An XML-based Serialization of Information Exchanged by Software Agents

Sînică Alboaie
Institute of Theoretical Computer Science, Romanian Academy and Iaşi branch
abss@iit.iit.tuiasi.ro

Sabin Buraga and Lenuţa Alboaie
Faculty of Computer Science, "A.I.Cuza" University of Iaşi, Romania
{busaco,adria}@infoiasi.ro – http://www.infoiasi.ro/~busaco/

Keywords: Software Agent, Serialization, XML, Distributed Resources

Received: July 15, 2003

In this paper, we present an agent-based object-oriented solution to access the Web distributed resources. We describe Omega *– an agent framework viewed as a hierarchical space of a set of distributed objects that models the Web resources. Also, we propose an XML-based model that can be used as a universal manner for serialization of the objects processed by the (mobile) agents. The serialization mechanism can use the Simple Object Access Protocol (SOAP) serialization facilities, also.*

1 Introduction

The primary goal of Tim Berners-Lee's vision of the Semantic Web [5, 12] is to develop different mechanisms to automatically exchange, by the software entities, knowledge on the Web instead of the conventional manner used for accessing distributed resources.

To do this, computer scientists need to achieve the following:

- To understand the semantic mechanism of all kinds of queries, and what kind of components the process of questioning the Web formally consists of;
- To rigorously capture, represent or symbolize the knowledge contained on the Web.

To accomplish this goal, we are designing and implementing a framework – *Omega* [2, 3] – for agent software development viewed as a tree-like space of a set of distributed objects that models the Web resources by using XML (Extensible Markup Language) [7, 9] constructs. The *Omega* system offers a flexible framework for building agent-oriented distributed applications on the Web (see details in section 2 of this paper).

To assure the Web scalability, independently designed programs (especially Web agents) must be able to exchange and to process the meaning of data and metadata in an independent manner. Semantic interoperability can be completed only if different users (agents, tools, other Web clients, etc.) interpret XML – the actual *lingua franca* of the World-Wide Web computing entities – documents in the same way.

The *Omega* framework offers an addressing space for the Web objects and a mechanism for remotely accessing the Web distributed resources (objects). In section 2.2 of the paper we'll present the internal architecture of the *Omega* system, its functionality and base classes. A script-like language is provided, in order to implement an active (execution) part of the system and to integrate the *Omega* object space with notions such as execution thread, function, instruction, data types (see details in section 2.3).

To enable the flexible querying and accessing mechanisms about the distributed Web resources, we must offer a facility for serialization – in an independent way – of the data and metadata (objects) processed by the *Omega* agent system. In section 3, we investigate different possibilities of serialization given by the XML family of mark-up languages [9, 24]. Some of the drawbacks due of the lack of a description language regarding the objects' properties can be elegantly resolved by XML. Also, a SOAP-based serialization mechanism is presented and some advantages of the SOAP protocol are discussed (see section 3.2).

Even our approach can be used in the context of Web services discovery and description infrastructures, the paper does not intend to discuss these issues.

From the authors' point of view, the serialization of the Web objects can be considered as a flexible way to exchange information between software agents. Related multi-agent environments are presented in section 4 and some possible further development directions are exposed in the last section of the paper.

2 *Omega* Agent Framework

2.1 Motivation

We can consider as the fundamental resources that computers expose to the software components (i.e. operating system, applications) or users the following items: computing capabilities, (volatile or non-volatile) memory, local and remote data (documents), metadata (different descriptions about several properties of the resources: content, structure, layout/interface, dynamics, security issues, etc.).

Of course, there are other modalities to describe these properties without using XML-based assertions, but with the penalty of the platform and software independence. Obviously, these documents (including XML resources) are made to be read and processed in a distributed system (the Web itself). To easily access and obtain the knowledge contained by a specific document, a universal mechanism/model – based on the XML family – must exist to accomplish that. This is the seminal idea of the Semantic Web [5, 12].

WWW Space as a Distributed Hypermedia System

Also, the World-Wide Web space can be viewed as a distributed hypermedia system that uses Internet technologies (i.e. TCP/IP protocol family) – a global system of heterogeneous networked computers. Advances in networking and Internet/Web technologies are leading to a network-centric computing model, and the Web and Internet itself, of course, are evolving into the infrastructure for global network computing. By populating this infrastructure with object-based components and combining them in various ways, we can enable the development and deployment of interoperable distributed object systems on the Web.

