Object-oriented Techniques and Systems in Integration of Software Environments *

Andrzej Uszok, Grzegorz Czajkowski,
Aleksander Laurentowski, Krzysztof Zieliński
Institute of Computer Science University of Mining and Metallurgy
Al. Mickiewicza 30, Kraków, Poland
e-mail: {uszok,grzes,pinio,kz}@ics.agh.edu.pl

August 1994

With growing number of distributed computer systems and information resources, there is an increasing need for technology to integrate them, creating an integrated distributed computing system. In several recent papers ([8, 6, 3]) we have proposed and described implementation of Interoperability Architecture *(IA), built around a family of distributed languages and software tools integrated by the specialized component called* Interoperability Server *(IS), and discussed some of its features. The goal of this paper is to present topics that emerged during the design and implementation of IA and which are closely related to the object-oriented techniques and methodology. After an introduction to IA and its design principles, we concentrate on two issues:*

- *using object-oriented systems in order to achieve interoperability among diverse software environments, representing different programming paradigms,*
- *wrapping up environments in software, providing clean and well-defined external interface to the environment together with reflection and delegation techniques (*"objectifying" an environment*).*

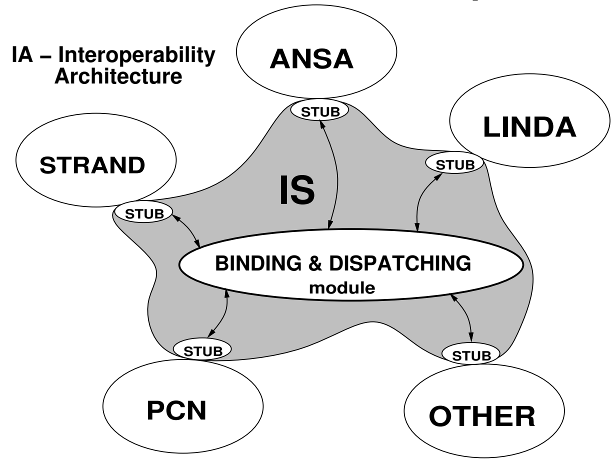
Drawing on the experience with implementing and using IA, we try to address several important questions concerning the features of the model.

1 Introduction to Interoperability Architecture

IA offers an uniform platform for the application software development. It enables composition of application with components created using different languages and techniques and executed in different environments. Our concept grows up from object-oriented approach but our effort is to allow interoperability among diverse distributed systems, not only object-oriented, without creating extra complexity. Then object model chosen by us for wrapping up these components of various origin should be universal and flexible.

*This work was sponsored by Polish Research Council (KBN), under grant no.: 8 S503 015 06.

We found ANSA (and CORBA) model (especially interface specification) quite suitable for our purpose and after some tuning we used it us the base for IA.



IA provides however different functionality than usual object-oriented distributed systems, it addresses the problem of integration of various environments and coordination of applications' execution written in them. Therefore developed by us object-model contains new interesting features.

Integration of programming environments is in our opinion better fitted to the development of a new distributed applications because they may be designed and implemented having a choice of computational and coordination models. A programmer gets an access to various programming techniques, processing models and may use them for the application development. A process of creating an application can be split into parts dealing separately with problem domains in which using a particular approach (e.g. object-oriented, logic or functional programming) is most suitable. The computational model is retained in languages making up the resulting environment.

This approach creates platform for finer grain integration, which means that functions and services written in different languages may be used transparently. The performance may benefit from using fine-grained parallelism if only component languages support it. A original computational model of the distributed application may be preserved across a system and enhanced by the components implemented using different software tools, particularly the most suitable for the given function. Application can be tailored exactly as it is desired. It provides mechanisms to open a closed software environment that is, it makes a cooperation possible via dynamic binding of the programs written in the same or different languages.

On the other hand this solution requires initially much more implementation work because the link between different environments should be established. Because it is the crucial part of the system, it must be designed extremely carefully. Special care must also be taken so that the link does not interfere negatively with environments which it connects. A framework for IS generation should be provided to make the system easy to expandable. It requires thorough familiarity with all component environments, which is usually time consuming to acquire. Necessity to take different approaches while developing applications may become a major problem at creating large programs – a cooperation of experts in different programming paradigms may be necessary.

One of the most important design goals was to preserve computational model of the environment. The call to service foreign to the calling environment should be accomplished using only the mechanisms available in that environment. For example, object-oriented systems should communicate with other IA entities via usual invocations of some object's methods and parallel logic programming systems should use stream-based communication to perpetual processes. We developed such facilities for these kinds of environments in IA.

2 Implementing IS in object-oriented distributed systems

In object-oriented distributed systems like ANSA or CORBA, we introduced a special object which represents IA in the environment. This object looks like an ordinary one for all objects and mechanisms of the environment. It registers services provided by other IA system in the environment's repository and exports from it services needed by other environments. On the other hand it provides mechanisms for communication with IS. Its main functionality is to translate messages from the internal environment representation to the IS protocol. It explains why this module was called a *Gateway*.

