Coordinating Open Distributed Systems

Juan Carlos Cruz[1], Stéphane Ducasse
University of Bern[2], Switzerland

Abstract. Open Distributed Systems are the dominating intellectual issue of the end of this century. Figuring out how to build those systems will become a central issue in distributed system research in the next future. Although CORBA seems to provide all the necessary support to construct those systems. It provides a very limited support to the evolution of requirements in those systems. The main problem is that the description of the elements from which systems are built, and the way in which they are composed are mixed into the application code. Making them difficult to understand, modify and customize. We think that a solution to this problem goes through the introduction of the so called coordination models and languages into the CORBA model. We propose in this paper the introduction of our object coordination model called CoLaS into the CORBA model.

1 Motivation

Software development of distributed systems have changed significantly over the last two decades. This change has been motivated by the goal of producing open rather than close proprietary distributed systems. Open Distributed Systems (ODS in the following) [Crow96a] are systems made of components that may be obtained from a number of different sources which together work as a single distributed system. ODSs are basically "open" in terms of their topology, platform and evolution: they run on networks which are continuously changing and expanding, they are built on top of a heterogeneous platform of hardware and software pieces, and their requirements are continuously evolving. Evolution is the most difficult requirement to meet, since not all the application requirements can be known in advance. ODSs are the dominating intellectual issue of the end of this century. Figuring out how to build them will become a central issue in distributed systems research in the next future.

In 1988 the International Standards Organization (ISO) began work on preparing standards for Open Distributed Processing (ODP). These standards have now been completed, and define the interfaces and protocols to be used in the various components of an ODS.The ODP standards provide the framework within which ODSs may be built and executed. One of the most popular (if not the most) specifications for some parts of the ODP is the Common Object Re-

1. Software Consultant, ObjectShare AG. Zurich, BelleriveStrasse 201, CH 8034.

2. Author's address: IAM University of Berne, Neubruckstrasse 10, CH 3012 Berne, Switzerland. Tel: 41+(31) 6313315. Fax: 41+(31) 6313965. E-mail: {cruz, ducasse}@iam.unibe.ch.

quest Broker Architecture (CORBA) [OMG95a]. This middleware proposed by the Object Management Group (OMG) provides a standard for interoperability between independently developed components across networks of computers. Details such as the language in which they are written or the operating system in which they run is transparent to their clients. The OMG focused on distributed objects as a vehicle for system integration. The key benefit of building distributed systems with objects is encapsulation: data and state are only available through invocation of a set of defined operations. Object encapsulation makes system integration and evolution easier: differences in data representation are hidden inside objects, and new objects can be introduced or replaced in a system without affecting other objects.

Although the CORBA middleware seems to provide all the necessary support for building and executing ODSs, it only provides a very limited support to the evolution of requirements in those systems.The main problem is that the description of the elements from which systems are built, and the way in which they are composed remains mixed into the application code. Making them difficult to understand, modify and customize.

To our point of view a possible solution to this problem goes through the introduction of the so called coordination models and languages into the CORBA model. Coordination technology addresses the construction of open, flexible systems from active and independent software entities in concurrent and distributed systems. The main goal of a coordination model and language is to separate the coordination aspect from the computational aspect in a distributed system. Separation of concerns facilitates abstraction, understanding and evolution of concerns. In the Software Composition Group (SCG) we have been working in the last few years in the definition of a coordination model and language for concurrent object oriented systems. Some of our results are [Cruz98a][Duca98c]. The coordination model in which we actually work called CoLaS [Cruz99a] is based on the notion of Coordination Groups. A Coordination Group is an entity that specifies and enforces cooperation protocols, multi-action synchronizations, and proactive behaviour within groups of collaborating active objects.