The object model provides the ability to mimic real world processes in a fluid, dynamic and natural manner. The Web space allows for objects to be distributed to servers thereby centralizing access, processing, and maintenance, provides a multiplexing interface to distributed objects, and allows thin-clients (e.g., mobile phones or handheld devices). We can safely now state that **Web + Object** integration is a viable reality [24]. This is emphasized by different software organizations and companies – especially in the e-business domain – that are using Web-enabled distributed object technology, in the form of intranets and extranets, to solve their computing problems, and the emergence of an industry that provides Web and object interfaces to distributed object tools [4].

From the CGI Approach to a Distributed Object Infrastructure

But the Web didn't start out this way. Network-centric object computing is the result of a logical technological evolution. As originally conceived, it was driven by hypertext documents called Web pages or HTML documents [5, 9]. Initially, Web pages had static content (rich text and graphics at first, complex multimedia information later), and were interlinked. Browser applications running on user PCs or workstations were used to retrieve documents stored on Web servers. Helper applications supplemented the browser, handling other document types such as Word, PostScript, PDF (Portable Document Format) or different graphics, video, and audio formats. Web pages soon begun to include dynamic content as helper applications, called *plug-ins*, were integrated into the browser and CGI (Common Gateway Interface) scripts enabled users to input data to a Web server and access Internet services (i.e. data queries). Finally, programmatic content was added, on the client side, via Java applets, VBScript and JavaScript programs, to provide further interactive functionality and modify content in-place. These languages and techniques enable richer documents (e.g., animation and Web forms generated on-the-fly). Note that programmatic content can also include server-side execution of code such as accessing a remote database service (i.e. SQL queries) via specific Web application platforms (from CGI programs to PHP, ASP.NET or ColdFusion applications).

Prior to the addition of programmatic content, the Web was based on a client/server computing model which lacked scalability, common services, security, and a development environment needed to develop and deploy large-scale distributed applications. CGI scripts are not scalable because each requires a separate server-side process to handle each client request, services are limited to accessing database servers via CGI scripts, transaction information (such as credit card information) is not encrypted, and the programming model offered by HTML/HTTP using CGI and a three-tiered system is limiting.

With the advent of Java, and the distributed object infrastructures CORBA/IIOP and OLE/DCOM, the stage was set to evolve the Web from a document management system to a platform for distributed object computing and electronic commerce.

Bringing distributed objects to the Web offers the following advantages (to name only few of them):

- extensibility (e.g., for applications, services, and APIs built from objects, objects can easily be replaced or added);
- cross-platform interoperability;
- independent software development;
- reusable software components;
- componentware;
- network services;
- better utilization of system resources.

Existing legacy applications can even co-exist with distributed objects through the use of object wrappers. The interface could either be the client browser or browser-like with super-positioned distributed object infrastructures.

Mobile Agents

An important step towards *Internet/Web Computing* is represented by the mobile computations. A mobile object, usually called an *agent* when operating on behalf of a user, is a downloadable, executable object that can independently move (code and state) at its will – the mobile agent is not bound to the system in which it began the code execution and can travel from one node (host) on a network to another. Agent technology can be considered as a natural extension of object technology; conceptually, agents support a much richer and complex range of capability than objects, such as adaptability, cooperation, autonomy, negotiation and delegation [6, 17]. These capabilities give the possibility to build a sophisticated, expandable, maintainable, and distributed computing environment.

Mobile agents present the following main attributes [6, 15]:

- *reactivity* – the ability to respond to changes within agent environment;
- *autonomy* – the mobile agent is able to exercise control over its own actions (decisions);
- *goal-oriented* – the agents have a planned itinerary, they do not simply act in response to the environment;
- *communicative* – the ability to communicate with other agents, by exchanging information (knowledge); in this sense, agents present a collaborative behavior is order to achieve a common goal with other agents of the environment;
- *temporal continuity* – persistence of identity and state over long periods of time;
- *adaptability* – being able to learn and improve with experience;
- *mobility* – the mobile agents can transport themselves from one machine to another, in a self-directed manner.

Mobile agents provide a way to think about solving software problems in a networked environment that fits more naturally with the real world. Mobile agents can be used to access and manage information that is distributed over large areas [6].

The main benefit is that the software components can be integrated into a coherent and consistent software system – e.g. a multi-agent system – in which they work together to better meet the needs of the entire application (utilizing autonomy, responsiveness, pro-activeness and social ability).

