Towards a Unified Model of Heterogeneous Object-based Distributed Applications*

Aleksander Laurentowski
e-mail: pinio@ics.agh.edu.pl
Institute of Computer Science
University of Mining & Metallurgy
Al. Mickiewicza 30, 30-059 Cracow, Poland

Abstract

This paper describes briefly some basic conceptual problems likely to be encountered in design, implementation and maintenance of distributed systems, composed of heterogeneous objects, i.e. components existing in diverse environments, exploiting different computational models. These problems concern mainly an attempt to construct a universal model of computation for this kind of systems and a set of basic types of activities within such model. The proposed solutions are given in context of a currently developed monitoring and visualization system for heterogeneous object-based distributed systems, called MODIMOS.

1 Introduction

In the near future we shall probably see more and more new-generation distributed applications, composed of many diverse modules, each of them employing a different computational paradigm. Object-wrapping technique allows for hiding various programming and execution models (e.g. imperative, functional, or logic programming) in the object's implementation, so the developer is able to mix components of diverse origin in one system. Hence, particular parts of such applications could be executed in those programming environments which inherently express the nature of computation. Multi-paradigm programming is itself a large area of problems. We will try to focus only on the integration of environments with a "uniform", at first sight, object-based model. But does this well-known paradigm really offer an at-hand unified semantics? The emerging standards, like e.g. CORBA [3] offer the unification of services' (methods') invocation and dispatching, but do not specify a uniform model of object-based distributed environments. Such a model would be very useful in description of heterogeneous object systems, allowing e.g. for management, monitoring and visualization of different components and applications, as they tend to be particularly large and complex.

Implementations of existing distributed object environments differ in many aspects. Taking a common denominator of a set of them, we propose an abstract model of distributed object environment. This model is a foundation of our Managed Object-based Distributed Monitoring System (MODIMOS), which is a project aimed at development of an expandable platform for visualization and monitoring of distributed applications, built of interoperating heterogeneous components.

*This work was sponsored by KBN under grant no.: 8 S503 015 06.

2 Uniform Model of Computation of Object-based Distributed Systems

For efficient management and maintenance of object-based distributed environments, a uniform abstract model of their architecture and activity should be designed, with which they could comply. From a logical point of view, an application in an object-based system consists usually of some number of communicating independent objects. They cooperate with each other performing a task, having access to other objects which are publicly accessible servers or even members of another application (system). Interactions between objects are supported by interfaces, describing operations available to the others. Well-known object-based distributed systems show some similarity of their computational models. We have analyzed a set of existing environments and decided that for our purposes the abstract model should be a superset (union) of their models of computation.

Hence, in our Logical Viewpoint Uniform Model, the following levels of abstraction have been recognized (in order of inclusion, as in Fig. 1): environment, application, container, object, interface and method. The detailed mapping of them into a set of popular programming environments, i.e. ANSA [1], SR [2] and a CORBA-compliant (Orbix) [4, ?] is shown in the Tab. 1.

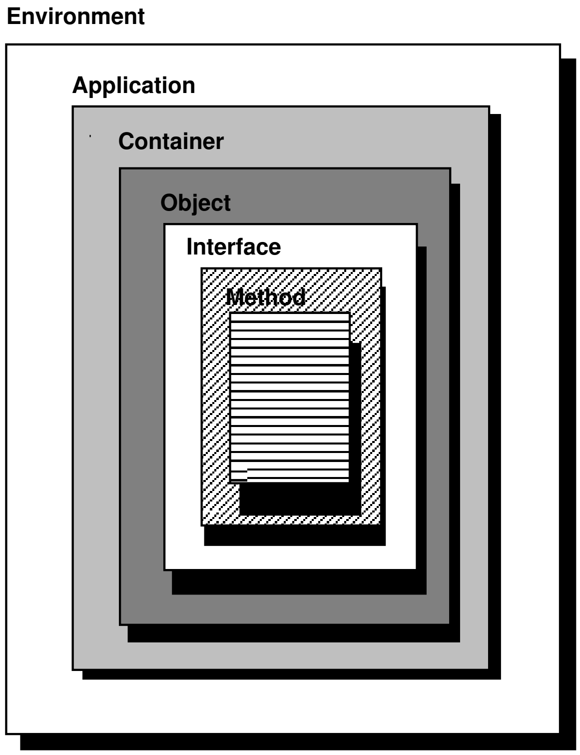

Figure 1: Hierarchy of the Uniform Model components

2.1 Uniform Model Components

A short description of the identified Uniform Model components and problems with their identification follows below.

Unit of abstraction	Environment		
	ANSA	SR	Orbix
application	—	makefile	—
container	capsule	virtual machine	process
object	object	resource, global	object
interface	interface	resource specification	interface
method	service	operation	method

Table 1: Mapping for some concrete environments

Environment A run-time system with tools, languages and libraries, enabling the programmer to create and run applications.

Application In close-architecture environments, like e.g. Synchronizing Resources, it is reasonable and straightforward to identify and differentiate an "application" entity, as a single program executed by one user. Problems arise when we consider open systems, like ANSA or CORBA, with publicly available services (implemented by specialized servers), shared by many users, written by many various and independent developers. One cannot differentiate an application by *makefile* structure or author, because there may be many of them. Tracing of interactions between objects cannot be used for application identification either, due to possible implicit communication cases (e.g. via a trivial database or disk file). The only definition that comes without question is functional, i.e. that only the user is able to identify the application he is currently using.

