

A Coordination Process Based on UML and a Software Architectural Description

P. Inverardi
Dipartimento di Matematica
Universita' dell'Aquila
Via Vetoio, 1 - Italia
inverard@univaq.it

H. Muccini
Dipartimento di Matematica
Universita' dell'Aquila
Via Vetoio, 1 - Italia
muccini@univaq.it

Abstract *Coordination models and SA descriptions work in different domains, but there are (at least) two strict relationship between them: i) SAs represent an high level description of the system while coordination models (languages) are closer to specification level; ii) coordination models are specialized to describe process interaction in a concurrent environment, SA descriptions contain inter-components communication descriptions. In this work, we are going to describe how SA description and UML can be integrated in a UML software development process with the aim to define a coordination process able to describe, analyze and validate system coordination properties that might be then specified with a suitable coordination language. A real-world case study will be used to discuss, in detail, the approach previously introduced.*

Keywords: Coordination process, UML, Software Architecture

1 Introduction

In recent years, the growing of system complexity and the need to manage highly complex concurrency between different processes has represented an important issue in different fields of research.

Coordination models have been introduced with the goal of finding solutions to the problem of managing the interaction among concurrent programs or activities. Several coordination languages (e.g., Occam, LOTOS) based on theoretical models like CSP, CCS, process alge-

bra or π -calculus have been proposed, putting in evidence the difficulty to manage large number of concurrent activities. Further research put forward the need of explicit *coordination models and languages*. *Data-oriented* or *control-oriented*, *endogenous* or *exogenous* coordination models (e.g., Linda, Gamma, MANIFOLD) have been built, abstracting away the details of computation and focusing on the interactions [2].

On the other hand, in the software engineering field the focus is continuously moving towards systems of larger dimensions and complexity. Software production is becoming more and more involved with distributed applications running on heterogeneous networks. As a result, applications are increasingly being designed as sets of autonomous, decoupled components, promoting faster and cheaper system development. In this context *Software Architecture* (SA) can play a significant role, improving the dependability of large complex software products, while reducing development times and costs [4]. The originality of the SA approach is to focus on the overall organization of a large software system (the glue) using abstractions of individual components and explicitly modeling their interaction. This approach makes it possible to design and apply tractable methods for the development, analysis, validation, and maintenance of large software systems.

Coordination models and SA descriptions work in different domains, but there are (at

least) two strict relationship between them:

- SAs represent an high level description of the system while coordination models (languages) are closer to specification level.
- Coordination models are specialized to describe process interaction in a concurrent environment, SA descriptions contain inter-components communication descriptions.

At the SA description level, many important design choices related to the way components interact, are already taken. As a result (see Figure 1) we have that SA level information influence the static and dynamic structure of the implemented system and drive the coordination model specification.

In this work, we are going to describe how SA description and UML [13] can be integrated in a UML software development process [11] with the aim to define a *coordination process* able to describe, analyze and validate system coordination properties that might be then specified with a suitable coordination language.

Although UML original purpose was for detailed design, UML extension mechanisms makes it potentially applicable much more broadly. Recent works [11] describe how UML Diagrams can correctly and completely describe each step in the software development life-cycle, from Requirements capturing to Design model description passing through a Software Architectural description [14, 5, 7].

In the rest of the paper, we concentrate on the requirements phase and show how the architectural description can complement the UML ones to allow an architectural validation step. In particular, we will see that the dynamic description that the SA specification provides allows for validating coordination policies that could not be validated otherwise.

A more comprehensive description of the integration methodology we propose can be found in [9].

2 The approach

The coordination process we are going to propose is described by UML diagrams and Software Architectural modeling tools and is based on UML software development process and SA description. Requirement understanding and capturing model is the first step in our coordination process. The UML [13] approach to identifying system requirements is based on Use Case Diagrams; *use cases* represent a possible way of using the system while *actors* are who or what (humans or a subsystems) carry out use cases. Each user needs several different use cases, each representing the different ways he or she uses the system.

To achieve a more precise understanding of the requirements and structure them for reuse and maintenance an analysis model can be described [11] using *analysis classes* and interacting analysis objects. Analysis classes describe how a specific use case is realized in terms of "abstract" cooperating classes and always fit one of three basic stereotypes: boundary, control or entity;

- boundary classes represents abstractions of windows, forms, communication interfaces;
- entity classes reflect logical data structure;
- control classes represent *coordination, sequencing, transactions and control of other objects* and are often used to encapsulate control related to a specific use case [11].

Each use case can be modeled by analysis classes. Each class may participate and play roles in several use-case realizations. A class diagram (of analysis classes) can be drawn to indicate which use-case realizations a class participate and plays roles in. This diagram gives an high-level *static* description of the classifiers implementing the use cases but it does not give information on *how the system evolves* in terms of use cases interactions.