The mobile agent architecture provides the "framework within which mobile agents can move across distributed environments, integrate with local resources and other mobile agents, and communicate the results of their activities back to the user. This framework can then be used to build mobile agents that perform user-driven tasks to fulfill distributed information management goals." [6]

Taking this notion further, the mobile agents could be used to monitor the network activities and provide input to QoS (Quality of Service) and global optimization mechanisms. They could be used during negotiation (with representative agents) to solve different constraint optimization problems.

One key research area is to provide security against malicious agents (who intend to access local resources or can carry a virus) and malicious hosts (who can alter the agent code/state or read private information).

Current mobile agent systems [17] – available as commercial or open-source applications – are implemented in different programming languages, such as C++, Java, Tcl, Scheme or Python.

2.2 Internal Architecture of the *Omega* System

Overview

Omega is an agent-based system that offers a tree-like addressing space for the Web objects and different techniques to remotely access the Web distributed resources (viewed as objects) [2, 3]. Each object processed by *Omega* can be viewed as a collection of objects included in that one. The links (edges) between the vertices of the tree are given by the aggregation relationship exposed by the object-oriented methodologies.

To emphasize the aggregation relationship, we attach to each object a name or an index, and in this way we can uniquely refer each object of the tree by its name/index (viewed as an identifier). Each object will have a unique list of the identifiers that represent its "address" in the addressing space used by the *Omega* agents. An identifier can be considered as an `IName` object (at the implementation level, an `IName` object can be viewed as an object-tree path or a list of object identifiers). By using a tree of objects, we can structure more easily the distributed resources for a given local web (such as a cluster or an intranet).

Functionality

We choose to use an interpreted environment for our multi-agent model and distributed object structure. Using such an environment, it was easier to consider serialization and various execution control mechanisms [11] which are contributed to the implementation of the *Omega* distributed object system.

Omega offers a distributed object structure, and its initial goal was to determine some good representations of data, types, instructions, functions and objects of an object-oriented language that can be used as a programming language for mobile agents. The result of this effort is a system written in C++ that is able to unify the notions behind the object-actor duality, namely the duality between passive and active objects [1]. From this point of view, *Omega* offers an infrastructure able to support Web-based distributed applications [18] (e.g., software agents used in clusters or Grid).

As an example, let us consider the problem of a system in which someone from a location *A* wish to obtain in real time data from another location *B*. There is more than one solution, and we present here just two possibilities [2]:

- Using the multi-agent paradigm, we create two agents in *A*: an agent for the information point, and an agent to be sent at the location *B* in order to obtain the information needed to be communicated to the location *A*. This approach is used in the design of *Omega* [3].
- Another way of solving this problem is to create a Web site at *B* (providing different server-side solutions [9] – i.e. CGI scripts or Java servlets) or a client/server application using a proprietary (TCP/IP-based) protocol [10].

We can observe that the first solution (the multi-agent approach) is more scalable and closer to follow certain good rules for programming design.

By using the multi-agent paradigm, a system can be easily divided into small entities with control over their interactions. Moreover, we can get a more flexible and adaptable approach (in our example, we can have a more adaptable way of presenting the information at the location *A*). This flexibility is a part of the client task, as opposed to the Web approach where we require more tasks for the server. In the case of using a client/server solution (with a proprietary communication protocol), some problems come from the high cost of the system design and maintenance.

The solutions that use C++ (networking, DCOM, CORBA) or even Java/C# are quite complicated and they are, in many cases, inappropriate for an open system, as the Internet – and Web, also – is. At the moment, for the generic problem of our example, a client/server solution is more popular in industry (in many cases based on HTML or XML). The later approach is adopted by Web services [18, 25] scenarios, also.

From the object-oriented paradigm's perspective, *Omega* can be seen as an object hierarchy that ensures a unitary way of programming, with an implementation of a name-service (presented in [2]) that is consistent for the resources (objects) that it makes available. The *Omega* system offers serialization mechanisms and garbage collection, also.

***Omega* Classes**

The `IObject` class is the base-class for every other class that has memory regions stored within a local system. Every object and function that needs a store space in *Omega* will use `IObject`. In this way, *Omega* assures a space model provided by a common distributed memory. This model is based on the existence of a given node of an `IObject`'s tree, which is easily addressable from the network.

Omega system offers a number of object types which provide functionality to the following classes:

- *String* class,
- *Number* class,
- *List* class,
- *Control* agent-execution class (i.e. support for virtual threads, scripting languages etc.).

