An XML/RDF-based Proposal to Exchange Information within a Multi-Agent System

Sabin Buraga[1], Sînică Alboaie[2], Lenuţa Alboaie[1]

[1] Faculty of Computer Science, "A.I. Cuza" University of Iaşi, Romania
{busaco,adria}@infoiasi.ro
[2] Institute of Theoretical Computer Science, Romanian Academy, Iaşi branch
abss@iit.iit.tuiasi.ro

Abstract. The paper presents different platform-independent methods of exchanging information between the members of a multi-agent system. We describe first a multi-agent infrastructure – called *Omega* – that can be considered as a hierarchical space of a set of distributed objects that models the Web resources. We propose an XML/RDF-based model that can be used as an universal manner for serialization and metadata description of the objects processed by the agents. Various relationships that can be established between the entities of a multi-agent system will also be described by RDF constructs.

1 Introduction

The primary goal of Tim Berners-Lee's vision of the Semantic Web [4, 13] is to enable intelligent queries for knowledge on the Web instead of the conventional mechanisms to access the 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; and to rigorously capture, represent or symbolise the knowledge available on Web.

To accomplish this goal, we are designing and implementing an infrastructure for agent software development, called *Omega* [2], viewed as a tree-like space of a set of distributed objects that models the Web resources by using XML/RDF assertions. The *Omega* system offers a flexible framework for building agent-oriented distributed applications on the Web – consult section 2 for details. To assure the Web scalability, independently designed programs – especially Web agents – must be able to exchange and process the meaning of data and metadata in an independent manner. Semantic interoperability can be completed only if different users (agents, Web services, other Web clients, etc.) interpret RDF statements 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 (that can be viewed as objects). To enable the flexible querying and accessing mechanisms

about the distributed Web resources, we must offer a facility for serialization – in an independent manner – the data and metadata (objects) processed by the *Omega* multi-agent system. The paper investigates various possibilities of serialization given by the XML family [7, 24]. Some of the drawbacks due of the lack of a description language regarding the objects' properties can be elegantly resolved by XML.

The serialization of the Web objects presented in section 3 can be considered as a flexible way to exchange information between software agents. Additionally, for each object, different metadata constructs can be attached to specify several semantic properties. These descriptions are written in RDF and presented in section 4.

Several relationships can be established between the entities of the multi-agent system. Following [9, 10], these relations can be easily expressed by RDF assertions. This machine-understandable approach can ease the development of Web-oriented software applications – e.g., Web intelligent agents, Web mediators or Web services – for different activities such as resource discovery and queries that involve time.

2 *Omega* Multi-Agent Infrastructure

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: the 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 (mobile) distributed system (the Web itself). To easily access and obtain the knowledge contained by a specific document, there must exists an universal mechanism/model – based on the XML family – to accomplish that. This is the seminal idea of the Semantic Web [4].

WWW Space as a Distributed Hypermedia System The Web can be viewed as a distributed hypermedia system that uses Internet technologies, a global system of heterogeneous networked computers. Advances in networking and Web/Internet technology are leading to a network-centric computing model, and the Web and Internet itself are evolving into the infrastructure for global network computing. By populating this infrastructure with object-based components and combining them in various ways, the development and deployment of interoperable distributed object systems is fast migrating on Web [23].

The object model provides the ability to mimic real world processes in a fluid, dynamic and natural way. The WWW space allows for objects to be distributed to servers thereby centralizing access, processing, and maintenance, provides a multiplexing interface to distributed objects, and function as a catalyst for the rapidly-growing world of thin-clients – i.e. mobile phones, handheld devices, intelligent appliances. We can safely now state that **Web + Object** integration is a viable reality. This is emphased 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.

After the CGI standard, 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.

Existing legacy applications can even co-exist with distributed objects through the use of object wrappers [23]. The interface could either be the client browser or browser-like with superpositioned 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 on a network to another.

Mobile agents present the following important attributes: *reactive* (the ability to respond to changes within agent environment), *autonomous* (the mobile agent is able to exercise control over its own actions), *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), and *mobile* (the mobile agents can transport themselves from one host to another).

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, 16, 17].

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 (utilising autonomy, responsiveness, pro-activeness and social ability).

Current mobile agent systems – available as commercial or open-source applications – are implemented in C++, Java, Tcl, Scheme, and Python programming languages (to name only few).