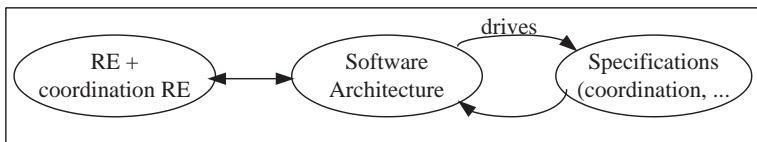


Figure 1: SA description drives Coordination models

The sequence of actions in a use case begins when an actor invokes the use case by sending some form of message to the system; the boundary class receives the communication request, sends the request to the control class that coordinates the various activities and lets the involved objects interact to realize the use case. Interaction Diagrams can model the chronological sequences of interactions but only as a sort of *coordination specification constraints*; since a limited amount of scenarios can be manually depicted.

The (global) dynamic model obtained by the use of Software Architectural description can provide a model of the global system interaction. For example, using an operational transition-based description of an SA, like the Cham [10], the system behavior is modeled as a global LTS; each transition represents a possible interaction between different components and each path a possible execution scenario coordinating different activities.

The LTS model is intended to be the coordination model while Interaction Diagrams (at the requirement level) the coordination specification. The LTS model could then be used to check the system correctness with respect to coordination requirements (coordination specifications).

The idea is to define a mapping between Analysis model and SA topology; actors in the Use-Case Diagrams represent a suitable abstraction of active components in the SA description; control classes can identify coordination components; control classes attributes can identify communication channels. Other classes can be hidden at the SA level or mapped to other components.

3 An example

Let consider the Tele Remote Medical Care System case-study (TRMCS) [3] requirements:

the system USER can (non deterministically) choose to SEND AN ALARM message or a CHECK MESSAGE to a component named ROUTER. The Router is always WAITING FOR ALARMS and CHECKS. If an alarm message is received by Router (and correctly processed by the Server component, to which the alarm is forwarded), an ACK MESSAGE is sent back to the User. If a check message is received, by Router, it is STORED in a local file; in the case an ERROR OCCUR, this has to be handled by the SERVER. The service has to be ever guaranteed (every day); the termination of the service is due; only ENABLED USERS can take advantage of this service; hardware fault tolerance must be managed; more than one user can run concurrently.

User(s), Router and Server are the system actors while Alarm, Check and Ack send and receive operations, ErrorHandling, CheckUser, CheckRouter and StoreInLog are the main functionalities. A Use Case Diagram can be used to depict this configuration (Fig. 2) while (analysis level) Class Diagram (Fig. 3.a) shows boundary, control and entity classes "implementing" them.

With a process oriented analysis, Alarm, Check and Ack processes are identified and the following coordination rules are taken out:

1. An Alarm message sent from User has to be followed by an acknowledgment message; it represents coordination between dependent processes;
2. Check and Alarm messages have to be concurrently managed. This is the coor-

