

# An XML-based Serialization of Information Exchanged by Software Agents

Sinică ALBOAIE  
Institute of Theoretical Computer Science,  
Romanian Academy, Iași branch  
abss@iit.iit.tuiasi.ro

Sabin C. BURAGA  
Faculty of Computer Science,  
“A.I. Cuza” University of Iași, Romania  
busaco@infoiasi.ro

Lenuța ALBOAIE  
Faculty of Computer Science,  
“A.I. Cuza” University of Iași, Romania  
adria@infoiasi.ro

## ABSTRACT

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 agents. The serialization mechanism can use SOAP protocol, also.

**Keywords:** Software Agent, Serialization, XML, Distributed Resources

## 1. INTRODUCTION

The primary goal of Tim Berners-Lee's vision of the Semantic Web [4] 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 accomplish this goal, we are designing 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) [5] constructs. The *Omega* system offers a flexible

framework for building agent-oriented distributed applications on the Web.

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 space – 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). 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 this paper, we investigate different possibilities of serialization given by the XML family [12]. Some of the drawbacks due of the lack of a description language regarding the objects' properties can be elegantly resolved by XML. Even our approach can be used in the context of Web services discovery and description infrastructures, the paper do not intend to discuss these issues.

From our point of view, the serialization of the Web objects can be considered as a flexible way to exchange information between software agents.

## 2. OMEGA AGENT FRAMEWORK

### 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 [7] 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. From this point of view, *Omega* offers an infrastructure able to support Web-based distributed applications [10] (e.g., software agents used in clusters or Grid).

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 [2] that is consistent for the resources (objects) that it makes available. The *Omega* system offers serialization mechanisms and garbage collection, also.

For example, 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 Classes** *Omega* system offers a number of object types which provide functionality to the following classes: *String*, *Number*, *List*, and *Control* agent-execution classes (i.e. support for virtual threads, scripting languages etc.).

In the *Omega* framework, the data are represented by different classes such as *IString*, *INumber*, *IOmegaStack*, *IOmegaList*, *IOmegaQueue* that are derived from the *IObject* generic class.

*Omega* offers two categories of data types [3]: *simple data types* – have no components (i.e. *INumber*, *IString*, etc.) – and *compound data types* – represent a mix-up of two or more simple types (e.g., *IName*, *IOmegaList*, *IAThread*).

### 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 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), and different techniques to implement the *high-level programming level statements*.

*Omega* is able to execute small (“scripting”) programs. We present below such a program called *test program* – new *IObjects* are created. At runtime everything is reduced to a creation of new *IObjects* in the distributed space of objects.

```
OmegaTrace ("Test begin")
BeginActor (SimulateDoWhile)
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 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 socket operations) 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 [6]. 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) [12] of the 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.

#### 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 [8] to retain the primary types of the data exchanged by agents in the

serialization and deserialization processes. The *Omega* encoding style is based on the usual XML Schema's data types. 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.

An example 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>
  <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>
  <!-- more other similar constructs... -->
</IName>

```

These XML elements could be used by the programmer to extend the functionality of the *Omega* system with new data types. The *Omega* system only proposes the presented XML-based manner of object serialization, but does not interdict other mechanisms to be adopted for serialization.

#### SOAP Object 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 (Simple Object Access Protocol)* [9,w3c] is a simple lightweight protocol used for XML-based structured and strong-type information exchange in a decentralized, distributed environment. The use of SOAP over HTTP enables resources already present on the Web to be unified by using the natural request/response model of the HTTP [12] protocol.

We use an existing tool named *gSOAP* [11], which is able to generate the code for serialization from a user-defined specification. 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. CONCLUSION AND FURTHER WORK

We are aware of multiple agent-oriented frameworks developed both in academia and software industry. 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 WWW space). One of the noticed difficulties is to design a platform-independent inter-agent communication language.

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 [1, 7]. From this point of view, *Omega* represents an infrastructure able to support the agent-oriented programming and to assure an XML-based flexible way for object serialization. As a further research work, the proposed model for serialization will be used to exchange knowledge (i.e. by using RDF or DAML+OIL statements) between intelligent Web agents. 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.

#### 5. REFERENCES

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

[2] S. Alboaie, S. Buraga, L. Alboaie, "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, Iași, 2002.

[3] S. Alboaie, G. Ciobanu, "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, Iași, 2002.

[4] T. Berners-Lee, **Weaving the Web**, Orion Business Books, London, 1999.

[5] T. Bray *et al.* (eds.), **Extensible Markup Language (XML) 1.0 (Second Edition)**, W3C Recommendation, Boston, 2000: <http://www.w3.org/TR/REC-xml>

[6] S. Buraga, G. Ciobanu, **Programming Workshop in Computer Networks** (in Romanian), Polirom, Iași, 2001.

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

[8] D. Fallside (ed.), **XML Schema**, W3C Recommendation, Boston, 2001: <http://www.w3.org/TR/xmlschema-0/>

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

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

[11] \* \* \*, **SOAPware**: <http://www.soapware.org/>

[12] \* \* \*, **World Wide Consortium's Technical Reports**, Boston, 2003: <http://www.w3.org/TR/>