Configuration in the Framework of Open Distributed Processing[†]

Andreas Vogel vogel@iro.umontreal.ca
Gregor v. Bochmann bochmann@iro.umontreal.ca
Petre Dini[‡] dini@crim.ca
Andreas Polze[‡] polze@inf.fu-berlin.de

Université de Montréal, Département d'IRO
CP 6128, Succursale A
Montréal, H3C 3J7, Canada

Abstract

Open Distributed Processing (ODP) is an ongoing standardization activity by both ISO and CCITT providing an object oriented reference model (RM ODP) for open distributed systems. Certain abstraction levels called viewpoints are introduced by the RM ODP.

The formal description technique LOTOS is used to define formal models for the computation and engineering viewpoint by sublanguages LOTOScomp and LOTOSeng. The use of these sublanguages as configuration languages on different abstraction levels is shown and illustrated by an example. The mappability of LOTOS specification in particular from the engineering viewpoint into implementations is outlined for the Object Space.

1. Introduction

Open distributed processing (ODP) is an ongoing standardization activity by both ISO and CCITT providing a object oriented reference model (RM ODP) for open distributed systems. The complexity of this kind of systems is managed by introducing viewpoints. A viewpoint means a certain abstraction of the system determined by a particular interest. There are the five viewpoints enterprise, information, computational, engineering, and technology. An introduction is given in section 2.

Configuration aspects appear in the computational, engineering, and technology viewpoint. Configuration languages for the computational and engineering viewpoint are defined as subsets of the formal description technique (FDT) LOTOS in section 3. LOTOS brings formal semantics in the modeling process and provides executable models.

† This work was supported by a grant from the Canadian Institute for Telecommunication Research (CITR), under the Networks of Centers of Excellence Program of the Canadian Government.

‡ P. Dini is with Centre de Recherche de Montreal and A. Polze with Freie Universität Berlin.

The Object Space allows processes on different nodes of a network to exchange arbitrary objects. It relies on the decoupled communication paradigm and is therefore well suited for dynamic configuration of distributed applications in the context of open systems. We use Object Space to implement our configuration scheme.

The addressing of configuration aspects in the ODP framework and the application of LOTOS as configuration language is discussed by a distributed multi-media example in section 5. Section 5 concludes the paper.

2. Open Distributed Processing

Open Distributed Processing[4,6,7,5] (ODP) is an ongoing standardization activity by both ISO and CCITT providing a object oriented reference model for open distributed systems. The main concept is the viewpoint. A viewpoint is focused on parts of an ODP system determined by a particular interest. The concepts, rules, and structures appropriate for the specification of an ODP system are given by a corresponding description language, e.g. computational language for the computational viewpoint. The semantics of ODP objects is defined by ODP functions. With respect to configuration, the trading function comparable to a directory service and several transparency function are of particular interest. The RM ODP characterizes its five viewpoints as follows:

Enterprise viewpoint
The enterprise viewpoint on an ODP system focuses on the purpose, scope, and policies for the system.

Information viewpoint
The information viewpoint on an ODP system focuses on the semantics of information and information processing activities in the system.

0-8186-5390-6/94 $3.00 © 1994 IEEE

Computational viewpoint

The computational viewpoint on an ODP system focuses on the functional decomposition of the system into objects which are candidates for distribution.

Hence, the corresponding computational language specifies the system in terms of communicating objects. Computational objects are composed by interfaces. Interaction between computational objects is described by a not further elaborated computational infrastructure. There are three types of interfaces:

operational interface
: provide operations. Invocation of an operation follows the remote procedure paradigm.

transactional interface
: is an operational interface providing only transactional operations.

stream interface
: support continuous media.

Operations and streams are syntactically defined by signatures, respectively.

Implicit and explicit binding between interfaces is considered by the computational language.

Engineering viewpoint

The engineering viewpoint on an ODP system focuses on the infrastructure required to support distribution. The engineering language specifies architecture of this infrastructure in terms of

node
: is a resource management domain.

capsule
: is the unit of allocation and encapsulation.

cluster
: is the set of basic engineering objects, the unit of activation and deactivation.

basic engineering object
: represents a computational object.

nucleus object
: coordinates processing, storage, and communications functions using the resources of the corresponding node.