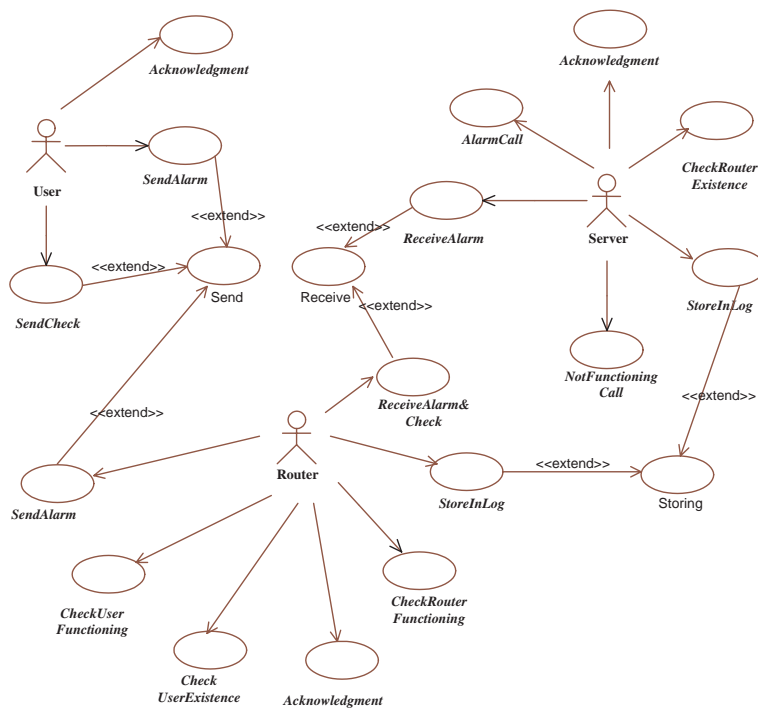


Figure 2: Trmcs Use Case Diagram

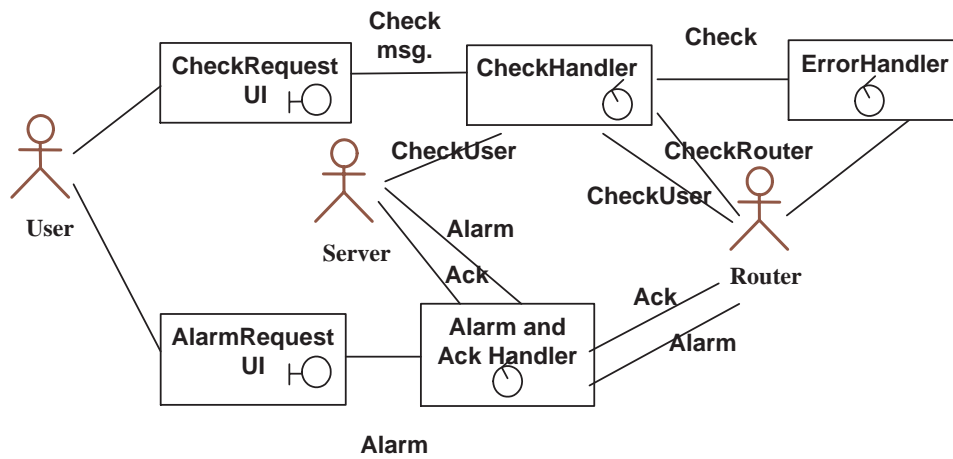


Figure 3: Analysis model of the Trmcs

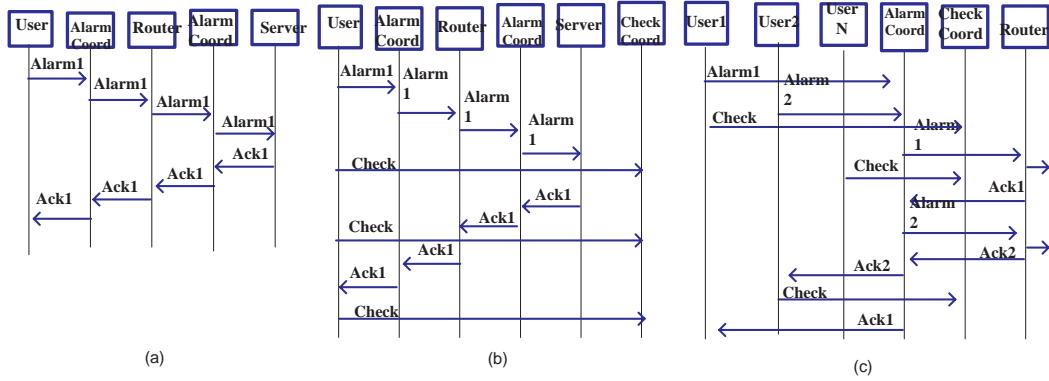


Figure 4: a) Server and Ack threads; b) Server and Check threads; c) N Users interacting

