its fields in the target XML schema. If the document we obtain has a richer schema, further elaborations are facilitated (performed in the third step); on the contrary, if the document structure is poor, it will be difficult to perform complex semantic transformations.

We have initially considered three groups containing several families: unstructured, semi-structured and structured resources. For the first group, we have developed classes which translate to XML both, flat files in LALR(1) grammar and files without a well-defined grammar. The first transformation is performed by Chaperon (http://chaperon.sourceforge.net), while the latter is performed by hand-coding the extraction rules in the component. In the case of semi-structured data, we have considered XML and HTML data sources and developed the related classes. In the case of HTML, the Java class uses Jtidy Parser (http://sourceforge.net/projects/Jtidy) to remove syntactical errors and transform an HTML document in a well-formed XHTML document. In the case of structured data, we are presently considering only RDB recordset.

XSLTProcessor

The *XSLTProcessor* component performs transformations to enable viewing of XML data according user requirements. The mapping from a raw XML data to the final desired XML document is executed applying a sequence of XSLT filters. Each filter maps an XML document to another XML document. The final result, after the application of all filters, is an XML document compliant with the desired structure. The information necessary to the component for the dynamic application of the XSL Transformations are stored in an XML configuration file. The behavior of the *XSLTProcessor* component is fully implemented and configuration is the only activity that must be performed to specialize the component for a particular resource.

The wrapping process entails three steps, each clearly separating considerations of objectives. As the first step, AIXO accesses the resource and collects data in its "native" format. The second step is a syntactic transformation that maps the "native" format to XML. The last step applies a sequence of XSLT filters through which the final XML document is obtained. Figure 2 shows the flow of methods calls . The wrapping process is a pipe of three activities.

4 AIXO in action

AIXO has been experimentally proven on different resources in different contexts. In one example the web White Pages of two countries are collected in one large virtual repository through a common XML view. In example two, AIXO has been used to present a flat file as an XML file; this file contains biological data. Finally, we make AIXO interact with GAMS (http://www.gams.com/), a problem solving tool. AIXO executes GAMS in the command line modality, and it captures GAMS input and out-

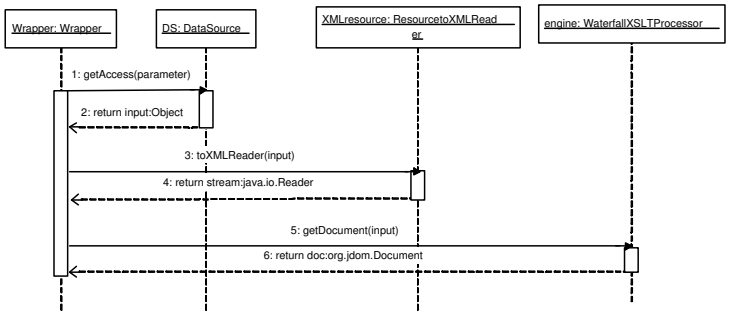

Figure 2: Operations performed during wrapping

put streams. Our next objective is to apply AIXO to BLAST, a very important command line tool in the molecular biology domain.

Example 1: HTML Semi-structured data

To test AIXO, we considered Italian White pages (http://paginebianche.virgilio.it/) and French White pages (http://wfe.pagesjaunes.fr/pb.cgi). The test is intended to demonstrate how the wrapper works with HTML pages, and how it facilitates the integration of information stored in web pages. In this case we need two wrappers, one for French White Pages and the other for Italian White Pages. Both wrappers contain the new *HTMLDataSource* class that is a specialization of the *DataSource* class. *HTMLDataSource* allows connection through HTTP protocol to a remote resource with a GET or POST request. Next the *HTMLToXMLResource* class, by the JTIDY parser, is used to normalize and transform HTML into XHTML. The *XSLTProcessor* component ultimately – using a XSLT filter – isolates data from the XHTML tags and inserts them into a more useful XML structure. In this example we have used two XSLT filters: one for the French White Pages and the other for Italian White Pages. Figures 3 and 4 show final XML documents.

Example 2: Unstructured Data

Flat files are commonly used because they can be easily written by text editor. Figure 5 provides an example of a flat file describing the result of a biological experiment; the file has been retrieved from the GEO Web site (http://www.ncbi.nlm.nih.gov/geo/).