The communication within this infrastructure is expressed by

stub object
: acts as a basic engineering object's representative to a object in a different cluster.

binder object
: maintains binding between interacting basic engineering objects.

protocol object
: interacts with other protocol objects to achieve interaction between basic engineering objects in different clusters.

channel
: is a configuration of stub, binder, protocol, and interceptor objects.

interceptor
: enables interactions to cross administrative and communications domains.

Technology viewpoint

The technology viewpoint on an ODP system focuses on the on the choice of technology to support the system.

The trading function

The trading function provides means to advertise services. It is realized by the trader. Service provider can export their services. Potential service user can import from the trader which means to get information on available service and their accessibility.

3. A formal modeling approach

A formal approach modeling the ODP description languages of the computation, and engineering viewpoint is given[15,16] using the formal description technique (FDT) LOTOS[10,3,2]. The general idea is to map the concepts of the description languages on generic LOTOS constructs also called templates. The modeling rules are expressed by relationships between such LOTOS templates. Genericity in LOTOS is limited to the parametrization of processes, hence a meta level is introduced by the definition of LOTOS sublanguages. By such a sublanguage, a generic model of a particular viewpoint is defined containing the class of specification satisfying the requirements of the corresponding ODP description language.

This approach gives formal semantics to the ODP modeling concepts and leads also to executable models. These models provide the symbolic execution LOTOS specifications supported by several tools.

4. The Object Space Approach

The Object Space approach supports communication and synchronization between components of a distributed application. All components get access to a shared associative data store known as the Object Space. Every component may write objects into the store which can subsequently be read by others. In order to read an object a component presents a template which is matched against the objects. If no matching object can be found within Object Space then the read operation blocks. This way objects may be passed from one component to another. Object Space itself is implemented in a distributed fashion, employing Object Space Manager processes in different nodes of a network.

This style of communication is called decoupled. Neither does the sender of an object know its receiver nor vice versa. If there exist several components trying to read one object it is nondeterministic which of them will succeed.

Decoupled communication is well suited for scalable distributed computing within a network of workstations. Object Space integrates coordination constructs (communication and synchronization) into the C++ language, thus integrating Linda-like communication with object-oriented essentials as inheritance, polymorphism and data encapsulation.

5. Configuration of a distributed multi-media application

A movie server - an in the nearer future upcoming multi-media application intended to be supported by communications, telecommunications, and computer industry - is selected as case study. An informal specification of this application containing beside the server itself also an appropriate client. The application provides facilities to playback movies on the clients site which are stored in digital form on the servers site. The above described modeling approach was applied to this case study[17].

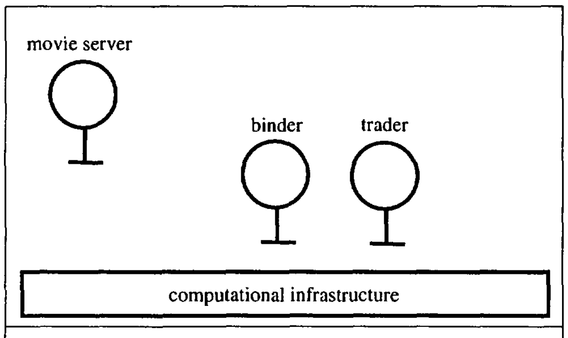


figure 1: Computational model - initial configuration

5.1. Computational Viewpoint

In the following certain configuration scenarios are discussed in the given framework. Instead of traces or trees representing the execution of the specification by a simulation tool[8] which require deeper understanding of LOTOS and the tool, figures are given to illustrate the system's configurations. (Circuits describe computational objects, T-bars stand for computational interfaces in figures 1 - 3.)

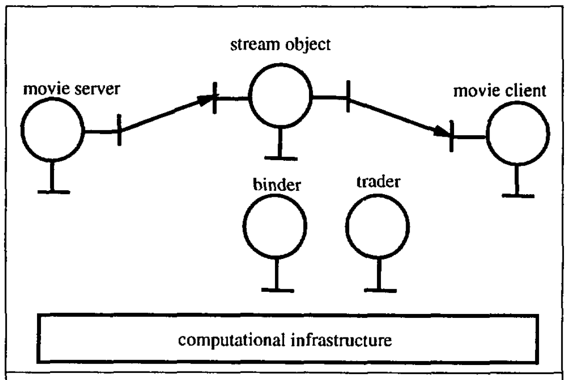


figure 2: Computational model - scenario A

Figure 1 illustrates the initial structure of the system from the computational viewpoint. There are instances of the objects movie server, trader, binder (supporting explicit binding), and the computational infrastructure.

