IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com
ITP4721

Managing the Interconnection and the Distribution of Urban Models with XML and Distributed Objects

Alain Becam[1], Maryvonne Miquel[2], Robert Laurini[3]
INSA of Lyon – LIRIS (ex-LISI),
20, avenue Albert Einstein
F69621 Villeurbanne Cedex
[1]abecam@lisi.insa-lyon.fr, [2]miquel@if.insa-lyon.fr,
[3]laurini@lisi.insa-lyon.fr

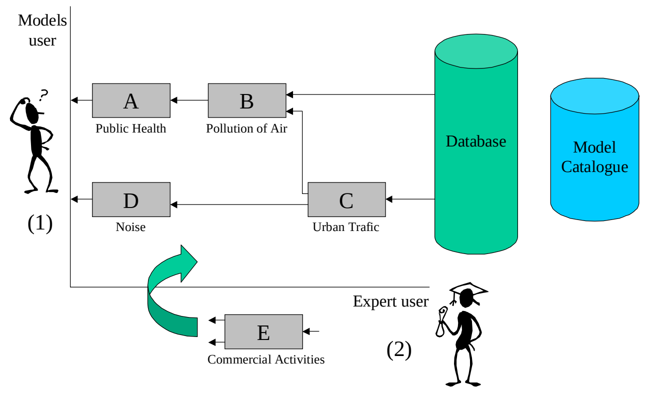
ABSTRACT

Our project deals with the reuse of existent urban models into an interconnection schema. An urban model is a specialised tool for simulating or estimating phenomena about the city. Using a specific technique of encapsulation, with a semi-automatic generation of communication drivers, the proposed environment offers a homogeneous view of the urban models, which may be used into a distributed environment. In this paper, we present the global distributed architecture, created with Java, and the chosen representation of the interconnection schema, explaining how this particular approach is well dedicated to manage the distributed urban models. This paper shows the advantage offered by the XML syntax to represent meta-data and specific data, and by a strong hierarchical management.

I. INTEGRATION OF URBAN MODELS

Urban models are specialised software tools, responding to a specific modelling or simulating goal. They simulate specific phenomena of the city and are a reliable decision-support for urban planners and decision-makers. The urban models are used for simulation and for estimation, they are in general finished products, used by experts and which answer to the pursued objectives. However, there is no standardisation of the models and no standard formats for their inputs and their outputs.

Figure 1*: A schematic representation of our goal: (1) to allow the standard use of different models and their interconnection for an user, and (2) to offer to an expert the possibility to integrate an existing model into the environment*



It may be interesting to inter-connect several models to profit from their specialities and thus to allow new experiments. For example, it is mainly a question of using the result of a model for the execution of another.

An interconnection of models defines a new complex model using existing models following a graph of interconnections (Figure 1).

The existing urban models are not developed to be interconnected, because they are finished software packages. Generally, users execute urban model without knowing exactly how it works. In contrast, to integrate a model into a global environment, it must be well known, in relevance with the domain, here the urban universe, and it must be homogeneous.

We can distinguish different of urban models:

- the class, which are models constructed like a class or like an API and easily usable,
- the procedures that may be controlled with batch files, through OLE/DCOM or CORBA, through sockets, ... , these models are not easily usable in any context but are built to be used,
- the complete programs, which are not built to be interconnected or used by an automatic way. These programs are very difficult to integrate.

The crystal box approach and the black box approach are two specific ways to deal with the integration of these models. The **crystal box** solution is based on a high level definition of the model, sufficient to design the model. This definition may be viewed as a program in a high level language or as a meta-model, a meta-data on the model.

The **black box** approach considers that the models exist and that the way to process is not or badly known. It is possible to federate the models or encapsulate them. Into a federation (DMSO, 97), the model must have a standard interface and is only integrated by the compliance to the interface syntax, just like the object into CORBA (Object Management Group, 01), which have to define an IDL. But this solution implies to write or to rewrite the models to conform to this syntax.

