A Review of Interoperability Techniques for Models, Data, and Knowledge in Environmental Software

A. Jolma[a] and A. Rizzoli[b]

[a]Helsinki University of Technology, P.O.Box 5200, 02015 HUT, Finland (ajolma@water.hut.fi).

[b]IDSIA, 6928 Manno, Switzerland (andrea@idsia.ch).

Abstract: An analysis of large and complex systems such as environmental systems linked to socio-economic systems usually requires several simulation models. These simulation models must be able to interface with each other in the conceptual level but there may also be some overlap in their application domains. The way simulation models are used normally generates large amounts of data, which need to be explored or mined for the analysis and possible decision process. We propose a set of knowledge-based agents, which share a common ontology to aid in the modeling and analysis in large and complex systems. An agent is an active software entity, which is designed for a certain task or tasks. The main requirements for such knowledge-based agents include ontology representation and storage, communication capabilities, and data inspection tools. For example an agent should be able to actively study, e.g., a hard disk volume for available information, discuss the findings with a knowledgeable person, and then produce and communicate a report. The paper discusses the theoretical, technological, and other implications of the proposed approach versus other paradigms. In particular we are interested in using XML as the main tool for ontology representation and in supporting more than one level of conceptualization of the problem domain. Some technologies, notably application servers, also implement some services, which could be utilized for ontology sharing and agent communication. Benefits and constraints of these are discussed.

Keywords: *Interoperability, Ontology, Agents, Environmental Information Systems*

1. INTRODUCTION

Examining any big environmental problem requires cooperation of experts and interdisciplinary analysis. Three main groups of experts, whose contribution and expertise must be amalgamated, are physicists, engineers, and social scientists, including economists. Geophysicists usually contribute by developing simulation models of various parts of the physical system. Engineers provide management models and technological solutions. Socio-economists contribute by information and analysis on how various people behave or will react to changes. Due to the complexity of the problem domain and multiple actors the information management of such projects faces several problems. These problems can usually be categorized as interoperability problems.

Interoperability is a technical term but the problem is not only technical. Without interoperability, integration is possible, but cumbersome, and its effectiveness is sensibly decreased, as integration is an essential element of the problem solving. Modern environmental assessment procedures and methodologies call for an ever more intensive use of integration (see for example the proceedings of the recent conference iEMSs 2002 http://iemss.org/iemss2002). Integration is the keyword and it must be accomplished across disciplines, across temporal and spatial domains, to deepen our understanding of the multifaceted complexity of environmental systems.

At the heart of interoperability problems lay the different ways to organize information and knowledge in different disciplines. It is thus natural to turn to generic research on information and knowledge for some answers. According to Guarino (1995) AI research has had two distinct perspectives on knowledge. The first, traditional approach is to view knowledge as functional utility for reasoning. In the language of geophysicists this means that the focus is on simulation model building. The second view of AI research sees knowledge having task-independent value. The task-independent value of knowledge is enhanced when parts of knowledge are organized into distinct conceptualizations. These conceptualizations are called ontologies and they have according to Guarino also potential in large-scale integration.

A lot of philosophical and AI research was largely theoretical considerations without real-world applications. Evolution of computing, especially networking, has lead to fruitful exchange of research results and development of new computing paradigms like agent-based computing and the Semantic Web. Agent-based computing has been

proposed as a suitable solution for problems, which require active and "smart" modules communicating with high-level messages (Nwana and Ndumu 1999). The Semantic Web (Berners-Lee et al 2001) is a vision for the World Wide Web, which builds on XML-based tools for ontologies.

Nwana and Ndumu (1999) have listed problems in distributed system development, which have motivated research on agent-based systems:

- The problem of information discovery: e.g., a description of a model tells me I need this kind of data, where do I find it?
- The communication problem: e.g., how to extract the needed information from the data stream coming from a source?
- The ontology problem: e.g., how do we know that "precip" at one message is the same as "precipitation" in another message?
- The legacy software problem: e.g., how to extract information from a binary file of unknown format created by legacy software?
- The reasoning and coordination problem: e.g., how to get from the functional descriptions of the problems to the actual tasks and their scheduling and coordination between subsystems?