2 The CoLaS Coordination Model

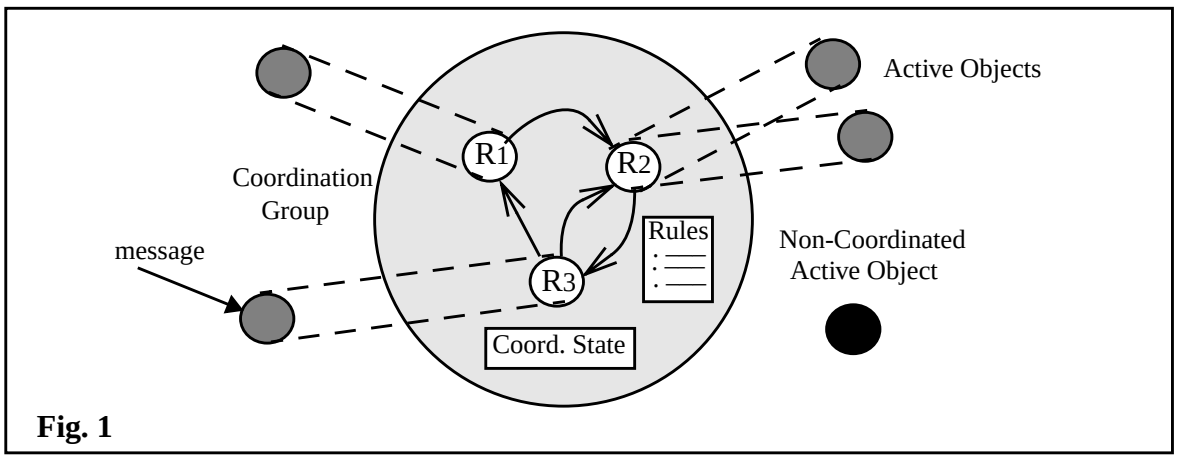

Fig. 1

The CoLaS coordination model is built of two kinds of entities: the participants and the coordination groups.

2.1 Participants

In CoLaS the participants are *active objects* [Brio96a]: objects that have control over concurrent message invocations. Active objects represent independent units of execution in a concurrent system. By default active objects communicate in an *asynchronous* way. Replies are managed using *explicit futures* so that objects do not blocked while waiting for their replies.

2.2 Coordination Groups

A *Coordination Group* (CG in the following) is an entity that specifies and enforces the coordination of a group of participants in the realization of a common task. The primary tasks of a CG are: (1) to enforce cooperation actions between participants, (2) to synchronize the occurrence of those actions, and (3) to enforce proactive actions [Andr96a] (in the following proactions) in participants based on the state of the coordination.

Coordination Specification

A CG is composed of five elements (Fig. 1): a Role Specification, a Coordination State, a Cooperation Protocol, Multi-Action Synchronizations and Proactions.

- *The Role Specification*: defines the roles that participants play in the group. A role identifies abstractly entities sharing a common behavior. To play a role, an object should possess at least the functionalities required by the role (interface compatibility). A role can be played by more than one participant.
- *The Coordination State*: defines the information needed for the coordination of the group. It concerns historical information like whether a given action has occurred or not in the system or participant state information. The Coordination state is global to the group, and/or global to all the participants of a given role, and/or local to each participant within a given a role. The coordination state is specified by declaring coordination variables.
- *The Cooperation Protocol*: defines implications between participant actions. These implications have the form <Role> <Message> <Operator> <Coordination Actions>. The <Operator> specifies <Coordination Actions> that have to be done when a participant playing the role <Role> receives the message <Message>. We have three types of operator: ImpliesBefore, ImpliesAfter and Corresponds.

  -The operators ImpliesBefore and ImpliesAfter specify different moments at which the coordination Actions must be executed. ImpliesBefore (resp. after) specifies that the coordination actions have to be executed before (resp. after) the execution of the <Message> [Cruz99a].

  -The Corresponds operator [1]allows the definition of participant behavior specific to the coordination, i.e. the fact that an object plays a role in a GC gives it an extra behavior

1. The Corresponds operator has been recently introduced. It allows a clear separation between the intrinsic behavior of a participant and his behavior due to his participation to a CG. Note this is possible because the implementation language of CoLaS is dynamically typed.

defined using the Corresponds operator. The Corresponds operator specifies the coordination actions that have to be executed when a participant receives the <Message>.