Our final objective is to provide a workshop of composition of models. Thus, a "final" user will be able to use models of the workshop and to connect them between them to test. On the other hand, an expert user should also be able to integrate one or more models in the workshop.

To use existing models without changing them, we may encapsulate them. The encapsulation controls the model and is used as a homogeneous model (Becam, 00). An architecture allowing the interoperability of distributed models was proposed in (Becam, 01). In this paper, we focus on the interconnection of distributed models in order to define new super-models. We propose meta-models with a logical view of models allowing the management of the interconnection schema. We first present the different works about software reuse and components technologies in section 2. Section 3 explains how to integrate the urban models in a unified environment using XML. Section 4 proposes solutions for the interconnection management. We conclude in section 5.

2. SOFTWARE REUSE AND DISTRIBUTED COMPONENTS

The need of software reuse comes from the rapid evolution of the computer universe. A lot of powerful software tools exist and it is inefficient and difficult to re-write everything anytime. So, designers want to create only the original part of their modelling tools, using the existent tools to deal with the other problems.

The first application of this idea is the shared libraries used into the modern operating systems. They deal with specific problems and may be used by several programs simultaneously. Libraries are written to this specific usage and give the possibility to write smaller application by avoiding the rewriting of each widely used procedure.

Recently, the different systems have integrated the possibility of collaboration between different applications, by OLE/DCOM, CORBA or EJB. Into CORBA (Object Management Group, 01), the software pieces of code are written respecting a defined interface syntax. They are usable together, following the designed possibilities. With OLE/DCOM (Microsoft, 00), the integration is at a binary level and allows the use of one application into another. The best-known use of OLE/DCOM concerns the Microsoft Office, where it is possible to integrate a document from one application into the document of another application and to run the native application by selecting the document.

The Enterprise JavaBeans (EJB) (Sun Microsystems, 01) of Sun Microsystems manages persistent distributed objects written in Java and using Java RMI. JINI (Sun Microsystems, 02) is an integrated solution for management of distributed objects. Because CORBA is well supported by Java, this last environment is not a closed world.

These different solutions are powerful and efficient. But they need elements specifically written for this meaning.

Some systems allow to integrate existing software components, like the DOMIS project (MITRE, 00), which studies the use of OMG technologies for the reuse of existing components, or like ACE (Schmidt, 02), which offers an environment written in C++ to integrate, via several patterns, existing programs with few rewrite effort.

Recently, some systems have been created to deal with the intelligent integration of heterogeneous components. We may name Protégé (Grosso, 99), which helps users to build tools using knowledge acquisition, or IBROW3 (Benjamins, 98), which is an intelligent brokering service on the web, using knowledge components, in fact PSM (problem-solving methods). A web-based system for the reuse of urban models has also been proposed in (Iazeolla, 98).

Finally, the focus of some projects is to add meta-data, like MOF (DSTC, 00), which is adapted to CORBA, or like XMI (Object Management Group, 99), which is a format of exchange. We may also speak about SOAP [W3C 01a], which encapsulates some data into a message in XML [W3C 01b], explaining the possible treatment of these data. The meta-model may be used to dynamically create the model (Maxwell, 98).

Into our project, we first want to standardize the models to allow their use as components into a network.

3. IMPLEMENTING META-MODELS USING XML

3.1 Encapsulation of Models

In order to control some heterogeneous heavy pieces of code, we propose to use these elements just like software components.

Using Java Technology, an encapsulation is used to abstract the model into a virtual standard object. In order to be usable like an object, an encapsulation is composed of three specific parts:

- a *driver*, which accesses to the urban model considering the element is a peripheral. The driver is composed of a generic Java part linked to a specific part that is able to access the element. The driver is the atomic interface between the raw model and the environment. There is always one driver for one model,
- a *spooler*, which controls, if needed, the accesses to the driver and uses a queue to allow the concurrence. The spooler allows a virtual simultaneous use of the element, but we still have to deal with the possibilities of jam. For this problem, the supervisor must verify the execution of each encapsulation.
- a *communication module*, which is connected to the environment using Java RMI, homogeneous, and usable like an object.