In this paper we investigate the possibility and benefits of the merger of the more traditional interoperability technology with the results of AI research. This paper is organized as follows. In section two we investigate the issue of technical interoperability. Section three examines one solution to the interoperability problem in more detail, namely XML-based technology. Section four describes ontologies and agents and examines how agents could be the glue to keep the various interoperable components together. Section five is the discussion of the findings of this paper.

2. INTEROPERABILITY

2.1. Introduction

Interoperability is defined here as the ability (of the user) to use more than one system together as one. This includes requesting and receiving services from one system and utilizing the results in requesting and receiving services from another system. Interoperability is not a new concept; towards the end of the 80s the lack of it was a common curse of a system administrator managing a wide and heterogeneous network of computers, where the departments of a same organization used different operating systems, software tools, and databases.

Interoperability relies on requesting and receiving services. A service can be conceptualized as an exchange of resources. An example of a resource is a description. In environmental information systems descriptions are: (i) descriptions of the problems, (ii) descriptions of the environmental system, (iii) descriptions of the models or other tools, (iv) descriptions of the observation data, and (v) descriptions of general knowledge. Other resources include (virtual) machines capable of processing these descriptions and other generic tools.

At first, the focus on interoperability research was on the ability of making different operating systems communicate in a networked environment. This technical problem of interoperability was solved thanks to the widespread adoption of shared protocols for communication networks, such as TCP/IP, on which the Internet is based. Yet, interoperability was out of immediate reach, since seamless data exchange was still prevented by syntactical problems: a document, stored on a UNIX based workstation, could not be accessed from a PC-based computer unless appropriate preprocessing was performed. Different research groups set to work to overcome this limitation and the results can be found in the software packages we now use every day. Among the various products which overcame syntactic interoperability we mention: ODBC, CORBA, and XML.

ODBC, the Open Database Connectivity is a widely accepted Application Programming Interface[1], developed by Microsoft, that allows uniform access to a wide range of database systems. It inserts a layer between the client application and the database; this layer hides the details of the database behind a general and published abstraction level. ODBC uses SQL[2] as an enabling technology to communicate with the data source. Specific drivers are designed for most database systems. The ODBC API has been ported to various operating systems and can be used within many programming languages, thus supporting real interoperability.

CORBA, the Common Object Request Broker Architecture by the Object Management Group is a software architecture and infrastructure that allows computer applications, running on different machines, under different operating systems

[1] ODBC specification and other material is available by anonymous ftp from ftp://ftp.microsoft.com/developr/ODBC

[2] SQL is Structured Query Language, a language for communicating with a relational database. Several SQL and SQL related standards exist.

to work together on a network[3]. CORBA has been the first effort to go beyond the notion of 'remote procedure call', which we first find in UNIX networked systems. In CORBA the programmer wraps his/her application publishing its interface thanks to the IDL (Interface Description Language), which is defined generically but implemented specifically for every language which supports CORBA. Such a wrapped application can then announce its services on a network. Other applications which know about this published interface can therefore use the services provided by the remote application. It is the Object Request Broker that manages the requests coming from client applications and directs them to the servers. Microsoft developed their own flavor of CORBA, first in a local environment thanks to the COM (Component Object Model) architecture, then distributing it in a homogeneous (all Windows) network (the DCOM, for Distributed COM) and finally making the step to open up the architecture to other operating systems with the .Net architecture. Sun Microsystems's Java language did practically the same implementing the Java RMI (Remote Method Invocation) architecture.

CORBA and similar approaches have not yet reached the widespread adoption and popularity of ODBC. The most plausible reasons are the computational burden, the difficulty of setting up a distributed network of transparent applications, the slowness of the implemented code, and possibly the lack for the real need of distributed computing, but we will come later on this last point.

Finally, XML, the eXtensible Markup Language, developed and promoted by the World Wide Web Consortium[4], W3C, addresses interoperability at the document level. XML is a derivative of SGML (Structured General Markup Language). XML was introduced as a tool for interoperability between a Web client, Web server, and databases; furthermore, user interface issues were high on the list of requirements (Bosak, 1997). While CORBA and its companions allows one to remotely invoke an application, it does not say anything on the data format requested from the remote application and therefore a way to specify this was needed and XML is the answer.

[3] CORBA specification and other material is available from http://www.omg.org/technology/

documents/corba_spec_catalog.htm

[4] XML specification and other material is available from http://www.w3.org/XML/

3. XML TECHNOLOGY

In the following several XML technologies are shortly described. The description is not technical but tries to describe the technology in the context of this paper.