2.2 General Architecture of *Omega*

Overview The *Omega* is an agent-based system that offers an addressing space (viewed as a tree) for the Web objects and different techniques to remotely access the Web distributed resources (viewed as objects) [2].

Each object processed by *Omega* can be considered 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 emphase 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 an 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 implement serialization and various execution control mechanisms [11] which contributed to the implementation of the *Omega* distributed objects 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.

Currently, *Omega* offers to active programs what the World-Wide Web space provides by default for presentation of some static entities (documents), namely an infrastructure able to support Web-based distributed applications (e.g., agents used in clusters or Grid [21]).

***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.

The *Omega* system offers a number of object types which provide functionality to the following classes: *String*, *Number*, *List*, and *Control* agent-execution (i.e. support for virtual threads, scripting languages etc.). Data is represented by different classes such as `IString`, `INumber`, `IOmegaStack`, `IOmegaList` that are derived from the `IObject` generic class.

Omega offers two categories of data types [2]: *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`).

2.3 *Omega* Language

Omega provides an active part, namely an execution part, which is the implementation of a script-like programming language. In this active part, we are trying to integrate the object space with notions such as execution thread, function, instruction, data types to be modeled with the help of `IObject` abstraction.

Therefore the *Omega* object environment and the *OmegaKernel* mini-interpreter provide (this makes *Omega* able to execute small scripting programs):

- A data model (base type-system, the construction of new objects);
- An address space (every object has its own Internet-consistent address);
- Techniques to implement the high-level programming level statements.

The language provided by the *Omega* framework may be easily extended with other instructions to build a more complex computational model of the developed agents. 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

Several interactions between the Web agents developed within *Omega* system can be accomplished by using serialization mechanisms.

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 stores 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. `IOmegaSockets` used for socket operations) the serialization and deserialization activities can not be viewed as a proper solution.

The object serialization do not imply the serialization of the whole subtree 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 [1].

XML-based Serialization As an optimal manner to serialize the *Omega* objects, we choose an XML-based representation. To describe these objects, we could adopt an RDF-based model. The RDF assertions could give the possibility to express semantics of an *Omega* object.

We are using the XML namespaces defined by the XML Schema specification [14] 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 [1]:

```
<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>
```

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*) [15, 24] is a simple lightweight protocol used for 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, and 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), SOAP can call methods, services, components, and objects on remote servers. However, unlike these protocols, which use binary formats for the calls, SOAP uses text format (Unicode), with the help of XML, to structure the nature of the exchanges.

SOAP can generally operate 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) [24]. It defines a reduced set of parameters that are specified in the HTTP header, making it easier topass through proxies and firewalls. Using SOAP over HTTP also enables resources already present on the Web to be unified by using the natural request/response model of HTTP protocol.

We use an existing tool named *gSoap* [22], which is able to generate the code for serialization from a user-defined specification. The *gSOAP* compiler tools provide an 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.

4 Describing *Omega* Objects in RDF

4.1 Resource Description Framework

Short General Presentation *Resource Description Framework* (*RDF*) allows the description of the metadata associated of the Web documents (resources). RDF consists of a model for the representation of the named properties and property values. This is suitable to model objects behaviours. RDF properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs. RDF properties can also signify relationships between resources. In object-oriented design terminology, resources correspond to objects and properties correspond to instance variables [8, 19].

To facilitate the definition of metadata, RDF is based on *classes*. A collection of classes, typically designed for a specific purpose or domain, is called a *schema* [8]. Through the sharability of schemas, RDF supports the reusability of metadata definitions. The RDF schemas may themselves be expressed in RDF. The RDF syntax is XML-based.

RDF Model The basic model of RDF consists of three object types [19]:

resources All objects being described by RDF expressions are called *resources* and they are always named by *Uniform Resource Identifiers* (*URI*) [5] plus optional anchor identifiers. Using URI schemas, every type of resource can be identified in the same manner.

properties A *property* is a specific aspect, characteristic, attribute, or relation to describe a resource. Each property has a specific meaning, defines its permitted values, the type of resources it can express, and its relationship with other properties (via RDF Schema).

statements A specific resource together with a named property, plus the value of that property for that resource is an RDF *statement*. These three individual parts of a statement are called, respectively, the *subject*, the *predicate*, and the *object*. The object of a statement (e.g., the property value) can be another resource or a literal.