Container Containers are areas of memory shared by objects, in which the objects reside. We usually regard entities equivalent with operating system processes as containers. Containers have various names, e.g. in ANSA they are called capsules, in SR – virtual machines, Orbix has no particular name for the processes where its server objects reside.

Object We decided to regard as objects all the places, where interfaces can be plugged in, to satisfy cases, where one object can possess more than one interface (like e.g. in ANSA or CORBA).

Interface Interface resides in an object and provides attributes and methods for external use.

Method Methods are characterized by their signatures, usually containing a name of the method and its parameters and results, as well as their type information.

As we decided the Uniform Model to be a union, particular environments may have smaller computational models, i.e. they can lack some of the enumerated levels. For example, to enroll a new environment to the MODIMOS monitoring system, its model of computation should conform at least in part to the specified Uniform Model and the monitor should be aware of the lacking levels and the traditional names of the existing ones. In that way, the system can visualize structure and behavior of the investigated application in a consistent, unified scheme, showing the local idiosyncrasies on demand.

3 Dynamic Behavior of a System

Besides the hierarchical structure of the uniform model of computation, a basic set of events arising in a distributed environment can be defined within that model. Such a set would be particularly useful for the needs of monitoring and visualization of the system's dynamic behavior. So, in our Uniform Model a set of relevant events is associated with each abstraction unit. In MODIMOS, these events are generated by notification functions embedded in the application's code and reported to the monitoring system.

3.1 Basic Events of Interest

The following events have been defined in the Uniform Model, (see Tab. 2 below).

Unit of abstraction	**Events**	**Properties**
Environment	Registration Deregistration	Name, NodesList Ticket
Application	Create Destroy	Ident, Name Ident, Reason
Container	Create Destroy	Ident, Name, Location, ProcId Ident, Reason
Object	Create Destroy	Ident, Name Ident, Reason
Interface	Create Destroy	Ident, Name Ident, Reason
Method	Call End	Ident, Name, SynchType, Parameters Ident, Result

Table 2: Uniform Model Events

A set of properties pertinent to each event is also shown. They are actually implemented as parameters of the notification functions. The above events concern mostly creation and destruction of the abstract units. Notice the treatment of methods is unified with the rest of the units. Methods are invoked (created) – they are even given identifiers for the time of existence – and finally end (are destroyed). This approach helps with visualization of interactions among objects.

4 Identification and Naming

A specific problem in the area of heterogeneous object environments is to assure a space of globally unique object identifiers in the whole system. Each environment possesses its own, non-compatible format of identifiers. We propose usage of structured identifiers to solve that problem. An identifier of a Uniform Model component in MODIMOS is a concatenation of a *ticket* (a key identifier pertinent for each application, assigned arbitrarily by the monitoring module), with a native identifier, treated as a sequence of bytes.

In case there is no native identifier for a unit in its environment (or it is totally hidden from the programmer), we propose a scheme which can help to assure uniqueness of identifiers in the whole heterogeneous system. The scheme covers also names of units, as they are often more interesting and readable for the end-user or observer of the application. The scheme applied in MODIMOS is shown in the Tab. 3 below.

Unit of abstraction	Identifier	Name
Application	'A:no', where 'no' is maintained by an external module	given by the user
Container	native or 'C:IPaddress:ProcId'	native or 'CONTAINER:HostAddress:ProcId'
Object	for environments with multi-interface objects: native, for other: 'O:InterfaceIdentifier'	for environments with multi-interface objects: native, for other: 'OBJECT:InterfaceName'
Interface	native	native, possibly: 'MyClass:MyObject'
Method	'OP:InterfaceIdentifier:Count', where Count is an invocation call counter, maintained externally	native

Table 3: Scheme of uniform identifiers and names

The identifier of an application can be constructed as a concatenation of a prefix and a number assigned by some external module (in MODIMOS, the global monitor). The application's name, due to the reasons given in Sec. 2, must be given by the user (in MODIMOS, at preprocessing time). A concatenation of host address and process id assures uniqueness of container's identifier and name, although e.g. in SR they are part of the language. In environments with multi-interface objects, there is always an identifier of an "object" (usually an instance of some abstract type "OBJECT::"), allowing for adding of interfaces and other operations. Otherwise, an "artificial" identifier must be constructed, as specified. Interfaces cause no pain, but methods need keeping counters of current invocations (in MODIMOS, such counters are added during instrumentation of code).

5 Conclusion

This position paper presented an approach to definition of a uniform model of computation in distributed object-based systems. Such a model can substantially help in description of future heterogeneous applications, composed of object of diverse origin. The solutions presented were verified during implementation of a monitoring and visualization system for this kind of applications, called MODIMOS. The proposed model is simple and incomplete for many cases, e.g. it does not address problems of the applications' statical structure, inheritance, typing nor interface matching, but we plan to extend our Uniform Model in further work.

References

[1] ANSAware 4.0 – Application Programmer's Manual, APM Ltd. Cambridge, UK, 1992.

[2] G.R. Andrews, R.A. Olsson, M. Coffin, I. Elshoff, K. Nilsen, T. Purdin, and G. Townsend, An overview of the SR language and implementation, *ACM Trans. on Progr. Languages and Systems*, 10(1), Jan. 1988.

[3] Draft Common Object Request Broker Architecture Revision 1.1, OMG Report 91-12-1, OMG Inc., 1991.

[4] Chris Horn, Standardizing Your Object Interface, *Object Magazine* 3(3) Sep-Oct 1993, pp. 56-64.