From the point of view of a programmer an XML document is either a stream of events or a tree. The stream of events approach is faster since it can be implemented along with parsing and it requires less memory. The tree view is available only after parsing an XML document, it is more versatile but also requires more memory for the data structure. There is a standardized API called Document Object Model (DOM) for the tree view of an XML document. DOM has been implemented in many programming languages.

XML Namespaces is a method of declaring (in an XML document) that certain element types and attributes should be interpreted in a specified context or sense. An XML namespace thus creates a vocabulary and more than one vocabularies can be safely mixed in one document which is a strong feature supporting interoperability. The specification is vague but seems to work in practice (Bourret 2000). Namespaces makes it possible to introduce and use crude ontologies in XML.

There are several technologies for specifying the schema, i.e., grammar, of XML documents. The best known is DTD (from SGML origin), XML Schema (from W3C), and Relax NG (from OASIS). The schema does not imply any semantics (as a namespace may) but it has great practical value. A schema is of value to a user creating documents since s/he can be given interactive help about what is expected/valid at any given point in the document. A schema is of value to programmers since it greatly reduces the need of writing assertions.

XML Path Language is an expression language for referring to certain parts of an XML document. XPath treats an XML document as a tree of nodes and defines a way for expressing each node as a string. A string expression in XPath can expand to several nodes. XPath is a relative to regular expressions which are used for text processing, e.g., for automated conversions from ASCII data files to database insertions.

XSL Transformations is a part of XML Style sheet Language (XSL). The other parts are XPath and a specific vocabulary for specifying formatting semantics. XSL is originally intended for XML → HTML transformations, i.e., for visualizing XML documents, but it can be used as a general tool for XML → XML transformations. XML → XML transformations have potentially great

value in transforming structured documents from one ontology or application to another.

The Resource Description Framework (RDF) is a foundation for processing metadata. As metadata is data about data, the RDF model is about saying that a certain thing (resource, concept, or object) has a certain property, and that the value of the property is something (another resource, concept or object). Thus basic RDF statements are triples of subject, predicate, and object. RDF is a framework and only provides the basic vocabulary for (XML) metadata documents. The semantics of the metadata must come from some other source, the Dublin Core, for example.

The Web Ontology Language (OWL) is on-going work of W3C aimed at producing an XML and RDF based language for ontologies. OWL is based on the DARPA Agent Markup Language (DAML+OIL). OIL is Ontology Inference Layer developed separately.

A simple example of a RDF description (triple), which utilizes property from DAML vocabulary, is:

```
<rdf:Description rdf:about="#sadanta">
<daml:samePropertyAs
rdf:resource="http://hydrology.org/voca
bulary/precipitation"/>
</rdf:Description>
```

This description tells us that the local concept "sadanta" (precipitation in Finnish), is the same as the concept "precipitation" in the vocabulary of some (hypothetical) international organizations. Given this description any RDF aware software which handles local documents and documents – or services – conforming to the international standard would have no problems in doing the right thing in this respect. This is an example where the RDF and OWL go beyond schema languages (Relax NG, XML Schema).

Knowledge stored as RDF statements and OWL object structures support classification by inference. Classification by inference is based on object's membership in classes (in sets in fact) which have certain properties, for example if we know that Kirkkojärvi is a humic lake, and humic lakes have low Secchi disk depth, then we can infer that Kirkkojärvi has a low Secchi disk depth. This works also the other way. If we have two pieces of information: "humic lakes have low Secchi disk depth" and "Kirkkojärvi has a low Secchi disk depth", the first one in our knowledge base and the other in our database, we could query "what are the potential humic lakes described in the database".

4. ONTOLOGIES AND AGENTS

4.1. Ontologies

Ontology is a philosophical discipline, a study of things that exist a-priori; a particular system of categories accounting for a certain vision of the world; or an engineering artifact, constituted by a specific vocabulary, used to describe a certain reality, and the intended meaning of the vocabulary words (Guarino 1998). We shall stick to the last use of the word in this paper.

Ontology as an engineering artifact, devised as a part of IS development, is a generic concept, which covers data models of database engineering, object models of software engineering, descriptions of simulation models of systems engineering, and knowledge models of knowledge engineering. Ontologies are agreements, something to commit to. A large part of our knowledge can be expressed as ontologies; the rest is rules how to use and reason with the things described by ontologies.