The goal of the encapsulation is to give locally a Java object, which is well integrated into the environment written in Java and which corresponds exactly with the Raw Model.

These objects are then integrated into a distributed environment using Java RMI, managed by different servers. A unique global server manages the whole environment. Local servers exist on every computer of the environment, and manage only their computer and its elements.

This meta-model dedicate the encapsulation to the urban model, indicating the good driver and giving its parameters. In order to describe the urban models, a meta-model is written in XML [W3C 01b]. A part of this meta-model is centralized to the environment to enrich a catalogue of urban models.

We adopt the XML syntax for three main reasons: the clarity of XML, the possibilities of process offered by the "tag" approach and the easiness to document the model following this syntax.

3. 2 The Meta-models

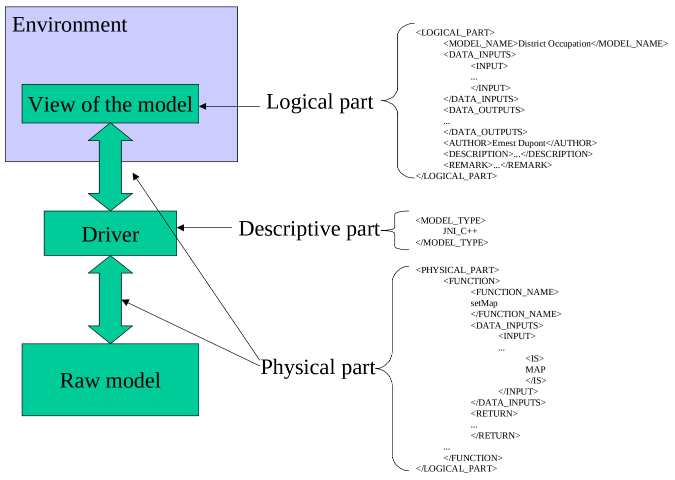
The meta-model is composed of three parts: logical, descriptive, physical.

The logical part represents the logical view of the model. The logical part gives the name of the model, describes the different inputs, outputs and parameters of the model, giving a typological description of the data, and a possible semantic description. The logical part also contains a user description of the model and some miscellaneous information.

The descriptive part of the model is used to choose a relevant driver according to the type of the model. The local server builds the driver using the name given in the descriptive part. A single class with exactly the name given by the "model_type" must exist (Figure 2). This class is retrieved and instantiated using the reflection capabilities of Java.

The physical part depends on the nature of the raw model. It describes how the driver has to work and how the allowed accesses are linked with the logical inputs, outputs and parameters described in the logical part.

Figure 2*: The role of the different parts of the meta-model*



```
<LOGICAL_PART>
        <MODEL_NAME>District Occupation</MODEL_NAME>
        <DATA_INPUTS>
                <INPUT>
                ...
                </INPUT>
        </DATA_INPUTS>
        <DATA_OUTPUTS>
        ...
        </DATA_OUTPUTS>
        <AUTHOR>Ernest Dupont</AUTHOR>
        <DESCRIPTION>...</DESCRIPTION>
        <REMARK>...</REMARK>
</LOGICAL_PART>
```

```
<MODEL_TYPE>
        JNI_C++
</MODEL_TYPE>
```

```
<PHYSICAL_PART>
        <FUNCTION>
                <FUNCTION_NAME>
                setMap
                </FUNCTION_NAME>
                <DATA_INPUTS>
                        <INPUT>
                        ...
                                <IS>
                                MAP
                                </IS>
                        </INPUT>
                </DATA_INPUTS>
                <RETURN>
                ...
                </RETURN>
        ...
        </FUNCTION>
</LOGICAL_PART>
```

We choose to use a simple hierarchical view to offer human-readable meta-files but also to have logical descriptions. Our meta-files are simple trees.