- *The Multi-Action Synchronizations:* specifies synchronizations constraints over message exchanged between participants. They have the form <Role> <Message> <Operator> <Synchronization Conditions>. They specify conditions that constraint the execution of the message <Message> received by a participant playing the role <Role>. Three different types of operators can be specified: Ignore, Disables, and Atomic. Depending on the operator, the message <Message> should be ignored or delayed by the object. In the Atomic case the <Message> represents a set of messages that must be executed as an atomic unit. The synchronization conditions refers to information like: the state of the group, the identity of the receiver/sender of a message, historical information on the coordination, etc. [Cruz99a].
- The Proactions: specifies Coordination Actions that must be enforced by the CG independently of messages exchanged by participants assuming that a certain condition holds. These conditions are expressed in terms of the coordination state. Proactions have the form <Condition> <ProOperator> <Coordination Actions>. One kind of ProOperators can be used: Once.It ensures that the coordination actions are executed only one time when the condition is satisfied. The evaluation of the conditions is done by the CG.

The last three elements of this model are specified using rules [Andr96a][Mins97a].

2.3 Dynamic Aspects

CoLaS support three types of dynamic coordination changes [Cruz99a]: (1) new participants can join and leave the group at any time, (2) new groups can be created and destroyed at any time, and (3) the coordination behavior can be changed by adding and removing rules to the CG. Evolution of coordination requirements can be addressed by combining the different types of dynamic changes.

2.4 Evaluation

The CoLaS model combines the advantages of using groups as an organisation abstraction and rules as explicit entities regulating the coordination. It provides the following concrete advantages:

- *Separation of Concerns:* The coordination and the computational aspects are specified separately in two different entities the CGs and participants. This separation promotes reuse of computation and coordination abstractions:participants can be reused independently of how they are coordinated, and coordination protocols can be reused independently on different groups of active objects.
- *Encapsulation:* The coordination is specified in CGs independently of the internal representation of the objects it coordinates. Participants are referred abstractly according to the role they play in the CG.
- *High-Level Abstraction:* All low level operations concerning the coordination are managed by CoLaS. Programmers focus on how to express coordination and not on how to do it.

- *Multi-Object Coordination:* The coordination is not limited to two objects but to groups of objects. Their behavior is specified abstractly in terms of roles and their respective interfaces.
- *Dynamic Evolution of the Coordination:* The model supports dynamic coordination changes in three distinct aspects: (1) coordination behavior can be changed by adding and removing rules, (2) new coordination groups can be created and destroyed at any time, (3) new participants can join and leave the group at any time.

2.5 CoLaS-D

Actually we experiment the construction of a coordination programming system for distributed active objects called CoLaS-D. The CoLaS-D programming system (Fig. 2) is based in the CoLaS coordination model. CoLaS-D is been built on top of a middleware framework called Distributed Smalltalk (DST)[Objs98]. DST is a CORBA 2.0 compliant framework for Smalltalk.