Figure 3: French White Pages

To wrap the file, we extended the *DataSource* class, extracted the LALR(1) grammar generating the flat file, and generated the correspondent XML document using the *ResourceToXML* component. This mapping is performed by mapping each element of the syntax to an XML element. Finally, we applied the set of XSLT filters using the *XSLTProcessor* component. This set of filters suppresses some meaningless elements and modifies the structure of the other elements. The result of the elaboration is shown on Figure 6.

Example 3: Command Line Program

In scientific research, access to command line programs and extraction of data from the output of these programs is often required. This is true in the case of Blast (http://www.ncbi.nlm.nih.gov/BLAST/), which is commonly used in molecular biology research. In

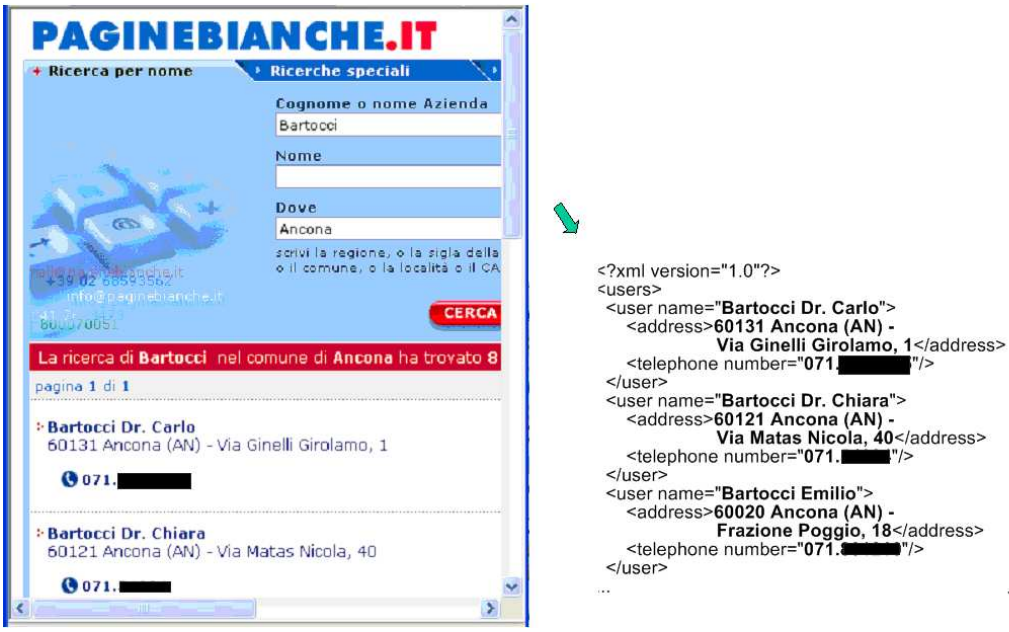

Figure 4: Italian White Pages

ÞLATFORM=GPL28

!Platform_sample_d=GSM11

!Platform_sample_id=GSM12

!Platform_sample_id=GSM13

ID	ROW	COLUMN	GENE

1	1	1	"Human mRNA for alpha-catenin, complete cds"

2	3	2	"Human ETS2 gene"

Figure 5: A Flat File Containing Biological Data

```
<?xml version="1.0" encoding="UTF-8"?>
<Gene_Platform>
    <platform id="GPL28"/>
    <samples>
        <sample id="GSM11"/>
        <sample id="GSM12"/>
        <sample id="GSM13"/>
    </samples>
    <MACROArray>
        <gene id="1" row="1" column="1">"Human mRNA for
        alpha-catenin, complete cds"</gene>
        <gene id="2" row="3" column="2">"Human ETS2 gene"</gene>
    </MACROArray>
</Gene_Platform>
```

Figure 6: XML document containing biological data

this paper we consider GAMS, used to solve linear programming problems in the operative research field. To build a wrapper for the GAMS output, we use the *cmdLineDataSource*, a specialization of the *DataSource* class that captures the output of the program in a stream. To interpret GAMS output, we have implemented a new extended class from *ResourceToXMLReader* called *GAMSToXMLReader*. As final step, we define the filter required to extract only the information we require. Figure 7 contains the GAMS output and Figure 8 shows the XML before the filter application.

