computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific Journal of Computing

THE USE OF XML TECHNOLOGIES FOR EXCHANGING INFORMATION WITHIN A MULTI-AGENT SYSTEM

Sabin Buraga [1)], Sînică Alboaie [2)], Lenuța Alboaie [1, 2)]

[1)] Faculty of Computer Science, "A.I.Cuza" University of Iaşi,
Berthelot St., 16 – Iaşi, Romania, busaco@infoiasi.ro, http://www.infoiasi.ro/~busaco
[2)] Institute of Theoretical Computer Science, Romanian Academy, Iaşi branch, {abss,adria}@iit.iit.tuiasi.ro

Abstract: *This paper presents different XML-based techniques for exchanging information between the constituents of a multi-agent system. We expose a multi-agent infrastructure – called* Omega *– that can be considered as a hierarchical space of distributed objects set those models the Web resources. We suggest an XML/RDF-based model that can be used as a common approach for serialization and metadata description of the objects processed by the agents. Diverse relationships that can be established between the entities of a multi-agent system will also be expressed by different RDF constructs.*

Keywords: *Multi-Agent Systems, Web Resource, XML, RDF, Distributed Computing*

1. INTRODUCTION

The primary objective of Tim Berners-Lee's vision of the Semantic Web [4, 12] is to facilitate intelligent queries for knowledge on the Web instead of the conventional mechanisms to access the WWW space's resources. To accomplish this, computer scientists need 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 symbolize the knowledge existing on Web.

To achieve 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 dispersed objects that models the Web resources by using XML/RDF statements. The *Omega* system proposes 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 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 presents 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 have to offer certain facilities for serialization – in an independent manner – of the data and metadata (objects) processed by the *Omega* multi-agent system. The paper investigates various possibilities of serialization given by the XML family [6, 23]. 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 supple approach to exchange information between software agents. Moreover, for each object, different metadata constructs can be attached to denote several semantic properties. These descriptions are expressed as RDF statements and are presented in section 4. Several relationships can be established between the entities of the multi-agent system. Following [8, 9], these relations can be easily modeled by RDF assertions. This machine-understandable approach can ease the development of Web-oriented software applications for diverse activities such as resource discovery.

2. *OMEGA* – A MULTI-AGENT INFRASTRUCTURE

Context

We can consider as the primary resources that computers expose to the software components (i.e. operating system or/and applications) or users the

following items:

- computing capabilities,
- (volatile or non-volatile) memory,
- local and remote data (documents),
- metadata (diverse descriptions about some properties of the resources: content, structure, layout/interface, dynamics, security issues, etc.).

Of course, there are other ways to describe these properties without the use of XML-based assertions, but with the penalty of the platform and software independence. Clearly, 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, an XML-based mechanism must exist to accomplish that. In fact, this is one of the seminal ideas of the Semantic Web [4].

WWW as a Distributed Hypermedia System

The Web space 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 technologies are leading to a network-centric computing model, and the Web and Internet itself are developing into the infrastructure for worldwide network computing. By populating this infrastructure with object-based components and combining them in different ways, the development and deployment of interoperable distributed object systems is quick migrating on Web [22].

The object model gives the capability to mimic real world processes in a fluid, dynamic and expected manner. 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, smart appliances. We can safely now state that **Web + Object** integration is a viable reality.

This is highlighted by diverse software organizations and companies – especially in the e-business field – 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 (Common Gateway Interface) standard, with the advent of Java, and the distributed object infrastructures such as CORBA/IIOP and OLE/DCOM, the stage was set to evolve the Web from a document management structure to a platform for distributed object computing and e-business.

The actual legacy applications can still co-exist with distributed objects through the use of object wrappers [22]. The interface could either be the client browser or browser-like with super-positioned distributed object infrastructures.

Software Mobile Agents

An important step towards *Internet/Web Computing* is represented by the mobile calculus. A mobile object, usually called an *agent* when is operating on behalf of a user, is a downloadable, executable entity 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 go from one node on a network to another.

