# Coordination Requirements for Open Distributed Systems

Thilo Kielmann and Guido Wirtz
Dept. of Electrical Engineering and Computer Science, Univ. of Siegen, Germany
{kielmann,guido}@informatik.uni-siegen.de

This work discusses the central requirements of coordination models suitable for open distributed systems and evaluates some commercially available systems w.r.t. these requirements. Finally, the design of the *Objective Linda* coordination model is sketched.

## 1. Introduction

Programming of open distributed systems is primarily concerned with coordinating concurrently operating active entities. Concurrent programming languages based on the concept of *generative communication* [2] initiated the research area of *coordination* [3]. Today, the interaction between active entities is typically investigated based on this notion; see for example the very influential work in [1].

Coordination as the key concept for modelling concurrent systems involves managing the communication which is necessary due to the distributed nature of a system, the expression of parallel and distributed algorithms, as well as all aspects of the composition of concurrent systems. We characterize coordination by the following notions: *Agents* are active, self–contained entities performing *actions* on their own behalf. Actions are divided into two different classes: (1) *Inter–Agent actions* which perform the communication between different agents and hence are the subject of coordination models. (2) *Intra–Agent actions* which are all actions belonging to a single agent like e.g. computations. We call a collection (or a system of) interacting agents a *configuration*. Hence, *coordination* can be defined as managing the inter–agent activities of agents collected in a configuration.

The primary objectives of object–orientation concern program modularization, encapsulation, and reuse of software components. The obvious idea of exploiting the encapsulation property for concurrent programming has generated a lot of research work. The most promising approach to concurrent object–oriented programming seems to be based on *active objects* which execute their own threads of control while they are still protected by their class interface [8]. Furthermore, the active–object approach enables generative communication to be included in object–oriented concurrent systems.

Open distributed processing is a currently evolving field which is characterized by the upcoming ISO reference model of open distributed processing (RM–ODP) [4]. In the RM–ODP definition, *distributed systems* have to cope with *remoteness* of components, with *concurrency*, the *lack of a global state*, and *asynchrony* of state changes. In addition, *open distributed systems* are characterized by *heterogeneity* in all parts of the involved systems, *autonomy* of various management or control authorities and organizational entities, *evolution* of the system configuration, and *mobility* of programs and data.

## 2. Properties of Coordination Models for Open Distributed Systems

Open Distributed Systems are inherently *dynamic* and *heterogeneous* systems. Hence, an appropriate coordination model has to fulfill at least the following requirements:

- At any time, agents are allowed to enter or leave a configuration (**Dynamics**). This implies that coordination laws must not rely on the existence of specific agents. The same holds for communication. Generative communication [2] by generating and consuming separate entities, usually contained in a specific computational space, is required in order to provide a suitable communication model (**Generativeness**). On the other hand, it is also essential to protect a configuration from undesired interaction with agents outside (**Encapsulation**).
  By definition of open distributed systems, there is no overall compile time. Hence, it must be possible to program new agents during the runtime of an already existing system, i.e. specify an agent's behaviour in a separate program (**Decentralization**).
- Coordination models for use in heterogeneous environments must not rely on properties of specific hardware, programming languages or communication media like data types or their representations (**Interoperability**).

Besides capturing the requirements of *what* to specify, the methods used *how* to model systems are essential in order to build really large but still maintainable systems.

- A model suited for coordinating large systems must be as simple as possible. Hence, all agents should be modelled in a uniform way (**Homogeneity**). For the purpose of really large systems, it is vital to divide the overall configuration into smaller subconfigurations. Hence, it must be possible to treat entire configurations like single agents at a more abstract coordination level (**Hierarchical Abstraction**). Inter–agent actions have to be cleanly separated from intra–agent actions in order to distinguish between the concerns of coordination on one hand and of computations on the other (**Separation of Concerns**).

Last but not least, a coordination model should be based on a rigorous formal semantics in order to allow reasoning about specifications.

## 3. Commercially available systems.

The RM–ODP model conceptually provides the basis for commercially available systems. In this model, communication is performed by providing and requesting services. This model enforces a request–reply communication structure which confines communication between agents to remote procedure calls. Offering and using services is done by communicating with a so–called *trader*, a repository of type definitions which is used to identify offered and requested service types. This service lookup requires two–step communication which reduces the communication model's *homogeneity*. Furthermore, communication based on object identifiers (given by the trader) inherently bears the potential of dangling references in the case of dynamically changing configurations due to disappearing (or moving) objects which contradicts the requirement of *dynamics*.

The Common Object Request Broker Architecture (CORBA) [7] provides interoperability between heterogeneous machines and applications. Its central component, the Object Request Broker (ORB), acts as a trader in the sense of RM–ODP. CORBA systems comply with the requirement of *separation of concerns*, because computations and remote

object calls are treated in different ways. *Decentralization* is possible by using CORBA's dynamic interface mechanism which allows to introduce new interfaces at runtime. Unfortunately, *dynamics* can not be supported due to the lack of *generativeness*. Furthermore, CORBA provides neither means for *hierarchical abstractions* nor for *encapsulation*.