5 Integration of AIXO in BioAgent

BioAgent (Merelli et al., 2002) enables information retrieval and distributed computations that adopt the mobile agent paradigm. An agent of the system interacts with other agents to accomplish its tasks. Typically, a task includes access to several resources and services, but agents often do not know how to interact with all types of resources. The goal is to provide the agent with the "XML view of the world". All wrapped resources are provided to agents as XML documents. The transformation of a resource concerns not only syntactic heterogeneity, but also semantic problems. So the final resource presentation style must be comprehensible to an agent, thus overcoming any syntactic and semantic issues. The use of ontologies (Fensel, 2001) might provide a way to specify the admissible output structure of a resource.

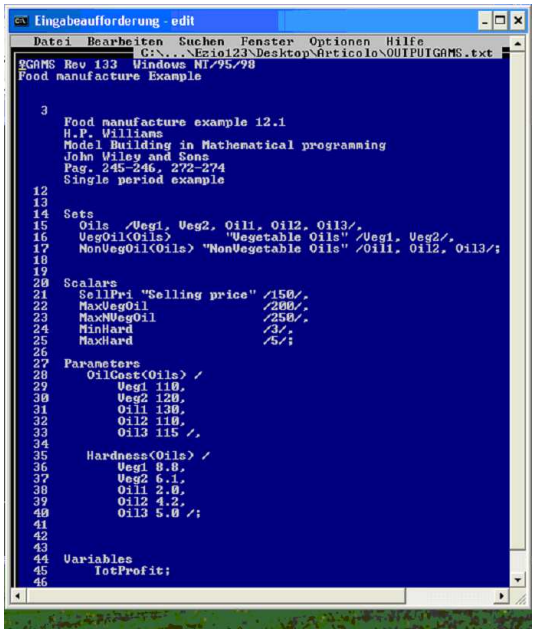

Figure 7: Command Line Program

BioAgent provides access to resources using both ontologies and wrappers. AIXO is used both to present resources as XML documents and to map XML-based ontologies. A document compliant with one ontology can be translated into a document compliant with another document. Results are encouraging; agents interact correctly with all resources and management of ontologies is simplified. A typical interaction between a bioscientist and BioAgent involves the following steps (Figure 9):

1. A BioScientist specifies the set of tasks to be performed.
2. The system generates a pool of agents to execute the task.
3. Agents migrate and clone in order to efficiently accomplish the task.
4. Agents query resources. AIXO implements the abstraction layer so that agents interact only with XML documents. In cases agents expect different type of documents (ontologies mismatching), AIXO can map documents together.
5. Agents merge results and furnish data to the bioscientists.

We have tested this system by clustering extensive DNA-microarray data retrieved from the GEO Web site. For the moment, interactions appear in read-only mode; we are investigating the possibility of accessing and modifying a resource using the XML document abstraction.

```
<?xml version="1.0"?>
<GamsOutput>
    <Date>23:57:47</Date>
    <Time>10/06/02</Time>
    <Title>Food manufacture Example</Title>
    <model>
        <Header>
            Food manufacture example 12.1
            H.P. Williams
            Model Building in Mathematical programming
            John Wiley and Sons
            Pag. 245-246, 272-274
            Single period example
        </Header>
        <sets>
            <set name="Oils">
                <description></description>
                <index>Veg1</index>
                <index>Veg2</index>
                <index>Oil1</index>
                <index>Oil2</index>
                <index>Oil3</index>
            </set>
            <set name="VegOil(Oils)">
                <description>Vegetable Oils</description>
                <index>Veg1</index>
                <index>Veg2</index>
            </set>
            <set name="NonVegOil(Oils)">
                <description>NonVegetable Oils</description>
                <index>Oil1</index>
                <index>Oil2</index>
                <index>Oil3</index>
            </set>
        </sets>
...
```

Figure 8: GAMS *raw* XML Document

6 Related Works

According to Kushmerick (Kushmerick, 1997) we can distinguish two main categories of wrappers: Hand-Coding Wrappers and Boosted Wrappers. Boosted Wrappers can learn, after an initial learning stage, how a document is structured and how to extract its data. Hand coding wrappers are configured by human users or with the support of tools that allow automatic configuration. The drawbacks of Boosted Wrappers are limited expressive power and the large number of required sample pages (Baumgartner et al., 2001b). AIXO belongs to the Hand-coding wrapper category; Table 1 describes the main features of AIXO compared to other widely used wrapper generation systems. NoDoSE (Adelberg, 1998) extracts data from the structured text document and considers HTML as a special case. XWrap (Liu et al., 2000) wraps HTML pages and assesses possible changes in the resource's structure. W4F (Sahuguet and Azavant, 1999b; Sahuguet and Azavant, 1999a) uses an SQL-like query language called HEL, but provides