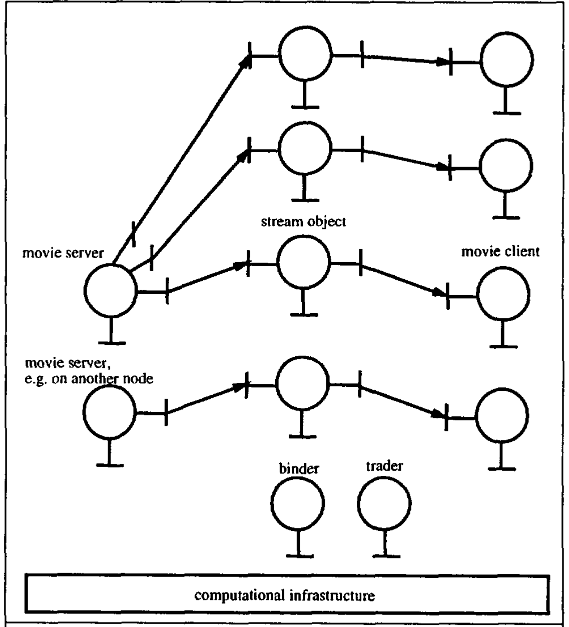
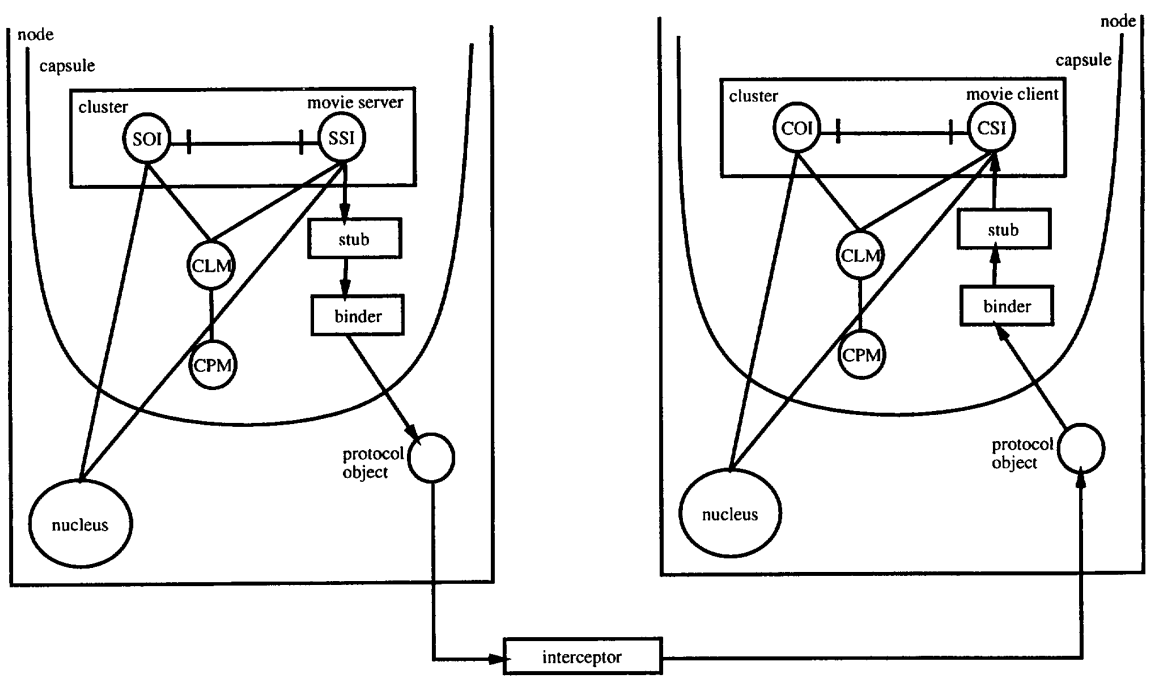

stream object
movie server
movie client
movie server, e.g. on another node
binder
trader
computational infrastructure

figure 3: Computational model - scenario B

Figure 2 shows a scenario where a movie client has been instantiated and explicitly bound to the movie server by a stream object. To reach the scenario A from the initial configuration the following steps were executed:



CLM: cluster manager
SOI: server operational interface
COI: client operational interface

CPM: capsule manager
SSI: server stream interface
CSI: client stream interface

figure 4: Engineering model - scenario A

- Instantiation of a movie client.
- The movie client requests a stream connection from the server, therefor
  - the movie server instantiates a new stream interface and
  - requests from the binder the instantiation of a stream object. The binder returns the interface identifier of the stream interface.
- When the interface of the stream object are known to the server and the client, these interfaces and the stream interfaces of the client and the server are activated and data frames can be transmitted.

It is supposed that the movie server's capacity is limited to support three clients. Hence a fourth client receives a negative response from the server when asking for a stream connection. Then the client can get information from the trader about other available movie servers and an interface reference. If the parameters of another movie server seem to be suitable for the client, a stream connection can be established as shown above. Such a configuration is shown in figure 3.

5.2. Engineering Viewpoint

From the engineering viewpoint, the situation is different. Distribution of objects is explicitly visible. The units of distribution are node, capsule, and cluster. Computational objects are mapped on basic engineering objects, in this case each computational interface is represented by a basic engineering object, e.g. server's and client's operational and stream interfaces in our case. Figure 4 illustrates the engineering viewpoint of the configuration which was given in Figure 2 from the computational viewpoint.

It is asumed that the server and the client are located on different nodes. Within a node, there are by definition of the engineering language a nucleus, i.e. the abstraction of an operating system, a protocol object, and capsules. In the configuration shown in figure 4, there are in each of the nodes one instance of a capsule. A capsule contains by its definition a capsule manager and a the cluster manager of every instantiated cluster, stubs, binders, and clusters. In our configuration, there is only one cluster in each of the capsules containing an engineering operational and a stream interface, respectively. Figure 4 also illustrates an already established stream connection indicated by the arrows.

5.3. Technology Viewpoint

We use the Object Space approach to distributed computing for the implementation of our scenario as shown in figure 4. Object Space relies on the concept of associatively addressed objects in a shared dataspace. We want to outline briefly how a distributed system may be configured using stub, binder and protocol objects as mentioned earlier on top of Object Space.

The stream-based communication between client and server of our movie application uses a connection-oriented transport protocol (e.g. TCP, ST II). Prior to that communication a binding between client and server has to be performed. The stub objects shown in figure 4 are capable of transfering data between server and client once a connection has been established. But when discussing configuration aspect we have to focus on the binder objects which allow a negotiation between client and server prior to connection establishment.

Binder objects use Object Space communication to establish a connection. A server offers its service by storing a special object in the Object Space. This object — named as protocol object in figure 4 — describes the server (a unique ID), the name of the service offered and a way to access the stub object (i.e. a network address+port). If a server is capable of serving multiple clients in paralle it simply stores several protocol objects in Object Space.

Due to the associative addressing mechanism within Object Space the binder of a client may easily retrieve a matching protocol object from Object Space" (with the operation `in`) if it knows the name of a service. So the binder can figure out a server's address and the stub object may establish a connection.

Since the client removes the protocol object describing the server's interface from Object Space only one client may successfully access a certain interface at one time. After completion of all communication the server stores a new protocol object in Object Space, thus indicating that the service is available again. This provides for some reliability after a communication breakdown — clients may come and go without affecting the server's behaviour.

Object Space is itself implemented in a distributed fashion. It employs several Object Space Manager processes on different nodes in a network. So Object Space Manager processes are part of the nucleus as shown in figure 4.

6. Conclusions

The shown approach to configuration is based on the ODP framework and the use of the formal description technique LOTOS. This leads to formal and executable models of different abstraction levels determined by the ODP viewpoints, computational and engineering.

