

Coordination languages as communication components

Selma MATOUGUI
ENST-Bretagne
Technopole Brest Iroise
BP 832 - 29285 Brest Cedex
Selma.matougui@enst-bretagne.fr

1) Introduction

Massively parallel and distributed systems open new horizons for large applications and present new challenges for software technology. The number of present processors increased and a new problem appeared: how coordinate the cooperation of large number of concurrent active entities [Arbab 98]. It has been proven more advantageous, in several domains, to divide a complex problem in several more simple problems. This is especially true in software development and the coordination paradigm can be seen as an orthogonal way to decrease the complexity of distributed and parallel computer systems.

Programming a distributed or parallel system can be seen as the combination of two distinct activities: the actual computing part comprising a number of processes involved in manipulating data and a coordination part responsible for the communication and cooperation among the processes.

This has led to the design and implementation of a number of coordination models and their associated programming languages. Communication components [Beugnard 00] or Mediums also aim to encapsulate coordination and able separation of concerns in the development of applications using two different entities components and mediums.

The rest of this article is organised as follows: the second section gives some coordination definitions and problems and makes difference between coordination models and languages. Section 3 describes three approaches for coordination. Section 4 describes mediums and the conclusion makes mapping between mediums and coordination languages.

2) Coordination

2.1) Definitions and problems

The concept of coordination is by no means limited to Computer Science. Malone characterises coordination, in [Malone 94], as an emerging research area with an interdisciplinary focus, playing a key issue in many diverse disciplines such as economics and operational research, organisation theory and biology. Consequently, there are many definitions of what is coordination. It can be defined as:

- “The additional information processing performed when multiple, connected actors pursue goals that a single actor pursuing the same goals would not perform.” [Malone 94]

- “The integration and harmonious adjustment of individual work efforts towards the accomplishment of a larger goal.” [Singh 92]
- “The process of building programs by gluing together active pieces.” [Carriero 92]

As we can see, coordination capitalizes two main activities: communication and synchronisation.

We have selected a representative set of coordination problems. For each one we give some hints about how these problems can be managed using a coordination process.

2.1.1 Management of Shared Resources

Managing shared resources is one of the most common coordination problems that occurs in open systems. Whenever multiple activities share some limited resources, a coordination process, which controls the allocation of and/or access to the resources, is needed. The coordination process must: serialize the incoming concurrent requests, select the request to be served, control that all clients will eventually obtain the resource, control security aspects, take care of possible software and hardware failures during the allocation process, etc.

2.1.2 Transfer of Information Between Activities

Producer/consumer relationships occur in software systems whenever one activity produces some information that is used by another activity. A coordination process that controls the transfer of information between activities is needed. The coordination process must: guarantee the physical transfer of information between the entities, control their synchronization, and eventually control the replication of information in case of a replicated transfer.

2.1.3 Activity Synchronization

Execution of activities in a concurrent setting must be synchronized in order to either communicate, or to perform some common operations (i.e. accessing shared resources). A coordination process controlling the synchronization must constraint the order of execution of these activities so that only acceptable execution sequences can ever be used. A coordination process that controls the synchronization between the concurrent activities must basically: serialize concurrent execution requests, and select (by using a pre-defined synchronization policy like for example: mutual-exclusion, multiple readers/only one writer, etc.) the request to be executed.

2.1.4 Group Decisions

It is very common that a group of entities working together are confronted with the problem of making coordinated decisions. These decisions cannot be made in isolation by one of the members, possibly because no individual has enough authority, competence or information to decide. A coordination process that controls the group decision must: ensure that all the entities will receive an invitation to participate in the group decision, implement a decision scheme, announce the final decision to the group, and decide what to do in case of failures of the entities which participate.

2.2) Model and language

Coordination can be expressed in terms of coordination models and languages. In the following, we try to clarify these two different notions. For coordination models we prefer the following definition:

“A coordination model is the glue that binds separate activities into an ensemble.” [Carriero 92]

In other words, a coordination model provides a framework in which the interaction of individual agents can be expressed. Coordination models are usually characterised by the components out of which they are built: Coordination Entities (what is coordinated?), Coordination Media (where is the coordination?), and Coordination Laws (what is the technique used for coordination?)

However, a coordination language is:

“The linguistic embodiment of a coordination model.” [Carriero 92]

Thus, a coordination language should combine two orthogonal models: one for coordination (the inter-agent actions) and one for computation (the intra-agent actions).

3) Coordination languages and models

This third section aims to present three approaches for coordination divided in three non-exclusive categories: data-driven, control driven and component approach.