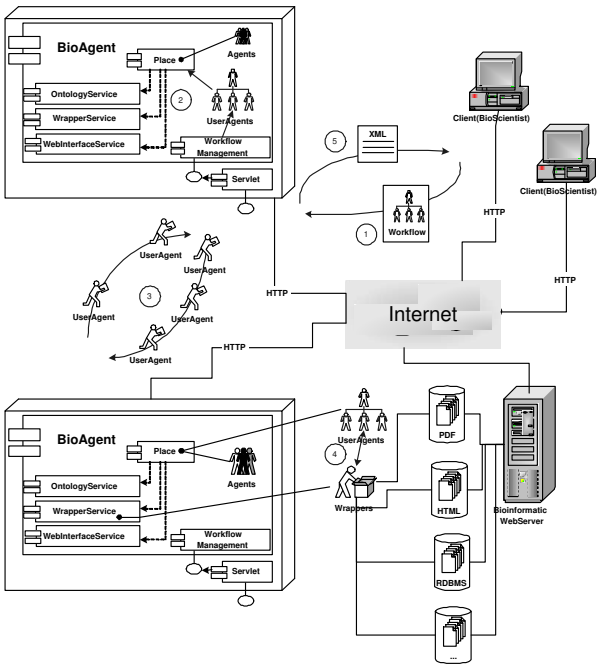

Figure 9: Interactions between agents and wrappers

little ease-of-handling. The user must be expert in the use of HEL and HTML to develop a wrapper. Lixto (Baumgartner et al., 2001b; Baumgartner et al., 2001a), thanks to GUI, allows the user to develop wrappers without knowledge of its extraction language Elog, but also deals essentially with HTML.

The problem of efficiently managing XML documents has been recently addressed in XMLTK (Avila-Campillo et al., 2002), a scalable toolkit enabling the manipulation of XML elements with a low memory-consumption.

AIXO, unlike the above systems, is an easily extendible architecture that promotes reusage of developed components. Classes developed and used for a particular type of resource can be shared with the rest of the community. The wrapping procedure is simple and linear involving three clearly distinct steps with different purposes: resource access, raw XML extraction, and generation of the final XML document.

Modularity, reuse and flexibility improve diffusion, but hinder performance. In fact the three steps could be optimized into one step. We propose the application of AIXO in contexts where performances are not the primary issue. To deal with a wide range of resources AIXO has been designed to support several protocols and resources. AIXO lacks a resource monitor, automatic wrapper generation and GUI, all system's parts that we aim to develop in the next version of AIXO.

7 Future Works and Conclusions

In this article we have presented Any Input XML Output (AIXO), a flexible and general wrapper architecture whose key features are flexibility, simplicity, the use of XSLT as extraction language, and the capacity to reuse and share implemented components. The use of XSLT permits the integration of existing tools and technologies into the system. Adoption of a standard language also means that a user approaching AIXO for the first time can employ previous knowledge to the new task. Running AIXO involves three distinct steps, each with different objectives and employing different technologies. These clear distinctions increase simplicity.

We use our Java implementation to evaluate the effectiveness of the system in different contexts: web pages, flat files and command line programs. We also integrate AIXO into BioAgent to facilitate access to resources and services. All results are encouraging. In the near future we intend to develop classes to wrap BLAST, a popular and widely used tool in the molecular biology field.

The use of a standard and well known language such as XSLT does not necessarily mean a more simplified method of constructing transformations. Our goal is to develop a graphic environment for the definition of XSLT transformation. The simplicity and intuitive use of a visual language would simplify the work of a generic user. A graphic console would also facilitate configuration of parameters and the creation of XML configuration files for all components of the systems.

Future work concerns the definition of a public repository for implemented classes to simplify sharing and reuse of implemented access protocols or XML transformers.

REFERENCES

Adelberg, B. (1998). NoDoSE a tool for semi-automatically extracting structured and semistructured data from text documents. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 283–294. ACM Press.