The advantages of these approach are seen in

- the formal and executable models and
- a standardized modeling framework for open distributed systems.

The executable models provide the addressing of configuration aspect already in early design stages and on different abstraction levels. The validation of intended configurations of the systems is supported by tools executing, e.g. simulating the models.

A similar modeling approaches is given with the ANSA reference model[1] which is derived from the RM ODP and Object Management Architecture (OMA)[12]. However, instead of using formal models, interface definition languages (IDL) are introduced. This leads canonically to compiler supported distributed implementations, but they are lacking of a behaviour definition and are consequently not executable on the model level. A similar situation is described with the Darwin[11] and the Surgeon[9] approaches.

A approach to transform LOTOS models into distributed implementations has been already presented[15]. This approach bases on a distributed, C++ based programming environment called, Object Space[13,14], supporting decoupled communication between its components. The further elaboration of ODP's technology viewpoint will lead to a deeper understanding of this question.

References

1.
"The ANSA Reference Manual.," Architecture Projects Management Limited, Cambridge (1989).

2.
T. Bolognesi and E. Brinksma, "Introduction to the ISO Specification Language LOTOS.," *Computer Networks and ISDN Systems,* 14, pp. 25-59 (1987).

3.
E. Brinksma, "Tutorial on LOTOS" in *The Formal Description Technique LOTOS,* ed. P. van Eijk, C.A. Vissers, and M. Diaz, pp. 171-194, Nort-Holland (1989).

4.
CCITT and ISO/IEC JTC1/SC 21/WG 7, "Information Technology - Basic Reference Model of Open Distributed Processing - Part 1: Overview and Guide to Use," ISO/IEC/JTC1/SC21 N7053, Working Draft (July 1992).

5.
CCITT and ISO/IEC JTC1/SC 21/WG 7, "Information Technology - Basic Reference Model of Open Distributed Processing - Part 5: Architectural Semantics," ISO/IEC/JTC1/SC21 N7056, Working Draft, May 1992.

6.
CCITT X.902 and ISO CD 10746-2.2, "Information Technology - Basic Reference Model of Open Distributed Processing - Part 2: Descriptive Model," ISO/IEC/JTC1/SC21 N7524, Committee Draft (December 1992).

7.
CCITT X.903 and ISO CD 10746-3, "Information Technology - Basic Reference Model of Open Distributed Processing - Part 3: Prescriptive Model," ISO/IEC/JTC1/SC21 N7525, Committee Draft (December 1992).

8.
H. Eertink and D. Wolz, "Symbolic Execution of LOTOS Specifications" in *Formal Description Techniques V,* ed. M. Diaz and R. Groz, pp. 295-310, North-Holland (1993).

9.
C. Hofmeister, E. White, and J. Purtilo, "Surgeo: a packager for dynamically reconfigurable distributed applications," *Software Engineering Journal,* Volume 8 Number 2, pp. 95-101 (March 1993).

10.
ISO, *LOTOS - A formal description technique based on the temporal ordering of observational behaviour* (1988).

11.
J. Magee, N. Dulay, and J. Kramer, "Structuring parallel and distributed programs," *Software Engineering Journal,* Volume 8 Number 2, pp. 73-82 (March 1993).

12.
OMG, "Object Management Architecture," OMG Document.

13.
A. Polze and A. Vogel, "Generierung verteilter Prototypimplementationen aus LOTOS-Spezifikationen nach dem Object Space-Ansatz," Report B-93-2, Freie Universität Berlin, Institut für Informatik (1993).

14.
A. Polze, "The Object Space Approach: Decoupled Communication in C++" in *Proceedings of TOOLS USA'93,* Santa Barbara (August 1993.).

15.
A. Vogel, "Entwurf, Realisierung und Test von ODP-Systemen auf der Grundlage formaler Beschreibungstechniken," submitted as PhD thesis, Humboldt-Universität zu Berlin (1993). in German.

16.
A. Vogel, "On ODPs Architectural Semantics using LOTOS" in *Proceedings of the 1st International Conference on ODP,* pp. 340-345, Berlin (1993).

17.
A. Vogel, (ed.), "Project Overview, QoS Architecture, and QoS Negotiation Protocol," Project report, CITR project on broadband services (Oct. 1993.).