This module should be built separately for each new application because it has to contain appropriate *stub* parts generated according to specifications of methods exported and imported by it. However in some systems (CORBA-compliant) it is possible to built this module in the universal way using DII (Dynamic Invocation Interface) and DSI (Dynamic Skeleton Interface). These mechanisms enable *Gateway* to be a server for any method and call any method from other objects. *Gateway* built in such away possesses a reflection features, its methods and functionalities can change or be configured in a free way.

From the other hand *Gateway* is an ordinary object of the given environment. Thus, no program executing in this environment (except for *Gateway*) is aware of the fact that services offered by it are defined in other languages and systems. The original computational model of the environment has been left unchanged.

3 Environments as objects

The set of environments making up IA is dynamic, that is, new (possibly unknown yet) environments may be added to the group of cooperating systems. As one of the main design principles was not to exclude any paradigm or environment from IA, we must be prepared to cope with the diversity of systems. Taking advantage of object-oriented approach, we have decided to wrap up an environment (a not object-oriented *candidate* for *membership* in IA) in software, providing a clean and well-defined interface to the external world. This interface consists of a set of methods, which may be invoked by IS. In turn, IS is also seen as an object by environments in IA.

An environment in IA exhibits interesting properties related to reflection. The set of services offered for use in IA, seen alternatively as methods of an *objectified* environment, may be changed dynamically through registration and deregistration. Thus one can write programs (more or less intelligent) which make decisions concerning the actual contents of interface.

4 Design analysis of IA

We try to integrate different distributed processing environments as generally as possible, so that new, not existing yet environments or the ones with which we are not familiar yet can be easily integrated into the set of cooperating entities. The current implementation includes: ANSA, Orbix, Strand, Linda, PCN and SR. There are three things in a given environment which might be affected by our work:

- The way programs are written,
- Compiling,
- Starting an application/environment.

We would like to interfere as little as possible with these items; however, the fact that the language is a component of some bigger structure will not be always invisible to the user. In other words, it is not always possible to achieve transparency in these subjects.

It would be desirable that the way remote, foreign procedure be called in exactly the same way native code procedure is called. However, it can not always be achieved. For example, in languages where data and program are equivalent (e.g. logic programming) it is possible to call any procedure, for example typed in from the console. If such a procedure is not found in a program, run-time system reports an error. However, without tinkering with the system code, it is impossible to have it check if the called procedure is or is not provided by the environment. In such a case some more involved mechanisms, like perpetual server-processes and incomplete-message communication.

The process of compiling may be affected in systems like ANSA, where it is necessary to know the names of all called procedures at compile-time (usually, such languages have C as their syntax base). Automatic generation of such information requires some preprocessing.

Internally, every environment preserves its own coordination mechanisms. If a process becomes a coordinator of computations, the coordination mechanisms from its environment are reflected in the global coordination. Thus, we have a two-level coordination: the global one, inherited from the system in which the coordinator is written and the local one, which depends on the environment and affects computations which are carried out entirely within that environment, i.e. without communication with IA.

The size of individual computational grains are not limited a priori neither from below or above – in our system, one can call both services written in Orbix[1], where there is no internal concurrency and the smallest grain is a operating system process and services defined in Strand or PCN where processes are extremely lightweight.

5 Summary

In our opinion, the value of the sum of several different distributed software environments can be increased by providing them with constructs enabling cooperation without

[1] As we have been informed, the newest version of Orbix will provide the possibility to use the C lightweight threads library, if available on the given operating system.

the loss of the original computational model. The proposed *Interoperability Architecture* addresses this challenge. In this paper we concentrated on several aspects of IA related directly to object-oriented techniques. The main future research goals are examining a model of heterogeneous objects (i.e. coming from different systems), finding their common features and provide a framework for efficient cooperation. Also, the process of creating *Gateway* should be more automatic for some paradigms (i.e. object-oriented, logic programming, functional programming) and the configuration and autonomous behavior of *Gateway* should be investigated,

Bibliografia

[1] *ANSAware 4.0 — Application Programmer's Manual*, APM Ltd. Cambridge (1992).

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

[3] G. Czajkowski, A. Uszok, K. Zielinski *Distributed Declarative Systems as Parts of Cooperating Software Environments*, Proceedings of ICLP'94 Italy Post-Conference Workshop 'Integration of Declarative Paradigms', TR 224 Max-Planck Institute fur Informatik (1994)

[4] R. Hayes, S. W. Manweiler, R. D. Schlichting, *A Simple System for Construction Distributed, Mixed–language Programs*, The University of Arizona TR 87–1b, (1987).

[5] F. Nayeri, B. Hurwitz, *Dispatching Experiments in a Distributed System.* Technical Report, GTE Laboratories Incorporated (1993).

[6] A. Uszok, G. Czajkowski, K. Zielinski *Interoperability Gateway Construction for Object-Oriented Distributed Systems* Proceedings of Nordic Workshop NWPER'94 Lund, Lund University (1994)

[7] J. C. Wileden, A. L. Wolf, W. R. Rosenblatt, P. Tarr, *Specification Level Interoperability*, Com. of ACM, May vol. 34 (1991).

[8] K. Zieliński, G. Czajkowski, A. Uszok *Coordination through Interoperability in Distributed Processing*, Presented at ECOOP'94 Workshop 'Models and Languages for Coordination of Parallelism and Distribution' (organizers: O. Nierstrasz, A. Yonezawa, P. Cianciarini) (1994)