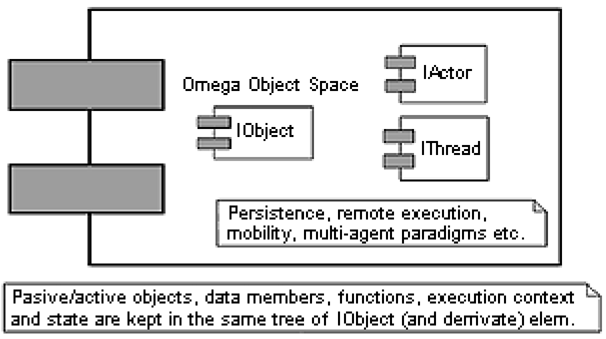
Omega Object Space
IActor
IObject
IThread
Persistence, remote execution, mobility, multi-agent paradigms etc.
Pasive/active objects, data members, functions, execution context and state are kept in the same tree of IObject (and derivate) elem.

Figure 1: *Omega* objects

Within the *Omega* framework, data types are represented by different classes such as `IString`, `INumber`, `IOmegaStack`, `IOmegaList`, `IOmegaQueue` that are derived from the `IObject` generic class (see also Figure 1).

Omega offers two categories of data types [3]:

- *Simple data types* – have no components (i.e. `INumber`, `IString`, etc.)

- *Compound data types* – represent a mix-up of two or more simple types (e.g., `IName`, `IOmegaList`, `IAThread`).

A compound data type can be considered as an “array” or a “struct” (very similar with the `struct` used in the standard C language).

In our approach, the string data type (`IString`) is not similar to the common concept of the “string”' type (present in all modern programming languages). At the implementation level, *Omega* system will use for `IString` another manner to store the content of a string (we do not use XML Schema's `xsd:string` – see details in [13]).

2.3 Omega Language

For the object system presented above, we provide an active (execution) part, which is the implementation of a scripting language that is using *Omega* objects. We can integrate the object space with notions such as execution thread, function, instruction, data types to be modeled with the help of `IObject` abstraction. The execution threads represented by an `IAThread` object (actor thread) will have a current execution context in which it can keep the local names and a global name list of the task (a task has more execution threads, some objects have attached execution threads, and they have the same name list from the task they belong to).

To simplify the development of a high-level control language, we are started from a data-type model that had `IString`, `INumber`, `IThread`, and `IObject` as base types and various types derived from `IOmegaActor` (this class is derived from `IActor`). The system is able to initialize and execute `IOmegaActor` objects.

Therefore the *Omega* object environment and the *OmegaKernel* mini-interpreter provide [2]:

- A *data model* (base type-system, the construction of new objects),
- An *address space* (every object has its own address consistent at the Internet – by using the TCP/IP stack – level),
- Techniques to implement the *high-level programming level statements* (e.g., `if`, `while`, or `goto`).

The *Omega* system is able to execute small ("scripting") programs. We present below such a program called *test program* – new `IObject`s are created. At runtime everything is reduced to a creation of new `IObject`s in the distributed space of objects.

```
# A simulation of while statement
OmegaTrace ("Test begin")
# OmegaTrace could be used
# for debugging purposes
BeginActor (SimulateDoWhile)
# BeginActor initializes
# an independent actor thread
NewINumber i 0
label begin
Inc i
OmegaTrace ("i++ in SimulateDoWhile")
LessThenGoTo i 2 begin
EndActor
SimulateDoWhile ()
OmegaTrace ("Test end")
```

The language provided by the *Omega* framework is similar to an assembler language and may be easily extended with other instructions. The main syntactic construct is similar to a function (method) call. An important step was to create a mechanism for representing data structures, statements and objects under the same abstraction (`IObject`) that is a network shared entity.

3 Serialization Mechanism

All classes derived from `IObject` must implement the *serialization* (*marshalling*) and *deserialization* (*unmarshalling*) methods. The process of building of the new data types is based on the fact that an `IObject` has a member of the `IOmegaList` type. That member contains associated links which are instances of the derived classes. In this manner, the serialization of the new types of objects can be automatically accomplished by *Omega* via members' serialization and the call of the overloaded own methods. Of course, for several types of objects – e.g. `IOmegaSocket` used for usual BSD-like socket operations [10], such as `bind()`, `listen()`, `accept()` or `connect()` – the serialization and deserialization activities can not be viewed as a proper solution.