3.1) Data-driven

Almost all coordination models belonging to this category have evolved around the notion of a Shared Dataspace. A Shared Dataspace is a common, content-addressable data structure. All processes involved in some computation can communicate among themselves only indirectly via this medium. The activity in a data oriented coordination application tends to center around a substantial shared body of data; the application is essentially concerned with what happens to data. In this category, coordination models and languages are said endogenous; they provide primitives that must be incorporated within a computation for its coordination. The most representative model in this category is Linda [Carriero 89].

Linda

Linda is historically the first member of the family of coordination languages. It provides a simple way of separating computation from communication concerns. Linda is based on the so-called generative communication paradigm: if two processes wish to exchange some data, then the sender generates a new data object (referred to as a tuple) and places it in some shared dataspace (known as a tuple space) from which the receiver can retrieve it. Linda is in fact not a fully edged coordination language but a set of some simple coordination primitives. In particular, `out(t)` is used to put a passive tuple t in the tuple space, `in(t)` retrieves a passive tuple t from the tuple space, `rd(t)` retrieves a copy of t from the tuple space and `eval(p)` puts an active tuple p in the tuple space.

Linda has inspired the creation of many other similar languages, some are direct extensions to the basic Linda model but others differ significantly from it. These derivatives aim to improve and extend the basic model with multiple tuple spaces, enforcement of security and protection of the data posted to the tuple space, etc.

3.2) Control-driven

In control-driven or process-oriented coordination languages, the coordinated framework evolves by means of observing state changes in processes and, possibly, broadcast of events. The activity in a control-oriented application tends to center around processing or flow of

control and, often, the very notion of data, as such, simply does not exist; such an application is essentially described as a collection of activities that consume their input data, and subsequently produce, remember, and transform “new data” that they generate by themselves. Processes communicate with their environment by means of clearly defined interfaces, usually referred to as input or output ports. These models are said exogenous, they encourage the development of coordination modules separately and independently of the computation modules they are supposed to coordinate. One of the coordination models and languages that clearly demonstrates the separation of computation vs. coordination is Manifold [Arbab 93].

Manifold

Manifold is a coordination language for managing complex, dynamically changing interconnections among sets of independent, concurrent, cooperating processes. The processes that comprise an application are either computation or coordinator processes. Computation processes can be written in any conventional programming language. Coordinator processes are distinguished from the others in that they are written in the Manifold language. The purpose of a coordinator process is to establish and manage the communications among other (computation or coordinator) processes. The only entities that the language recognizes are processes, ports, events, and streams (which are asynchronous channels), and the only control structure that exists is an event-driven state transition mechanism. Programming in Manifold is a sequence of dynamically creating process instances and dynamically (re)connecting the ports of some of these processes via streams, in reaction to observed event occurrences.

3.3) Towards a component approach

We have seen the two main approaches that classify coordination models and languages and we present in this section a new one: the component approach. It is closer to the control-driven and consists in abstracting coordination in reusable elements coordinating components which exposes only their interfaces. CoLas [Cruz 99] is one of the models illustrating this category.

CoLas

The CoLaS coordination model is built of two kinds of entities:

- The participants or components: they are active objects that have control over concurrent message invocations.
- The coordination groups: they are entities that specify and enforce the coordination of a group of participants in the realization of a common task

A Coordination is specified with a coordination Group composed of five elements: 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.
- *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. It can be global for the entire group or local for each participant.
- *The Cooperation Protocol*: defines implications between participant actions.
- *The Multi-Action Synchronizations*: specifies synchronizations constraints over message exchanged between participants.

- *The Proactions*: specifies coordination actions that must be enforced by the coordination group independently of messages exchanged by participants assuming that a certain condition holds.

The last three elements of this model are specified using rules and compiled to be implemented.

4) Communication components

An interaction or communication component [Cariou 00] (also called medium) is the reification of an interaction, communication or coordination system, protocol or services into a software component. These interaction protocols or systems implemented or integrated in mediums are various in type and complexity: an event broadcast, a consensus protocol, coordination through shared memory, a multimedia stream broadcast or a vote system for instance. All these kinds of interaction abstractions are handled in the same way through mediums. A distributed application is then built by interconnecting ``conventional'' components with mediums that manage their communication and distribution.

An interaction component is first of all a component with main properties:

- It is an autonomous and deployable software entity.
- It clearly specifies the services it offers and those it requires. This allows the use of a component without knowing how it works.
- It can be combined with other components.

It is by consequence not important to know of what or by what they are made. In an application, the interaction or communication concern is managed by the mediums and the functional concern is localized in the classical components. This allows a good separation between these two concerns. A medium is by nature potentially distributed.

A number of mediums have already been specified:

Vote medium:

The vote medium allows a communication between two kinds of components. Some components initiate votes: they propose a set of values from which the other components (the voters) would choose an element. The initiator of the vote specifies the maximum duration of the vote during which the voter components are allowed to vote. The vote is not mandatory. The vote service blocks until the vote is finished (i.e. until all the voters have voted or that voting time has run out). Several votes can take place simultaneously.