Avila-Campillo, I., Green, T. J., Gupta, A., Onizuka, M., Raven, D., and Suciu, D. (2002). XMLTK: An XML toolkit for scalable XML stream processing. In *Proceedings of Programming Language Technologies for XML (PLANX)*, Pittsburgh, PA.

Baker, P., Brass, A., Bechhofer, S., Goble, C., Paton, N., and Stevens, R. (1998). Tambis: Transparent access to multiple bioinformatics information sources. In *Proceedings of the Sixth International Conference on Intelligent Systems for Molecular Biology, ISMB98*, Montreal.

System	GUI	Input	Output	Architecture Flexibility	GPL	Extraction Language	Access	Resource Monitor
XWrap	Y	HTML	XML	N	N	XWrap	HTTP (Get)	Y
W4F	Y	HTML	XML, Java Classes	N	N	HEL	HTTP (Get and Post)	N
NoDoSe	Y	Structured Text Document	Generic or DBMS	N	N	NoDoSe	HTTP (Get)	N
Lixto	Y	HTML	XML	N	N	Elog	HTTP (Get)	N
AIXO	N	HTML, XML, DBMS, FlatFile	XML	Y	Y	XSLT	HTTP (Get and Post) JDBC-ODBC, Filesystem	N

Table 1: Systems Comparison

Baumgartner, R., Flesca, S., and Gottlob, G. (2001a). Supervised wrapper generation with Lixto. In *The VLDB Journal*, pages 715–716.

Baumgartner, R., Flesca, S., and Gottlob, G. (2001b). Visual web information extraction with Lixto. In *The VLDB Journal*, pages 119–128.

Bellwood, T., Clément, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y. L., Januszewski, K., Lee, S., McKee, B., Munter, J., and von Riegen, C. (2002). UDDI version 3.0. Published specification, Oasis.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284:34–43.

Fensel, D. (2001). *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag.

Gruber, T. R. (1993). A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220.

IONA (2002). http://www.iona.com/.

Kolp, M., Giorgini, P., and Mylopoulos, J. (2001). A goal-based organizational perspective on multi-agents architectures. In *In Proceedings of the Eighth International Workshop on Agent Theories, architectures, and languages (ATAL-2001)*, Seattle, USA.

Kreger, H. (2001). Web services conceptual architecture (WSCA 1.0). Technical report, IBM Software Group.

Kushmerick, N. (1997). *Wrapper Induction for Information Extraction*. PhD thesis, University of Washington.

Liu, L., Pu, C., and Han, W. (2000). XWRAP: An XML-enabled wrapper construction system for web information sources. In *International Conference on Data Engineering (ICDE)*, pages 611–621.

Merelli, E., Culmone, R., and Mariani, L. (2002). Bioagent: A mobile agent system for bioscientists. In *NETTAB02 - Agents in Bioinformatics*, Bologna.

Müller, M. E. (1999). An intelligent multi-agent architecture for information retrieval from the internet. Technical report, Institute for Semantic Information Processing.

Pierce, M., Youn, C., and Fox, G. (2002). Application web services. Technical report, Community Grid Labs, Indiana University.

R.Jennings, N., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *International Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7–38.

Sahuguet, A. and Azavant, F. (1999a). Looking at the web through XML glasses. In *Conference on Cooperative Information Systems*, pages 148–159.

Sahuguet, A. and Azavant, F. (1999b). Web ecology: Recycling HTML pages as XML documents using W4F. In *WebDB*, pages 31–36.

Sudmann, N. P. and Johansen, D. (2001). Supporting mobile agent applications using wrappers. In *DEXA Workshop 2001*, pages 689–695.

W3C (1998). Document object model (DOM) level 1 specification. W3C recommendation, W3C.

W3C (1999). XSL transformations (XSLT) version 1.0. W3C recommendation, W3C.

W3C (2000a). Extensible markup language (XML) 1.0 (second edition). W3C recommendation, W3C.

W3C (2000b). Simple object access protocol (SOAP) 1.1. W3C note, W3C.

W3C (2001). Extensible stylesheet language (XSL) version 1.0. W3C recommendation, W3C.

W3C (2001). XML schema part 0: Primer. W3C recommendation, W3C.

W3C (2002a). Web service description usage scenarios. W3C working draft, W3C.

W3C (2002b). Web services description language (WSDL) version 1.2. W3C working draft, W3C.