For each access to a sharable object, a *proxy-object* is created, using the RPC (Remote Procedure Call) mechanism [9]. This proxy-object is placed in the same tree of the target object. In the tree of the accessed object, a *stub-object* is created, too. The stub will contain meta-descriptions about the sharable object and will be derived from `IObject`. The stub-object will be a member of the sharable object, to allow us to remotely access the stub. In this way, the system will be able to keep updated versions of the different object trees. To obtain the serialized form of an object, the RPC-like mechanism is able to transmit the URI (Uniform Resource Identifier) [9, 25] of that object. As a response, the system will get the serialized forms of the object and of the proxy-object as well, if it is possible. The *Omega* system is responsible to regularly update the proxy-objects.

The object serialization does not imply the serialization of the whole sub-tree that has as root the object in cause. For an object, only the serialization of the object itself and of the `IName` list of its children is done.

3.1 XML-based Serialization

The process of the *Omega*'s object serialization uses XML-based constructs. We use the XML namespaces defined by the XML Schema specification (see [13]) to retain the primary types of the data exchanged by agents in the serialization and deserialization processes.

An example is following (we are using an `IString` object):

```
<?xml version="1.0"?>
<IString>
   <name xsi:type="xsd:string">
      Hello from Omega
   </name>
</IString>
```

The *Omega* encoding style is based on the usual XML Schema's data types [13]. All data types used within the *Omega* system of agents must either be taken directly from the XML Schema or derived from *Omega* data types (see section 2.2).

The XML Schema specification (see *Datatypes* section from [13]) does not offer the possibility to express data types as XML elements, but only as attributes. To address this, the *Omega* framework declares a schema, called `OMEGA-ENC`, used to define an XML element for each data type (see the example below).

```
<OMEGA-ENC:int id="int1">
   33
</OMEGA-ENC:int>
```

Example
An example of *Omega* object serialization follows:

```
<element
   name="local_address_type"
   type="...">
  <simpleType
     name="local_address_type"
     base="xsd:string">
     <enumeration
        value="tree_id" />
     <enumeration
        value="unique_name" />
  </simpleType>
</element>
<element
   name="local_address"
   type="..." />
  <complexType
      name="local_address">
    <element
       name="la_type"
       type="local_address_type" />
    <element
       name="la_value"
       type="xsd:string" />
  </complexType>
</element>

<IName>
  <IOmegaDomain>
      ...
  </IOmegaDomain>
  <!-- info about local addr. -->
  <local_address>
     <la_type>
        tree_id
     </la_type>
     <la_value>
        1
     </la_value>
  </local_address>
  <local_address>
     <la_type>
        unique_name
     </la_type>
     <la_value>
        member_name
     </la_value>
  </local_address>
  <!-- other similar constructs... -->
</IName>
```

These XML elements could be used to extend the functionality of the *Omega* system with new data types.

We can note the *Omega* system only proposes the presented XML-based manner of object serialization, but does not interdict other mechanisms – e.g. SOAP-based serialization – to be adopted for data serialization.

3.2 SOAP-based Serialization

SOAP – or other protocols that use the RPC over XML approach (e.g., XML-RPC) – will be used to transport the serialized data. SOAP looks to be the right solution because of the great support it gets from different companies and organizations.

Short Description
SOAP (*Simple Object Access Protocol*) [14, 25] is a simple lightweight protocol used for XML-based structured and strong-type information exchange in a decentralized, distributed environment. The protocol is based on XML and consists of three parts:

- An envelope that describes the contents of the message and how to use it;
- A set of rules for serializing data exchanged between applications;
- A procedure to represent remote procedure calls, that is the way in which queries and the resulting responses to the procedure are represented.

Similar to object distribution models (e.g., IIOP and DCOM) [4], SOAP can invoke methods, services, components, and objects on remote servers. However, unlike these protocols, which use binary formats for the calls, SOAP uses a text format (Unicode), with the help of XML, to structure the nature of the exchanges.

SOAP can generally function with several protocols, such as FTP (File Transfer Protocol) or SMTP (Simple Mail Transfer Protocol), but it is particularly well-suited for the HTTP (HyperText Transfer Protocol) [9, 25]. It defines a reduced set of parameters that are specified in the HTTP header, making it easier to pass through proxies and firewalls. The use of SOAP over HTTP also enables resources already present on the Web to be unified by using the natural request/response model of HTTP protocol. The only constraint is that a SOAP message via HTTP must use the MIME (Multi-purpose Internet Mail Extensions) [9, 25] type `text/xml`.