Mobile agents have the following important attributes:

- *reactive* (the capability to respond to changes within agent environment),
- *autonomous* (the mobile agent is capable to exercise control over its own actions),
- *goal-oriented* (the agents have a premeditated itinerary, they do not simply act in response to the environment),
- *communicative* (the skill to exchange information/knowledge with other agents),
- *mobile* (the mobile agents can transfer themselves from one host to another).

Mobile agents can be used to access and manage information that is distributed over large areas (see [5, 15, 16] for details).

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

Existing 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).

***Omega* System's Architecture**

General Description

The *Omega* is an agent-based system that presents an addressing space (considered as a tree) for the Web objects and diverse techniques to remotely access the Web distributed resources (viewed as objects) – see [2] for more details.

Each object processed by *Omega* can be considered as a group 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 add 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 single 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. By using a object tree, we can structure more easily the distributed resources for a known local web (such as a cluster or an intranet).

Omega *Functionality*

We decide to use an interpreted environment for our multi-agent model and distributed object structure. Using such an environment, it was easier to put into practice serialization and various execution control mechanisms [10] which contributed to the implementation of the *Omega* distributed objects system.

Omega presents a distributed object structure, and its initial goal was to determine some good representations of data, types, instructions, functions/procedures, and objects of an object-oriented language that can be used as a programming language for mobile agents. The consequence of this effort is a system written in C++ language, system that is able to combine the notions behind the object-actor duality, namely the duality between passive and active objects.

Presently, *Omega* system offers to active programs what the World-Wide Web space provides by default for presentation of some static entities (documents), i.e. an infrastructure able to support Web-based distributed applications (such as agents used in clusters or Grid [20]).

Omega *Classes*

The `IObject` class is the base-class for each other class that has memory sections stored within a local system. Each object and function that needs a store space in *Omega* will use the `IObject` class (see also Fig.1). In this way, the *Omega* system offers 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 system includes a number of object types which give functionality to the following classes: *String*, *Number*, *List*, and *Control* agent-execution (i.e. support for virtual threads or scripting languages). Certain data is represented by different classes such as `IString`, `INumber`, `IOmegaStack`, `IOmegaList` that are derived from the `IObject` basic class.

Omega offers two categories of data types [2]: *simple data types* – have no components (i.e. `INumber`, `IString`, etc.) – and *compound data types* – signify a mix-up of two or more simple types (e.g., `IName`, `IOmegaList`, `IAThread`).

***Omega* Language**

Omega offers an execution part which act as an implementation of a script-like language. In this active part, we are trying to assimilate 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),
- certain techniques to implement the high-level programming level statements.

The language provided by the *Omega* framework may be easily extended with other instructions to make a more complex computational model of the developed agents. An significant step was to create a mechanism for representing data structures, statements and objects under the same abstraction (`IObject`) that is a network shared entity.

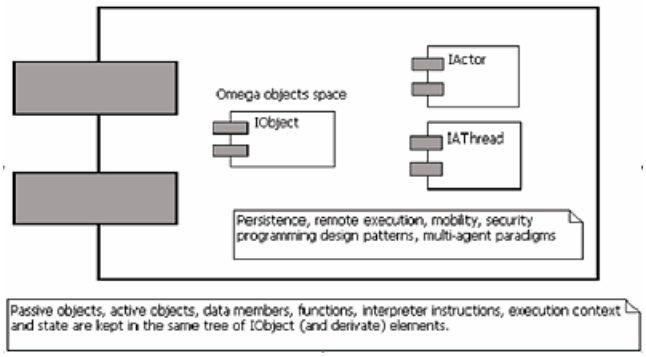

Fig.1 – *Omega* Objects

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 way, 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 certain object types (for example, `IOmegaSockets` used for socket operations) the serialization and deserialization actions can not be viewed as a suitable solution.

The object serialization does not imply the serialization of the entire 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 [1].

XML-based Serialization

As the best manner to serialize the *Omega* objects, we prefer an XML-based representation. To describe these objects, we could implement an RDF-based model. The RDF assertions could offer the possibility to express semantics of an *Omega* object.