3. 2 The Catalogue

Before its use, a model is integrated into the environment, the local server checks the validity of the meta-model, and sends the logical part to the global server. When the global server receives a logical part from a local server, it enriches the catalogue by adding this logical part if the same model does not already exist on another computer and by always adding the network information about the distant computer.

At the beginning of a session, the local server calls the driver using the model type described into the meta-model and gives the physical part to the driver, just by taking the part of the meta-model delimited by the "physical_part" tags. The physical part always stays into the original meta-model file, on the computer running the raw model.

The XML syntax allows a very simple communication by the "mark-up" characteristic of the language. We always take some entire parts of a meta-file by isolating the chunk between specific tags.

4. INTERCONNECTION MANAGEMENT

4. 1 The Logical Schema

The logical schema indicates the interconnections between the different used elements of the environment. These interconnections are only at a logical level. The element of this schema may be viewed as asked instances of the different elements of the environment. When used, the different needed elements are instantiated by the local servers or by the global server, for the specific elements.

The logical schema is composed of:

- the list of the logical models used, which contains only the different logical parts of these models,
- the list of the virtual models of the schema: the different instances of the logical models, the specific models, the GUI models and the "dummy" models.
- the list of interconnections, described by a link between an output and an input.

In the logical schema, the supervisor verifies that every input, output or parameter is linked with another. An input, output or parameter may be "needed", "prohibited" or "optional". A prohibited input, output or parameter cannot be linked with another. After this simple check, the supervisor verifies the consistency of the links following the typological description and the semantic description.

Then, the logical schema is used to create a session. Using the list of logical models and the catalogue of models, the global server commands the creation of the needed encapsulations to the corresponding local servers. If a local server does not respond, the supervisor tries to choose another computer, if the needed model is present on different computers. When each encapsulation is created, the supervisor creates its particular models and begins the execution of the session.

4.2 Structure Definition

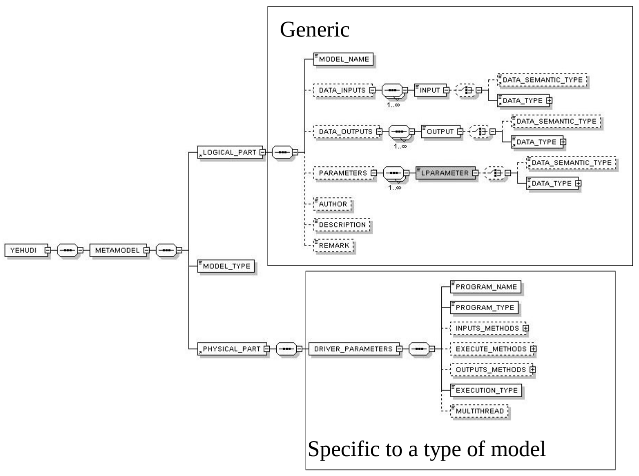
The different meta-data files of the proposed environment are strongly interdependent. In particular, the logical part is used into the meta-model, the catalogue and the different schemas. We use DTD (Document Type Definition) in order to define the structure of our XML document and to easily describe how our different files are defined.

The logical part is described into only one DTD and the meta-model, the catalogue and the schema just include one logical part in their structures. The structure definition of a meta-model is schematised into figure 3.

4. 3 Use of Meta-files

The supervisor uses the logical schema to command the different elements, encapsulations and specific models. The default strategy of execution can be described as follow:

***Figure 3**: The structure of the meta-model. The logical part is generic but may be extended and the physical part depends on the type of the model.*



"On start or after an event:

Step 1. Find each data available at output of models then send them to the models linked.
Step 2. Find each data available at input of models.
Step 3. Search for each model executable: has this model every data needed?
Step 4. Execute each executable model."

The supervisor updates the logical schema when it receives a message. If this message permits a new action, then the supervisor executes this sequence. Using an internal timer, the supervisor is also able to treat the schema during the session. The configuration of a session gives a default execution time for each model. The supervisor uses this default execution time or, if given, the estimation contained into the meta-model, to verify the good execution of a model. Without any message after this execution time, the supervisor considers there is a problem. This estimation may be updated during the session. And the logical schema is always updated, with the time of execution, the existence and position of data...