Also, SOAP protocol can help in activities of message exchange and routing and agent communication by integrating well-known actual standards (e.g., The Foundation of Intelligent Physical Agents – FIPA agent standard [23]).

The actual SOAP implementations are available for a broad range of programming languages, such as C++, C#, Java, Perl, PHP or Python.

SOAP vs. CORBA
Although SOAP was initially intended as a remote method invocation protocol running over the Internet and using XML messaging, the SOAP protocol is not just another Common Object Request Broker Architecture (CORBA) [4, 20].

SOAP presents the subsequent significant improvements [14, 16]:

- *Human readability* – SOAP does not expose a binary format like CORBA Internet Inter ORB Protocol (IIOP); even if SOAP is mainly projected to be read by machines and to give support for Web services, human readability is very useful for debugging purposes and rapid and simple implementations;
- *Simple installation* – because SOAP is based on HTTP and XML, the protocol can be implemented with slight effort by using existing processing libraries for XML and HTTP; contrary, CORBA requires complex software packages and does not provide a commonly accepted bootstrapping mechanism.

The SOAP protocol has the potential to become the connecting point between heterogeneous distributed platforms and architectures, such as Sun ONE, Microsoft .NET, Perl or PHP scripting applications.

SOAP Data Model

SOAP is based on a simple object-oriented data model. The SOAP data model consists of structured objects having certain properties and a type. The SOAP specification allows, through a set of unambiguous rules, alternative syntax forms for embedded and referenced objects. Objects can be embedded if there exists only one referenced to them; otherwise they are linked [14, 25].

SOAP does not provide its own schema language. For this, the protocol uses XML Schema [13] for validation of the syntactical correctness of SOAP serialization model. Also, SOAP serialization fits fine into Unified Modeling Language (UML) modeling [20]. Even if SOAP describes instance serialization only, the UML meta-model can be utilized to serialize UML models using SOAP serialization syntax [16]. This can be a helpful feature in the activity of multi-agent system design.

In [16], the SOAP-based serialization mechanism is discussed in conjunction to Resource Description Framework (RDF) and the related Semantic Web activity. Also, RDF assertions can be used to store certain metadata about existing objects [8].

Example

A short example is following, when a request to invoke a remote method of an object is made and a response that contains the result is returned. The invoked method returns the services provided by a given node (agent) of the system.

The SOAP request can be (first five lines are HTTP header fields followed by the SOAP envelope marked-up in XML; the `SOAPAction` field specifies the action to be executed on the remote site):

```
POST /omega/interface HTTP/1.1
Host: 193.231.30.197
Content-type: text/xml
Content-length: nnn
SOAPAction: urn:omega.ro:Omega:#getSrv
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=
     "http://schemas.xmlsoap.org/
             soap/envelope/"
      SOAP-ENV:encodingStyle=
        "http://schemas.xmlsoap.org/
            soap/encoding/">
   <SOAP-ENV:Body>
      <o:getSrv
         xmlns:o="urn:omega.ro:Omega">
         <node ip="193.231.30.225">
            thor.infoiasi.ro
         </node>
      </o:getSrv>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A possible response (on success) can be the following (first three lines denote the response given by Web server, followed by SOAP data – in this case an XML-based document that contains the list of the existing agents and additional information about them):

```
200 OK
Content-type: text/xml
Content-length: mmm
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=
     "http://schemas.xmlsoap.org/
             soap/envelope/"
      SOAP-ENV:encodingStyle=
        "http://schemas.xmlsoap.org/
            soap/encoding/">
   <SOAP-ENV:Body>
      <o:listSrv
         xmlns:o="urn:omega.ro:Omega">
         <service desc="...">
            <stateInformation>
               ...
            </stateInformation>
            <securityInformation>
               ...
            </securityInformation>
            <transportProfile>
               ...
            </transportProfile>
         </service>
         <service desc="...">
            <stateInformation>
               ...
            </stateInformation>
            <securityInformation>
               ...
            </securityInformation>
            <transportProfile>
               ...
            </transportProfile>
         </service>
         <!-- other information -->
      </o:listSrv>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Using *gSOAP* for Data Serialization
We are using an existing tool named *gSOAP* [21], which is able to generate the code for serialization from a user-defined specification.