We are using the XML namespaces defined by the XML Schema specification [13] to keep 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>
</IName>
```

SOAP-based Serialization

SOAP – or other protocols that use the RPC over XML approach (e.g., XML-RPC) – can be used to transfer the serialized data between the *Omega* software entities.

SOAP (*Simple Object Access Protocol*) [14, 23] is an effortless lightweight protocol used for structured and strong-type information exchange in a decentralized and 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 conventions 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 invoke methods, services, components, and objects on remote servers. On the other hand, unlike these protocols – which use binary formats for the calls –, SOAP utilizes XML to structure the nature of the data 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) [23]. It defines a reduced set of parameters that are specified in the HTTP header, making it easier to pass 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* [21], which is able to produce the code for serialization from a user-defined specification. The *gSOAP* compiler tools offer 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.

4. DESCRIBING *OMEGA* OBJECTS IN RDF

Resource Description Framework

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 proper to model objects behaviors.

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. Resources correspond to objects and properties correspond to instance variables [7, 18].

To ease the definition of metadata, RDF is based on *classes*. A collection of classes, usually designed for a specific purpose or domain, is called a *schema* [7]. Also, RDF supports the reusability of metadata definitions. The RDF schemas may themselves be expressed in RDF. Of course, the RDF syntax is an XML-based one.

The fundamental model of RDF consists of three object types [18]:

- **resources**

All objects being described by RDF expressions are called *resources* and they are denoted by *Uniform Resource Identifiers* (*URI*). Using URI standard schemas, each type of resource can be identified in the same manner.

- **properties**

A *property* is a particular 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.

- **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),
- *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. Presently, there are several proposals of model-theoretical semantics for RDF and RDF Schema [11, 17, 23].

Expressing the *Omega* Objects' State in RDF

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 keep 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 [8].

The system maintains these descriptions as RDF assertions that can be transported via SOAP 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 reject the access to considered object. The meta-descriptions regarding the permissions and the identity of the user who intend to access *Omega* objects must provide a certain cryptographic support.

Expressing Relations between *Omega* Entities

In order 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 considered. The temporal relationships between objects could be stored by RDF constructs, too. This approach is comparable to our model used to keep temporal relations established between (fragments of) Web sites [9].

The proposed model is primarily 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*) [9]. For each time relation, TRSL offers an element that corresponds to a particular relation (e.g. `<Meets>` element for *Meets* relation from ITL model). The beginning and ending of time periods are represented 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 full syntax and the semantics of TRSL language is presented in [9].

Example

An object-maintainer Web agent is build to discover different temporal relations between the *Omega* objects distributed in an intranet and can automatically produce the following TRSL assertions:

```
<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 Nov 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 the `RecentlyChanged` identifier. These objects are stored on `www.site.org` machine and the serialization action is planned to be performed on November 17, 2003 at a certain hour. 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 way.

5. RELATED WORK

The building process of the agent-oriented systems requires a different approach from that of conventional software systems development [5, 15].

We are aware of numerous platforms developed both in academia and software industry companies [19]. This fact confirms that many computer scientists are considering the agent-oriented software as a potential paradigm, designed and implemented especially in very dynamic environments (such as the Web space). Though, the existing implementations do not cover or provide certain services 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 existing open-source platforms.

The actual multi-agent platforms use diverse approaches for communication between agents, by using low-level communication protocols (e.g. TCP/IP or HTTP) or standard high-level languages – such as KQML (Knowledge Query Manipulation Language) [5].

The *Omega* system presents a certain advantage, by adopting an XML-based platform-independent approach in serialization and exchanging information between agents. The SOAP protocol is more flexible and easy to use than CORBA or DCOM solutions.

Some of the *Omega*'s facilities could be also integrated in the actual multi-agent systems, such as *MAIS* (*Mobile Agents Information System*) – a platform for creating dynamic clusters [16].

6. CONCLUSION

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

This paper focused on presenting the different platform-independent techniques of exchanging information between the entities of a multi-agent infrastructure. We proposed an XML/RDF-based model that can be used as a general manner for serialization and metadata description of the objects processed by the *Omega* agents. Various properties and relations established between the components (i.e. agents, objects, processes, etc.) of a multi-agent system can be expressed in a standardized and machine-understandable way. This advance provides semantic descriptions of the Web resources and could be considered as an attractive solution for exchanging knowledge between intelligent or/and mobile agents.