RDF also specifies three types of container objects: *Bag* (an unordered list of resources or literals), *Sequence* (an ordered list of resources or literals), and *Alternative* (a list of resources or literals that represent alternatives for the single value of a property). The containers may be defined by a URI pattern. RDF can also be used to make statements about other RDF statements (higher-order statements).

The RDF data model provides an abstract, conceptual framework for defining and using metadata. Currently, there are several proposals of model-theoretical semantics for RDF and RDF Schema [12, 18, 24].

4.2 Using RDF to Capture the State of the *Omega* Objects

For each object of the *Omega* multi-agent system, we can attach different metadata. These meta-descriptions will assure the control versioning (the version and the last verification time-stamp for each object) and the owner/parent of the created objects. For security purposes, the object metadata will retain the list of the associated object's permissions. This approach is inspired from our RDF-based model used for accessing resources of the distributed file systems [9].

The system keeps these descriptions as RDF assertions that can be transported to other objects during the information updating activities (object replication).

In the stub object of any shared objects, certain metadata is available to inform other objects about the object's author and about the permission list to access this object. The system verifies this information to grant or to deny the access to considered object. The meta-descriptions regarding the permissions and the identity of the user who want to access *Omega* objects must provide a certain cryptographic support.

4.3 Expressing Relations between *Omega* Entities

To catch the dynamics of the involved agents built within *Omega* and the links between them, a high-level RDF-based description of temporal relations can be adopted. The temporal relationships between objects could be stored by RDF constructs, too. This approach is very similar to our model used to retain temporal relations established between (fragments of) Web sites [10].

The proposed model is mainly focused on the description of interval temporal relations. The temporal structure introduced by Interval Temporal Logic (ITL) is a simple linear model of time and is detailed in [3].

For this, an XML-based language is proposed – *Temporal Relation Specification Language* (*TRSL*) [10]. For each time relation, TRSL offers an element that corresponds to a specific relation (e.g. `<Meets>` element for $Meets$ relation from ITL model). The beginning and ending of time periods are denoted by `begin` and `end` attributes, respectively. Also, TRSL defines the `dur` attribute for specifying a known or predictive time period (this will allow Web agents to reason about different actions that may need to be performed).

The syntax and the semantics of TRSL language is detailed in [10].

Example We consider the following scenario. An object-maintainer Web agent can discover different temporal relations between the *Omega* objects distributed in an intranet and can automatically generate the following TRSL document:

```
<rdf:RDF>
  <rdf:Bag id="RecentlyChanged">
    <rdf:li resource="object1" />
    <rdf:li resource="object2" />
  </rdf:Bag>
  <rdf:Description rdf:aboutEach="#RecentlyChanged">
    <!-- spatial information -->
    <f:Location f:dns="www.site.org">
       193.231.30.1
    </f:Location>
    <!-- metadata information -->
    <f:Owner>
       <rdf:Description rdf:about="http://www.omega.site/">
          <f:Login f:uid="714">busaco</f:Login>
       </rdf:Description>
    </f:Owner>
    ...
    <!-- temporal information -->
    <t:link t:type="temporal" t:action="Serialize"
            t:end="Mon Mar 17 19:35:14 EET 2003">
       <t:Finishes t:dur="2sec" />
    </t:link>
  </rdf:Description>
</rdf:RDF>
```

A collection of objects to be serialized is denoted by `RecentlyChanged` identifier. This objects are stored on `www.site.org` machine and the serialization action is planned to be performed on March 17 2003. The metadata information describes the *Omega*'s host and the login name of the system maintainer.

This approach can be used for the resource discovery activities performed in a distributed (mobile) environment, in a machine-understandable manner.

Also, TRSL can store information about simple queries for temporal databases and can be used as an universal XML-based language for expressing different assertions in temporal query languages.

5 Related Work

Althrough 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 [6, 16].

We are aware of multiple platforms developed both in academia and software industry companies [20]. 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 Web). However, the existing implementations have not convinced the whole community or do not cover/provide some 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.

The existing multi-agent platforms use different approaches for communication between agents, by using low-level communication protocols (TCP/IP or HTTP) or standard high-level languages – such as KQML (Knowledge Query Manipulation Language) [6].

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 in the *MAIS (Mobile Agents Information System)* – a platform for creating dynamic clusters [17].

6 Conclusion

Omega represents an infrastructure able to support the agent-oriented programming. By this approach, we tried to emphase a trend which is shaping the evolution of the software development techniques for open distributed applications.