Most toolkits for C++ Web services adopt a SOAP-centric view and offer APIs for C++ that require the use of class libraries for SOAP-specific data structures. This often forces a user to adapt the application logic to these libraries. In contrast, the *gSOAP* compiler tools provide a unique SOAP/XML-to-C/C++ language binding to ease the development of SOAP/XML Web services and clients in C and/or C++ languages.

The compiler enables the integration of (legacy) C/C++ programs, embedded systems, and real-time software in SOAP applications that share computational resources and information with other SOAP applications, possibly across different platforms, language environments, and disparate organizations located behind firewalls.

4 Related Work

Although there is not a formal framework for multi-agent systems development, due to dependence on application domains, it has been that the construction of these systems requires a different approach from that of conventional software systems development.

We are aware of multiple platforms developed both in academia and software industry companies [17]. This confirms that many computer scientists are considering the agent-oriented software as a possible paradigm, designed and implemented especially in very dynamic environments (such as World-Wide Web space). We can give different examples of frameworks and tools used to develop multi-agent systems (for more details, see [17]), some of them using the Internet open standards:

- *Tryllian's ADK (Agent Development Kit)* – an agent-based business integration platform, designed and built in Java, XML and JXTA with a modular architecture and a unique mobile component approach;
- *Toshiba's Bee-gent (Bonding and Encapsulation Enhancement Agents)* – a CORBA-based communication framework intended to provide co-operative processing in the advanced network society;
- *FIPA-OS* – a Java component-based layered toolkit enabling rapid development of FIPA (Foundation for Intelligent Physical Agents) compliant agents;
- *Grasshoper* – an open-source CORBA-based platform that allows software agents to move between different fixed and wireless computing systems and to execute various tasks in the process; this platform provides support for MASIF (Mobile Agent System Interoperability Facility) – a standard specification developed by the Object Management Group (OMG) [20];
- *JADE (Java Agent DEvelopment Framework)* – a widely used agent platform that can be distributed across heterogeneous machines and that can be configured via a remote GUI (Graphical User Interface);
- *Xraptor* – a simulation environment for continuous virtual multi-agent systems written in C++ for UNIX platforms that allows studying the behavior of agents in different 2- or 3-dimensional worlds.

Another interesting approach is *Agentcities* – a world wide initiative designed to help realize the commercial and research potential of agent based applications by constructing an open distributed network of platforms hosting diverse agents and services [19].

However, the existing implementations have not convinced the whole community or do not cover or provide certain facilities desired by programmers or final users. Some proprietary solutions, though well developed, are not built as open systems and can not be easily extended or modified. On the other hand, we were not impressed by the available open-source platforms. Therefore, from the authors' point of view, it was more useful and interesting to design and implement new systems, hoping that they will cover and combine better features.
The existing multi-agent platforms use different approaches for communication between agents, by using low-level communication protocols (TCP/IP, SMTP and HTTP) or standard high-level languages – such as KQML (Knowledge Query Manipulation Language) [6, 17]. One of the noticed difficulties is to design a platform-independent inter-agent communication language.

The *Omega* system presents an advantage, by adopting an XML-based platform-independent approach in serialization and exchanging information between agents. The SOAP model is more flexible and easy to use than CORBA or DCOM solutions. Some of the *Omega*'s facilities could be also integrated, for example, into the MAIS (Mobile Agents Information System) – a platform for creating dynamic clusters [15].

5 Conclusion

We have used the design principles of the distributed systems to develop our own software platforms and ideas related to the multi-agent paradigm and actor spaces (see also [1, 11]). From this point of view, the *Omega* project represents an infrastructure able to support the agent-oriented programming and to assure an XML-based flexible way for object serialization.

The paper focused on different platform-independent methods of exchanging information between the entities of a multi-agent infrastructure – *Omega* – presented in section 2.2. The *Omega* project can be viewed as a platform for developing distributed object middleware components [4]. *Omega* proposes a distributed object structure, and its initial goal was to determine some good representations of data, types, instructions, functions and objects of an object-oriented language that can be used as a programming language for mobile agents. The language provided by the *Omega* environment is a simple scripting language described in section 2.3.

To proper exchange information between the entities of a multi-agent system, an XML-like messaging solution is proposed. All classes within the *Omega* system must implement certain serialization (marshalling) and deserialization (unmarshalling) methods. The process of the *Omega*'s object serialization uses XML-based constructs and is detailed in section 3.1. Another method for object serialization is the use of SOAP-based serialization (see section 3.2).