Guarino and Welty (2000) have proposed four notions as the basis of formal methodology for ontology engineering: identity, unity, rigidity, and dependence. Identity of a thing is a property by which it can be distinguished from other similar things. The interesting thing about identity is that physical entities have a stronger identity (Lake Windermere is always Lake Windermere no matter what – at least to a certain point) than roles for example (the management board of Lake Windermere changes from time to time). Guarino and Welty call properties (like being a lake), which hold across ontologies 'rigid'. A non-rigid property is something, which is assumed based on situation ("which hat does the person hold now?"). Unity is a concept, which holds things, made of parts together and defines a thing as a sum of its properties. The unity of a thing may change when we move from one ontology to another even if the identity stays the same. A typical example is how differently people with different backgrounds see the same lake. Dependence is according to Guarino and Welty a very general meta-property of things. One form of dependence is required properties. For example in one ontology a body of surface water is not a lake unless it is at least 200 meters wide in at least one direction. Different dependencies in different ontologies create great interoperability problems.

4.2. Agents

An agent is an individual capable of decisions and actions within an environment, which it can observe to some extent. An agent should also be able to socially interact and communicate with

other agents (Wooldridge and Jennings 1995). From the software engineering point of view agent-based programming can be seen as a new paradigm (Wooldridge and Jennings 1999).

A software agent is a program or an object created by a program. In the first case the environment of the agent is "real" since the OS provides it and – if the computer is networked – the OS of other computers and devices connected to the network. In the second case the environment of the agent is simulated (not "real"). The concept of an agent is interesting since by definition every one of us is an agent. Thus the results of psychological, social, and decision analysis research – to name just a few – are all applicable.

Agents are intentional systems (Wooldridge and Jennings 1995), which operate on information (beliefs or knowledge) driven by a pro-attitude (intention, obligation, goal, etc). The pro-attitude can be expressed as utility functions. Resources available to them and actions possible to them limit what agents can do. Agents typically also have a notion of cognitive state.

A software agent can be designed by defining a utility function for it. The utility function is a real number valued function of actions and observations. The agent is designed in such a way that it tries to maximize its cumulative utility. This is in contrast to traditional programs which are designed for a task or tasks and do not have the concept of utility. This traditional behavior of programs is a special case for an agent program; the programmer or the user of an agent program needs just to define the outcome of a task, which then becomes the goal of the agent, and then provide the agent the necessary knowledge about the actions that will lead to the finalization of the task.

5. DISCUSSION

We can observe three computationally different types of ways to achieve interoperability: (i) requesting data or documents, (ii) exploiting the computational resources of a network node through a published interface, and (iii) sending executable instructions of some form from one network node to another.

In all of these interoperability scenarios we have an information exchange, which is based on the assumption that once the syntactic translation is made, the semantics comes without saying. This was the case in monolithic applications, where the modeler built an application from top to bottom, using internal components, or components which had been re-engineered to be used within the application. In distributed, multi-tiered applications, which are required in integrated modelling, this is not valid anymore. The modeller cannot go on and link her model outputs to a remote model inputs without making sure that the *meaning* of her outputs corresponds to the meaning of the remote model's inputs. Such an assertion can be made by close inspection of the remote source code, by reading the remote model documentation but these are time consuming and difficult tasks.

The current interoperability problem arises from the differences and incompatibilities between the ontologies of different systems. The problem is thus on the semantic level of the modelling environment. The paper of Kokkonen et al (2001) describes an approach and a technical solution to the problem linking a database to a model. The semantic level of the linking relies solely on the knowledge of a human user. It is easy to see that knowledge encoded in RDF descriptions as above can provide at least a partial solution to this problem. The next generation of interoperable applications will need to address the issue of semantics, that is the ability to represent knowledge in a structured and re-usable way.

5.1. Are there feasible solutions in sight?

As quickly presented in Section 2.1, ODBC, CORBA, and XML are technological solutions to various interoperability needs. All of them define or employ formal or semiformal languages for interfaces and communication. It is perfectly feasible to interlink various data sources, models, and other applications with these tools. The syntactic level of these linkages is usually nothing but a technical problem, easily overcome by program code. There is also little or no distinction between the case where the requested service is running in the same computer and the case where it is running elsewhere in the Internet.