The paper focused on different platform-independent methods of exchanging information between the entities of a multi-agent infrastructure. We proposed an XML/RDF-based model that can be used as an universal manner for serialization and metadata description of the objects processed by the agents.

Various properties and relations established between the components (agents, objects, processes) of a multi-agent system can be expressed in a standardized and machine-understandable manner. This approach provides semantic descriptions of the Web resources and could be an interesting solution for exchanging knowledge between intelligent or/and mobile agents.

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

References

1. S. Alboaie, S. Buraga, L. Alboaie, *An XML-based Serialization of Information Exchanged by Software Agents*, 7th World Multiconference on Systemics, Cybernetics and Informatics – SCI 2003, Orlando, Florida, 2003 (to appear)
2. 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, Iaşi, 2002
3. J. Allen, P. Hayes, "Moments and Points in an Interval-based Temporal Logic", *Computational Intelligence*, 5 (4), 1989
4. T. Berners-Lee, *Weaving the Web*, Orion Business Books, London, 1999
5. T. Berners-Lee *et al.* (eds.), *Uniform Resource Identifiers (URI): General Syntax*, Internet Standard, RFC 2396, IETF, 1998
6. J. Bradshow, *Software Agents*, AAAI Press, 1997

7. T. Bray *et al.* (eds.), *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation, Boston, 2000: `http://www.w3.org/TR/REC-xml`
8. D. Brickley, R. V. Guha, *Resource Description Framework (RDF) Schema Specification 1.0*, W3C Candidate Recommendation, Boston, 2000: `http://www.w3.org/TR/2000/REC-xml-20001006`
9. S. Buraga, *A Model for Accessing Resources of the Distributed File Systems*, in Advanced Environments, Tools and Applications for Cluster Computing Proceedings of the NATO ARW Mangalia, Romania, 1-6 September 2001, D. Grigoraş *et al.* (eds.), Lecture Notes in Computer Science – LNCS 2326, Springer-Verlag, 2002
10. S. Buraga, G. Ciobanu, *A RDF-based Model for Expressing Spatio-Temporal Relations Between Web Sites*, in Proceedings of the 3rd International Conference on Web Information Systems Engineering (WISE 2002), 12-14 December 2002, Singapore, IEEE Press, 2002
11. C. Callsen, *Open Distributed Heterogeneous Computing*, PhD Thesis, University of Illinois at Urbana-Champaign, 1997
12. W. Conen, R. Klapsing, *A Logical Interpretation of RDF*, in Linköping Electronic Articles in Computer and Information Science, 5, 2000: `http://www.ida.liu.se/ext/epa/cis/2000/013/tcover.html`
13. S. Decker *et al.*, *Knowledge Representation on the Web*, in F. Baader (ed.), International Workshop on Description Logic (DL'00): `http://www.cs.vu.nl/~frankn/abstracts/DL00.html`
14. D. Fallside (ed.), *XML Schema*, W3C Recommendation, Boston, 2001: `http://www.w3.org/TR/xmlschema-0/`
15. C. Gorman, *Programming Web Services with SOAP*, O'Reilly and Associates, 2001
16. S. Green, F. Somers, *Software Agents: A Review*: `http://www.cs.tcd.ie/research_groups/aig/iag/iag.html`
17. D. Grigoraş *et al.*, *MAIS – The Mobile Agents Information System Support for Creating Dynamic Clusters*, in Proceedings of ICA3PP, Beijing, 2002
18. P. Hayes (ed.), *RDF Model Theory*, W3C Working Draft, Boston, 2002: `http://www.w3.org/TR/rdf-mt/`
19. O. Lassila, R. Swick (eds.), *RDF Model and Syntax Specification*, W3C Recommendation, Boston, 1999: `http://www.w3.org/TR/REC-rdf-syntax/`
20. E. Mangina, *Review of Software Products for Multi-Agent Systems*, AgentLink.org, 2002: `http://www.agentlink.org`
21. L. Moreau, *Agents for the Grid: A Comparison with Web Services (Part I: the transport layer)*, in IEEE International Symposium on Cluster Computing and the Grid Proceedings, Berlin, Germany, May 2002
22. * * *, *SOAPware*: `http://www.soapware.org/`
23. * * *, *Web Object Integration*: `http://www.objs.com/survey/web-object-integration.htm`
24. * * *, *World Wide Consortium's Technical Reports*, Boston, 2002: `http://www.w3.org/TR/`