Reservation medium

The reservation medium manages a set of reservation identifiers (seats in a plane, car park places...) that can be reserved or cancelled by a group of components. Some components can observe the state of the medium: after any reservation or cancel, they are informed on the new number of available reservations.

Point to point medium

Two components can be connected to the medium: one sender and one receiver. The communication is one-way (from the sender to the receiver). It looks like a ``mailbox'' communication: the messages are sent in an asynchronous way. The sender and the receiver are never blocked. The two connected components must be different. The messages are ordered; they are read in the order they are sent.

Linda medium.

The Linda coordination model is based on a shared data space, containing tuples. A tuple is a sequence of fields that contain either a value or a variable. The tuples in the tuple space only contain values. The communication is based on the pattern matching between a tuple template (a mix between values and variables) and tuples of the space. A template tuple T1 matches a tuple T2 if they have the same number of fields and if T1 contains the same values at the same places as T2. The others fields of T1 are variables.

A medium is especially designed to be reusable. It offers services that can be easily adapted depending on the context of use. Non-functional properties such as the size of a system can simply be taken into account by implementation variants. For instance, the Linda medium can have a centralized implementation of the shared tuple space or a distributed one without affecting its clients in any way.

5) Conclusion

Endogenous models are sometimes more natural for a given application. However, they generally lead to intermixing of coordination primitives with computation code, which confuses the semantics of computation with coordination. This intermixing tends to disperse communication/coordination primitives throughout the source code, making the cooperation model and the coordination protocol of an application nebulous and implicit.

On the other hand, exogenous models (including component approach) encourage development of coordination modules separately and independently of the computation modules. Consequently the effort invested in the design and development of the coordination component of an application can manifest itself as “pure coordinator modules” which are easier to understand and can also be reused in other applications. However, they stay limited to rule compilation style. Between these two techniques, communication components can manifest themselves as a good intermediate

Mediums share the same goals as the coordination approaches; they both aim to abstract or encapsulate coordination to have a better separation of concerns and to increase reuse. However, medium’s asset is that the technique used for this purpose is not of a great importance, medium gives opportunity to mix coordination techniques and they specify a frontier which separates clearly the coordination from the entities to be coordinated. The medium approach offers a more flexible and more abstract vision of interactions available between elements.

As a consequence, mediums can be seen as an integrator architectural concept with:

- Unspecified abstraction level that comprises:
 - A good specification
 - Frontiers that limit the responsibilities of the medium and what are the data and tasks that it has to manage
 - Deployment in many sites: mediums are naturally distributed
 - Coherence
- Implementation variants: the underneath technique is not of a great importance: we can find either data driven mediums or control driven mediums.

6) Bibliography

- [Arbab 93] F. ARBAB, I. HERMAN and, P. SPILLING ; *An Overview of Manifold and its Implementations*. Concurrency: Practice and Experiences 5(1), 1993, pp.664-677
- [Arbab 98] F. ARBAB, G.A. PAPADOPOULOS ; *Coordination models and languages*. Software Engineering (SEN), SEN-R9834 December 31, 1998.
- [Beugnard 00] A. BEUGNARD, *Communication Services as Components for Telecommunication Applications*. 14th European Conference on Object-Oriented Programming (ECOOP'2000), Objects and Patterns in Telecom Workshop, Sophia Antipolis and Cannes (France), 2000.
- [Cariou 00] E. CARIOU, A. BEUGNARD ; *Specification of Communication Components in UML*, at The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000), H.R. Arabnia, CSREA edition, pp.785-791, vol.2, Las Vegas, 2000.
- [Carriero 89] N. CARRIERO, D. GELERNTER ; *Coordination Languages and their Significance*. Communication of the ACM 32 (4), 1989 pp. 444-458.
- [Carriero 92] N. CARRIERO, D. GELERNTER ; *Linda in context*. Communication of the ACM 35 (2), 1992 pp. 97-107.
- [Ciancarini] Paolo CIANCARINI ; *Coordination Models and Languages for Parallel Programming*.
- [Cruz 99] Juan Carlo CRUZ, Stéphane DUCASSE ; *Coordinating Open Distributed Systems*. Future Trends in Distributed Computing Systems '99 FTDCS'99
- [Günter 98] Manuel GUNTER ; *Explicit Connectors for Coordination of Active Objects*. Master Thesis of Faculty of Science, University of Bern 1998
- [Malone 94] T. W. Malone and K. Crowston, *The Interdisciplinary Study of Coordination*, ACM Computing Surveys 26, 1994, pp. 87-119.
- [Singh 92] B. SINGH ; *Interconnected Roles (IR): A Coordinated Model*. Technical Report CT-84-92, Microelectronics and Computer Technology Corp., Austin, TX, 1992.