Using these approaches, the *Omega* multi-agent system could integrate different Web services or could be integrated into complex distributed architectures such as Grid [18].

As a further research work, the proposed model for serialization will be used to exchange knowledge (using RDF, DAML+OIL or OWL assertions, for example) [8, 9, 12, 25] between intelligent Web agents. This research direction can be viewed as an effort to give support for Semantic Web projects [12, 21].

Also, we intend to experiment an XML-based version of the *Omega* language to be used to exchange mobile code of the software agents coded within the *Omega* framework.

References

[1] G. Agha, C. Callsen (1993) Actor Spaces: An Open Distributed Programming Paradigm, *Proceedings of the 4th ACM Symposium on Principles and Practice of Parallel Programming*, ACM Press

[2] S. Alboaie, S. Buraga, L. Alboaie (2002) An XML-based Object-Oriented Framework for Developing Software Agents, *Scientific Annals of the "A.I. Cuza" University*, Computer Science section, Tome XII, "A.I. Cuza" University Press, Iaşi, Romania, pp.109--134

[3] S. Alboaie, G. Ciobanu (2002) Designing and Developing Multi-Agent Systems, in *International Symposium on Parallel and Distributed Computing (ISPDC) Proceedings*, Scientific Annals of the "A.I. Cuza" University, Computer Science section, Tome XI, "A.I. Cuza" University Press, Iaşi, Romania, pp.142--153

[4] D. Bakken (2001) Middleware, in *Encyclopedia of Distributed Computing*, Kluwer Academic Press

[5] T. Berners-Lee (1999) *Weaving the Web*, Orion Business Books, London, UK

[6] J. Bradshow (1997) *Software Agents*, AAAI Press

[7] T. Bray *et al.* (eds.) (2000) *Extensible Markup Language (XML) 1.0 (Second Edition)*, World-Wide Web Consortium's Recommendation, Boston: `http://www.w3.org/TR/REC-xml`

[8] S. Buraga, S. Alboaie, A. Alboaie (2003) An XML/RDF-based Proposal to Exchange Information within a Multi-Agent System, in D. Grigoraş et al. (eds.), *Proceedings of NATO Advanced Research Workshop on Concurrent Information Processing and Computing*, IOS Press (to appear)

[9] S. Buraga (2001) *Web Technologies* (in Romanian), Matrix Rom, Bucureşti, Romania

[10] S. Buraga, G. Ciobanu (2001) *Programming Workshop in Computer Networks* (in Romanian), Polirom, Iaşi, Romania

[11] C. Callsen (1997) *Open Distributed Heterogeneous Computing*, PhD Thesis, University of Illinois at Urbana-Champaign

[12] J. Davies, D. Fensel, F. van Harmelen (eds.) (2003) *Towards the Semantic Web*, John Wiley & Sons

[13] D. Fallside (ed.) (2001) *XML Schema*, World-Wide Web Consortium's Recommendation, Boston: `http://www.w3.org/TR/xmlschema-0/`

[14] C. Gorman (2001) *Programming Web Services with SOAP*, O'Reilly and Associates

[15] D. Grigoraş et al. (2002) *MAIS – The Mobile Agents Information System Support for Creating Dynamic Clusters*, in Proceedings of ICA3PP, Beijing, China

[16] S. Haustein (2001) Semantic Web Languages: RDF vs. SOAP Serialization, *Proceedings of Semantic Web Workshop*, Hongkong, China

[17] E. Mangina (2003) *Review of Software Products for Multi-Agent Systems*, AgentLink.org: `http://www.agentlink.org`

[18] L. Moreau (2002) Agents for the Grid: A Comparison with Web Services (Part I: the transport layer), *IEEE International Symposium on Cluster Computing and the Grid Proceedings*, IEEE Press

[19] * * *, *Agentcitites Network*: `http://www.agentcities.net/`

[20] * * *, *Object Management Group Activity*: `http://www.omg.org/`

[21] * * *, *Semantic Web*: `http://www.semanticweb.org/`

[22] * * *, *SOAP Software*: `http://www.soapware.org/`

[23] * * *, *The Foundation of Intelligent Physical Agents* (*FIPA*): http://www.fipa.org/

[24] * * *, *Web Object Integration*: http://www.objs.com/survey/web-object-integration.htm

[25] * * *, *World Wide Consortium's Technical Reports*: http://www.w3.org/TR/