In addition, we plan to design and 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.

7. REFERENCES

[1] S. Alboaie, S. Buraga, L. Alboaie, An XML-based Serialization of Information Exchanged by Software Agents, Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics – SCI 2003, Orlando, Florida, 2003.

[2] S. Alboaie, G. Ciobanu, Designing and Developing Multi-Agent Systems, 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] J. Bradshow, Software Agents, AAAI Press,

1997.
[6] T. Bray et al. (eds.), Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, Boston, 2000: http://www.w3.org/TR/REC-xml
[7] 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-rdf-schema
[8] S. Buraga, A Model for Accessing Resources of the Distributed File Systems, in Advanced Environments, Tools and Applications for Cluster Computing, D. Grigoraş et al. (eds.), Lecture Notes in Computer Science – LNCS 2326, Springer-Verlag, 2002.
[9] S. Buraga, G. Ciobanu, A RDF-based Model for Expressing Spatio-Temporal Relations Between Web Sites, Proceedings of the 3rd International Conference on Web Information Systems Engineering (WISE 2002), IEEE Computer Society Press, 2002.
[10] C. Callsen, Open Distributed Heterogeneous Computing, PhD Thesis, University of Illinois at Urbana-Champaign, 1997.
[11] W. Conen, R. Klapsing, A Logical Interpretation of RDF, Linkoping Electronic Articles in Computer and Information Science, 5, 2000.
[12] 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
[13] D. Fallside (ed.), XML Schema Primer, W3C Recommendation, Boston, 2001: http://www.w3.org/TR/xmlschema-0/
[14] C. Gorman, Programming Web Services with SOAP, O'Reilly and Associates, 2001.
[15] S. Green, F. Somers, Software Agents: A Review: http://www.cs.tcd.ie/research_groups/aig/iag/iag.html
[16] D. Grigoraş et al., MAIS – The Mobile Agents Information System Support for Creating Dynamic Clusters, Proceedings of ICA3PP, Beijing, 2002.
[17] P. Hayes (ed.), RDF Semantics, W3C Working Draft, Boston, 2003: http://www.w3.org/TR/rdf-mt/
[18] O. Lassila, R. Swick (eds.), RDF Model and Syntax Specification, W3C Recommendation, Boston, 1999: http://www.w3.org/TR/REC-rdf-syntax/
[19] E. Mangina, Review of Software Products for Multi-Agent Systems, AgentLink.org, 2002: http://www.agentlink.org/
[20] 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, 2002.
*[21] * * *, SOAPware: http://www.soapware.org/*
*[22] * * *, Web Object Integration: http://www.objs.com/survey/web-object-integration.htm*
*[23] * * *, World Wide Consortium's Technical Reports, Boston, 2003: http://www.w3.org/TR/*

***Sabin-Corneliu Buraga**, MSc., is a lecturer at the Faculty of Computer Science, "A.I.Cuza" University of Iasi, Romania. Currently, he is a PhD candidate (in the final stage) at the same University and his areas of interests include Semantic Web, distributed computing, and Web technologies. He is the author of more than 30 articles published in the proceedings of different international conferences or in the scientific journals, published by Springer-Verlag, IEEE Computer Society Press or Elsevier. Over the years, he served as a member of the program committees or/and organizing committees of different international scientific events and summer schools. He is the initiator and editor of Web series of books published by Polirom Publishing House (Iasi, Romania).*

***Sinica Alboaie**, MSc., is a researcher at Institute of Theoretical Computer Science of the Romanian Academy - Iasi branch. He is doing research in distributed systems with agent technologies (main theme), cellular automata and applications in software design-patterns, programming languages. During the years, he also gain a vaste experience in designing software components for handheld devices.*

***Lenuta Alboaie**, MSc., is an assistant researcher at Institute of Theoretical Computer Science of the Romanian Academy - Iasi branch. Currently, she is a PhD student and her areas of interest cover certain aspects of distributed computing, such as agent-oriented, peer-to-peer and Grid computing. She also is interested on Web services, especially on XML-based protocols (such as SOAP). She is a member of the WebGroup - an young research group on Web technologies at Faculty of Computer Science, "A.I.Cuza" University of Iasi.*