Yet they do not address the semantics problem.

XML is the general background in most current efforts in putting the results of theoretical research in knowledge engineering into practice. Previously there have been some notable failures in similar efforts (For example the case of Telescript and General Magic, (Magdanz et al, 1997)) but currently this does not seem to be the case for XML. In one sense the XML technology represent a shift from programming to writing documents. For example XSL documents can be seen as documents, which describe how documents are changed.

This helps a lot in content management, but what about semantics?

As shown in Sections 3 and 4, the answer to this can be find in the efforts to create shared ontologies. Ontologies are services or metadata associ-

ated with services, which are needed to manage and organize knowledge. They do help in establishing the semantic relationships among the objects we want to model.

The open issues, for environmental modelling, are: (i) creation of shared ontologies for the various fields of environmental modelling; (ii) creation of software tools able to access and use the ontologies and to provide factual help in building distributed environmental information systems, including environmental models.

The former issue calls for the co-operation of a wide group of research networks and, while no co-ordinate approach exists up to date, we expect to see the first efforts towards the end of 2003.

The latter issue is the more interesting for the software engineer and the developers. We have proposed in Section 4 an approach based on agents, in accordance with the path shown by Berners-Lee and others in their seminal work (Berners-Lee et al., 2001).

In our view, an agent needs the ontology when it fulfills its tasks of matching data with a simulation model, using a solution template for a problem, or interprets a goal given to it by a user or another agent. Ontologies are engineering artifacts but agents can also help in writing them, since a lot of useful knowledge can be downloaded or reasoned from the Internet.

Above an EIS was described as a system, which consists of subsystems making and serving requests. The agent-based paradigm, which is based on communication and goal-orientation, should fit rather well to this. Two classes of agents emerge: (i) system-level agents, which communicate with users and service providers, and (ii) service provider agents, which communicate with system-level agents and try to fulfill requests by the available resources.

In conclusion, we believe that future environmental information systems will need to address the issue of semantic interoperability. This will require an explicit notion of ontologies and mappings between ontologies.

6. ACKNOWLEDGEMENTS

This research has been partly funded by grants from Maa- ja vesitekniikan Tuki ry. and Ministry of Agriculture and Forestry of Finland to the first author.

7. REFERENCES

Berners-Lee, T., Hendler, J., Lassila, O. 2001.The Semantic Web. *Scientific American*, May 2001.

Bosak, J. 1997. XML, Java, and the future of the Web. http://www.ibiblio.org/pub/sun-info/standards/xml/why/xmlapps.htm

Bourret, R. 2000. Namespace myths exploded. http://www.xml.com/pub/a/2000/03/08/namespaces/index.html

Guarino, N. 1995. Formal ontology, conceptual analysis and knowledge representation. *International Journal of Human and Computer Studies*. **43**(5/6): 907–928.

Guarino, N. 1998. Formal Ontology and Information Systems. In N. Guarino (ed.) Formal Ontology in Information Systems. Proceedings of FOIS'98, Trento, Italy, 6–8 June 1998. IOS Press, Amsterdam: 3–15.

Guarino, N. and Welty, C. 2000. Towards a methodology for ontology-based model engineering. In Proceedings of ECOOP-2000 Workshop on Model Engineering. Cannes, France.

Kokkonen, T., Jolma, A. and Koivusalo, H. 2001. Interfacing Environmental Simulation Models with Databases using XML. MODSIM 2001, Proceedings of the International Congress on Modelling and Simulation, Canberra, Australia, 10-13.12.2001. pp. 1643-1648.

Magdanz, E., Rau, N., Bernstein, N. 1997. Strategic Computing and Communications Technology. Group H: Standards. Final Report. http://www-inst.eecs.berkeley.edu/~eecsba1/s97/reports/eecsba1h/paperTOC.html

Nwana, H. S. and Ndumu, D. T. 1999. A Perspective on Software Agents Research. *The Knowledge Eng. Rev.* **14**(2): 125–142.

Wooldridge, M. and Jennings, N.R. 1995. Intelligent Agents: Theory and Practice. *The Knowledge Eng. Rev.* **10**(2): 115–152.

Wooldridge, M. and Jennings, N.R. 1999. Software Engineering with Agents: Pitfalls and Pratfalls. *IEEE Internet Computing*. **3**(3): 20–27.