The supervisor also tries to estimate the quality of data, using the meta-model information, the duration of the session and builds a summary of the session.

The proposed environment is already able to run several distributed models following a logical schema, and using a catalogue. The core architecture is developed, integrating the java part of the encapsulations, the different servers and the main specific models.

We also develop a graphical user interface, offering an integrated edition and execution of the schema, allowing a centralized management of the distributed models. This interface will offer a strong interactivity.

5. CONCLUSION

In our project, we propose a realistic solution to integrate several heterogeneous urban models. This work introduces a catalogue of models and allows its use in a network domain. This work is a first step toward an intelligent system for the models reuse. Now our system integrates a software integration of models with a good logical description, using meta-models, and following a schema of interconnection. In perspective, we will try to extend our system to offer more automations and intelligence. It is clear that the global use of models needs a good knowledge about them, following a well-accepted reference, the ontology, to give the model its common sense for each user. This project will evolve to take care of these considerations.

REFERENCES

Becam, A., Miquel, M., Laurini, R., (September 2000) "A distributed environment using ontology for the interoperability of urban data and models, In **Geographical Domain & Geographical Information systems-EuroConference on the Ontology and Epistemology for Spatial Data Standards,** GeoInfo 19, La Londe les Maures, France, 11-15.

Becam, A., Miquel, M., Laurini, R., (2001) "Yehudi: An orchestrated system for the interoperability of urban data and models," in **Proceedings of ISCA, PDCS 2001, 14th International Conference on Parallel and Distributed Computing Systems August,** Dallas, Texas, USA, 8-10.

Benjamins, V. R., Plaza, E., Motta, E., Fensel, D., Studer, R., Wielinga, B., Schreiber, G. and Zdrahal, Z., (1998) "IBROW3 - An intelligent brokering service for knowledge-component reuse on the World Wide Web. In **Proceedings of KAW'98.**

MORS Modeling and Simulation HLA Tutorial (June 9, 1997) http://hla.dmso.mil/papers/mors.html.

The Distributed Systems Technology Centre (DSTC), (2000), **Meta-Object Facility Information (MOF)** . http://www.dstc.edu.au/Research/Projects/MOF/.

Grosso, W. E., Eriksson, H., Fergerson, R. W., Gennari, J. H., Tu, S. W. & Musen, M. A., (1999), **Knowledge Modeling at the Millennium (The Design and Evolution of Protege-2000), Rapport.**

Iazeolla, G., D'ambrogio, A., (January 1998), "A web-based environment for the reuse of simulation models." In 1998 **SCS Western MultiConference on Computer Simulation,** San Diego, CA, USA.

Maxwell, T., (1998), **A Meta-Model Approach to Modular Simulation** http://kabir.cbl.uces.edu/SME3/MetaModels.html.

Microsoft, Corp., (2000), **COM, Microsoft's Component Object Model** http://www.microsoft.com/com/.

MITRE, (2000), **Distributed Object Management Integration System (DOMIS)** http://www.mitre.org/technology/domis/.

Object Management Group, Inc., (1999)**, XML Metadata Interchange** http://www-4.ibm.com/software/ad/standards/xmi.html.

Object Management Group, Inc., (2001)**, CORBA** http://www.corba.org/.

Schmidt , D. C., (2002), The ADAPTIVE Communication Environment (ACE). http:// www.cs.wustl.edu/~schmidt/ACE.html

Sun Microsystems, Inc., (2001), **Enterprise JavaBeans.** http://java.sun.com/products/ejb/.

Sun Microsystems, Inc., (2002)**, JINI Technologie.** http://www.jini.org.

World Wide Web Consortium (W3C), (2001)**, Simple Object Access Protocol (SOAP) 1.2, W3C Working Draft**. http://www.w3.org/TR/2001/WD-soap12-20010709/.

World Wide Web Consortium (W3C), (2001), **Extensible Markup Language** http://www.w3.org/XML/.