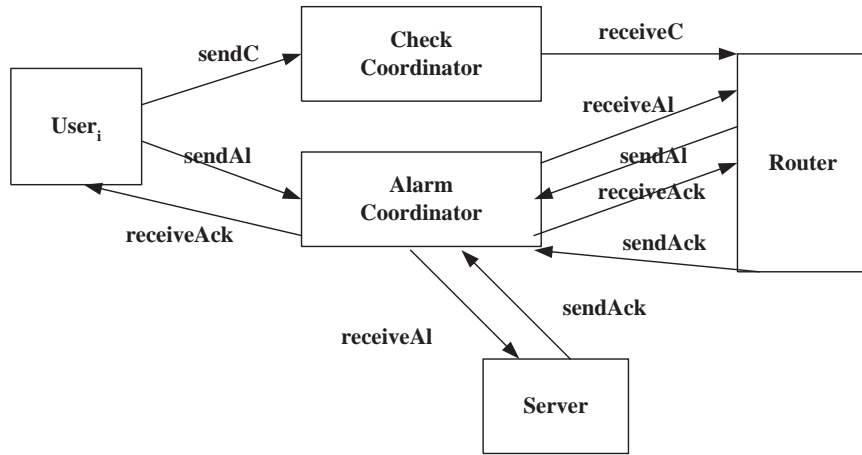


Figure 5: Trmcs Components and Connectors

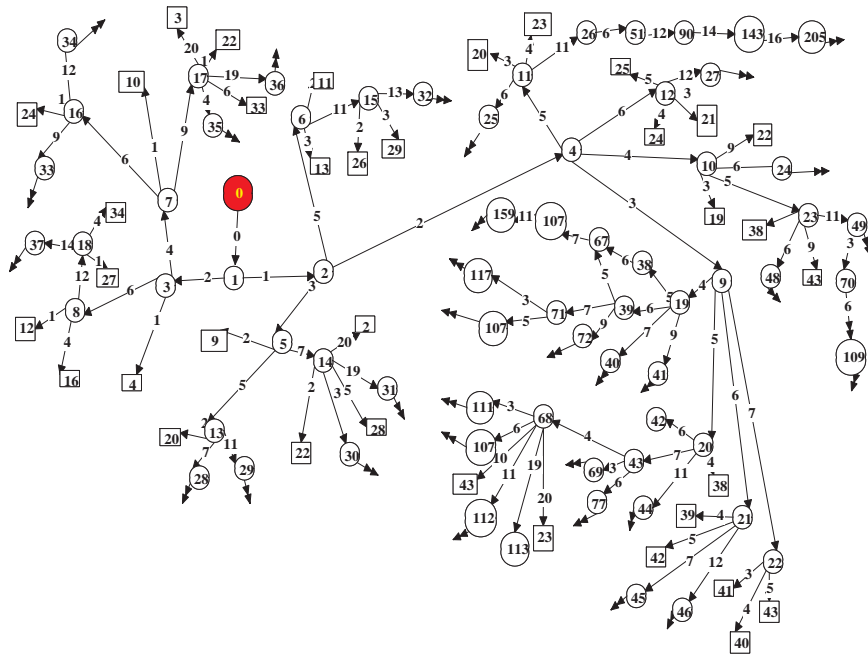


Figure 6: Trmcs LTS

dination between different threads;

3. The system can be used concurrently by n users; it represents coordination between different instance of different threads.

Figures 4.a, b, c represent possible dynamic scenarios conforming to the first, second and third described coordination policies respectively; using Interaction Diagrams we can draw interesting scenarios, without a global concurrency view.

SA topological description (Fig.5) can be extracted by class diagrams where User, Router and Server become active SA components and Check, Alarm, Clock and Time Handler coordination components, hiding the others. Modeling the system behavior via a transitional system we can generate a global LTS model describing how the coordination policies are integrated in the whole environment. Considering the case when two users are working on the TRMCS system, we have built the LTS model of the system dynamics. A portion of it is shown in Figure 6 while the whole LTS is around 500 states.

Each LTS complete path represents a system

global scenario, i.e., a possible combination of Alarms, Checks, Clocks and Acks messages exchanged by the system components. Scenarios could be validated by the scenario defined in Fig. 4.

References

- [1] R. Allen and D. Garlan. A Formal Basis for Architectural Connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, July 1997.
- [2] Farhad Arbab. What Do You Mean, Coordination? In the March '98 Issue of the *Bulletin of the Dutch Association for Theoretical Computer Science (NVTI)*. Available at: <http://www.cwi.nl/farhad/>.
- [3] S. Balsamo, P. Inverardi, C. Mangano, F. Russo. Performance Evaluation of a Software Architecture: A Case Study. *IEEE Proc. IWSSD-9*, April 1998, Ise-Shima, Japan.

- [4] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. SEI Series, Addison-Wesley 1998.
- [5] G. Booch. Software Architecture and the UML. Slides available at: <http://www.rational.com/uml/index.jttml>.
- [6] Darwin, an Architectural Description Language. Web site: <http://www-dse.doc.ic.ac.uk/research/darwin/darwin.html>.
- [7] H.E. Eriksson and M. Penker. UML Toolkit. John Wiley & Sons, Inc. publishing.
- [8] M. Fowler and D. Scott. UML Distilled. Addison Wesley publiser.
- [9] P. Inverardi and H. Muccini. Coordination models and Software Architectures in a Unified Software Development Process. Submitted to the fourth International Conference on Coordination Languages and Models (Coordination 2000). On-line at: <http://univaq.it/~inverard>.
- [10] P. Inverardi and A.L. Wolf. Formal Specifications and Analysis of Software Architectures Using the Chemical Abstract Machine Model. *IEEE Transactions on Software Engineering*, 21(4):100–114, April 1995.
- [11] I. Jacobson, G. Booch and J. Rumbaugh. The Unified Software Development Process. Addison Wesley, Object Technology Series.
- [12] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan and W. Mann. Specification and Analysis of System Architecture Using Rapide. *IEEE TSE, Special Issue on Software Architecture*, 21(4):336-355, April 1995.
- [13] Rational Corporation. Uml Resource Center. UML documentation, version 1.3. Available from: <http://www.rational.com/uml/index.jttml>.
- [14] Jason E. Robbins, Nenad Medvidovic, David F. Redmiles and David S. Rosenblum. Integrating Architecture Description Languages with a Standard Design Method. *Proc. 20th Int'l Conf. on Software Engineering* Apr. 1998, pp. 209-218.