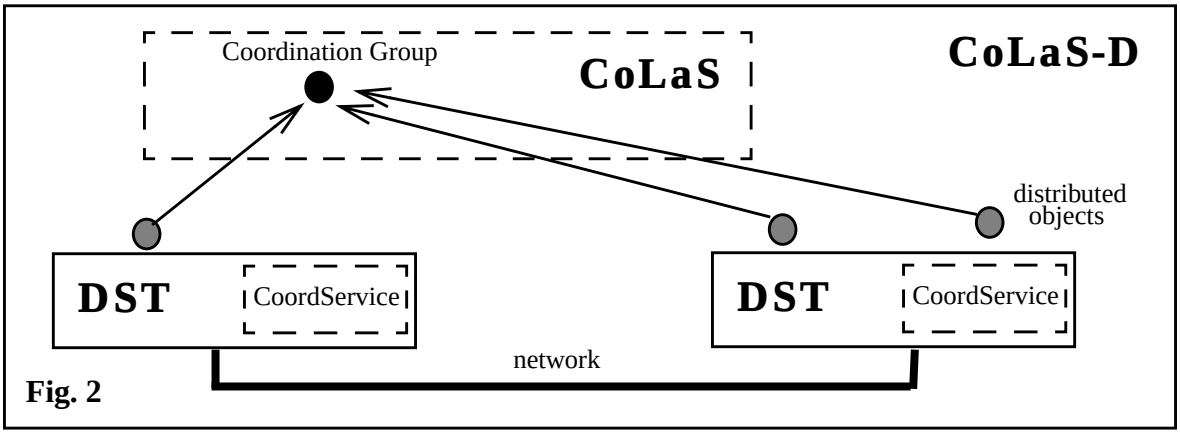

Fig. 2

We use in CoLaS-D a specific facility offered by the DST framework called the Implicit Invocation Interface (I3), an extension to the CORBA facilities that provides a more natural Smalltalk paradigm for developing distributed Smalltalk object systems. This facility can be used to realize interaction only between distributed Smalltalk objects. No IDLs definitions are required to interact. Instead of explicitly specify distributed classes interfaces using IDLs, developers can turn on the I3 message transmission mechanism and allow I3 to handle marshalling and unmarshalling of messages between distributed objects.

From a coordination viewpoint DST provides all the basic facilities required to realize coordination in a distributed object environment: remote communication of distributed objects, location of remote objects by using a Naming Service, control of concurrent access to objects by using a Concurrency Control Service, creation of remote objects by using a LifeCycle Service, and management of distributed transactions by using a Transaction Service. We decided to limited our work in a first time to the coordination of distributed Smalltalk active objects . We used the DST I3 facility to realize the distributed interaction necessary to the coordination. The first version of the CoLaS-D programming system defines a coordination service CoordService that maintains references to CGs across the network. Whenever a CG is created in a distributed system, the reference to the CG and its name its automatically register into the coordination service. Objects can join CGs in this way wherever they found in the network. They only need to request to the Coordination Service for remote references to the CGs where they want to partic-

ipate. From a coordination viewpoint it is transparent whether the CG founds local or remote. The problem with this approach is that CGs become critical points of failure. To recover from a CG failure a new CG can be created (or one of the participants could be picked to play a dual role as CG). Actually we work in a second version of CoLaS-D in which a solution to this problem is proposed using replicas [Coul94a].

3 Conclusion

We consider that a coordination model and language for *heterogeneous* distributed objects on top of CORBA is still missing. A such model and language will provide the necessary support for building and evolving ODS. We consider the CoLaS model as a good candidate to this. In our research agenda this work corresponds to the third version of the CoLaS-D programming system.

References

[Andr96a] J-M.Andreoli, S. Freeman and R.Pareschi, *The Coordination Language Facility: Coordination of Distributed Objects*, Journal of Theory and Practice of Object Systems (TAPOS), vol.2, no. 2, 1996, pp. 635-667.

[Brio96a] J-P. Briot, *An experiment in the Specification and Classification of Synchronization Schemes*, LNCS vol. 1049, Springer-Verlag, 1996, pp. 227-249.

[Coul96a] G. Coulouris, J. Dollimore and T. Kindberg, *Distributed Systems*, Addison-Wesley, 1994.

[Crow96a] J. Crowcroft, *Open Distributed Systems*, UCL Press, 1996.

[Duca98c] S.Ducasse and Manuel Guenter, *Coordination of Active Objects by Means of Explicit Connectors*, CTIS'98, IEEE Computer Society Press, pp. 572-577.

[Cruz98a] J.C.Cruz and S. Tichelaar, *Managing Evolution of Coordination Aspects in Open Systems*, CTIS'98,IEEE Computer Society Press.

[Cruz99a] J.C.Cruz and S. Ducasse, *A Group Based Approach for Coordinating Active Objects*, COORDINATION '99, LNCS 1594, Springer Verlag, pp. 355-370.

[Gele92a] D. Gelernter and N. Carriero, *Coordination Languages and their Significance*, Communication of the ACM vol. 35, no. 2, February 1992.

[Mins97a] N.Minsky and V. Ungureanu, *Regulated Coordination in Open Distributed Systems*, COORDINATION'97, LNCS 1282, Springer-Verlag, 1997, pp. 81-97.

[Obje98a] ObjectShare Inc., *Distributed Smalltalk*, 1998, http://www.objectshare.com/products/dst/info/dst.h

[OMG95a]*The Common Object Request Broker: Architecture and Specification*. Object Management Group, Framingham, MA, July 1995.