Microsoft's *Object Linking and Embedding* (OLE) is a set of so–called *component objects* for advanced document processing. It is based on the Component Object Model (COM) [6], which provides a binary interface standard between possibly heterogeneous application components. Like with CORBA, new interfaces can be introduced at runtime to a simplified form of a trader, called *component object library* (COL). This provides means for *decentralization*. *Interoperability* can be achieved with COM based on its binary component interface which introduces component access to application programs by so–called *virtual function tables* (v–tables) which realize bindings to given programming languages. Because computation and communication are intermixed in one mechanism (procedure call), *separation of concerns* is not well supported. Because COM has no notion of *generativeness*, it cannot really support system *dynamics*. Like CORBA, there are no provisions neither for *hierarchical abstractions* nor for *encapsulation*.

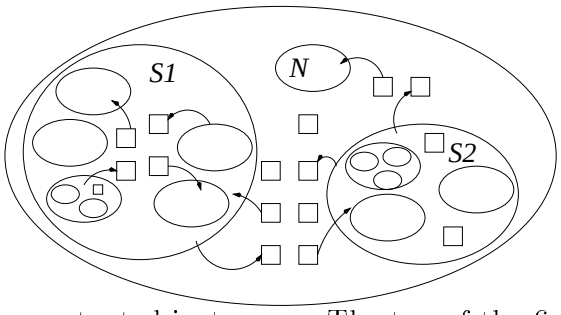
## 4. Objective Linda

Objective Linda [5] integrates generative communication from the Linda model [2] and object–oriented programming methodology. Objects are instances of abstract data types defined in a programming language–independent notation called *Object Interchange Language* (OIL). Language bindings to existing programming languages enable mixed–language interoperability. Because all objects are defined by corresponding abstract data types, communication can be performed without regard of specific implementations which in turn enables interoperability between heterogeneous hardware platforms, too.

Objective Linda configurations are systems of active objects which are instances of abstract data types. When activated, they execute a special *evaluate* routine from their interface the behaviour of which is defined by the type's specification. Every active object has its own object space (OS) which it can reference as its *self* OS in which it can create other active objects performing e.g. subcomputations of a given problem. Besides *self*, every active object knows another OS called its *context* denoting the environment in which it is located. The *context* OS is used to communicate with other active objects and to create active peers in order to perform tasks independently from their creator. This system structure yields a hierarchy of nested OS providing the hierarchical abstractions in which single agents can be treated in the same manner as complex configurations.

Communication in Objective Linda is performed by producing and consuming objects. Objects are selected for consumption from an object space on the basis of their type and the observer functions from their interface. This matching mechanism has significantly improved expressive power compared to the original Linda model. Hence, Objective Linda's communication is generative which in turn allows to model dynamic configurations where agents enter and leave systems without impairing their ability to communicate. Consequently, newly programmed agents can enter running configurations, too (Decentralization). Generative communication also provides encapsulation of active objects because their interface routines are never called while active. Instead, all communication is un-

der control of the *evaluate* routine via generation and consumption of objects. As all communication is performed in a uniform way and as all agents are uniformly modelled, the Objective Linda coordination model is quite homogeneous. Because the generative communication operations are orthogonal to computations performed inside the agents, it is easy to separate between the concerns of inter–agent and intra–agent actions.




The figure beside shows an example. Here, a workstation cluster concurrently performs computations using Objective Linda. Ovals denote active agents, squares denote passive data objects, and arrows illustrate generation and consumption of objects. The system mainly consists of two subclusters $S1$ and $S2$ which communicate locally and with each other via their common *context* object space. The top of the figure shows a new agent $N$ which just entered the configuration.

## 5. Conclusion

The Objective Linda coordination model has already shown its usefulness for various different concurrent systems. Currently, there are two implementations under way, one for a distributed shared memory system intended for parallel applications, and one on top of the PVM library covering the range of (open) distributed systems. A description providing further details can be found in [5].

## REFERENCES

1. J. M. Andreoli, P. Ciancarini, and R. Pareschi. Interaction Abstract Machines. In Gul Agha, Peter Wegner, and Akinori Yonezawa, editors, *Research Directions in Concurrent Object–Oriented Programming*, pages 257–280. MIT Press, Cambridge, Mass., 1993.
2. David Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.
3. David Gelernter and Nicholas Carriero. Coordination Languages and their Significance. *Communications of the ACM*, 35(2):96–107, 1992.
4. ISO/IEC JTC1/SC21/WG7. Reference Model of Open Distributed Processing. Draft International Standard ISO/IEC 10746–1 to 10746–4, Draft ITU–T Recommendation X.901 to X.904, May 1995.
5. Thilo Kielmann. Object–Oriented Distributed Programming with Objective Linda. In *Proc. First International Workshop on High Speed Networks and Open Distributed Platforms*, St. Petersburg, Russia, 1995.
6. Microsoft Corporation. The Component Object Model. *Dr. Dobbs Journal*, 12 1994.
7. Object Management Group. The Common Object Request Broker: Architecture and Specification. OMG Document Number 93.12.43, 1993.
8. Michael Papathomas. Concurrency in Object–Oriented Programming Languages. In O. Nierstrasz and D. Tsichritzis, editors, *Object-Oriented Software Composition*, chapter 2, pages 31–68. Prentice Hall, 1995.