Specification and Design of Component-based Coordination Systems by Integrating Coordination Patterns[1]

Pedro L. Pérez-Serrano
QUERCUS Software Engineering Group,
Computer Science Department,
University of Extremadura
Escuela Politécnica, Avda. Universidad, S/N
10071 Cáceres (Spain)
+34-927257173
plperez@unex.es

Marisol Sánchez-Alonso
QUERCUS Software Engineering Group,
Computer Science Department,
University of Extremadura
Escuela Politécnica, Avda. Universidad, S/N
10071 Cáceres (Spain)
+34-927257807
marisol@unex.es

ABSTRACT
Rewriting logic has been revealed as a powerful tool to represent concurrent and state-transitions aspects in a declarative way, providing an adequate environment to specify and execute system representations. Moreover, rewriting logic is reflective, allowing for the definition of operations that transform, combine and manipulate specification modules by making use of the logic itself. Taking advantage of these capabilities, this paper presents a set of tools based on the rewriting logic language Maude to express the specifications of component-based systems with important coordination constraints, where coordination aspects are treated as separate components from functional ones. This representation allows for the testing of the system behavior from the early stages in the development process by executing the specifications. In addition, the development of basic coordination patterns using UML is presented to describe the coordination relationships between components in any system, providing a standard notation that complements the tools of the proposal.

Categories and Subject Descriptors
D.2.1 Requirements/Specifications (D.3.1)

D.2.2 Design Tools and Techniques

D.2.4 Software/Program Verification (F.3.1)

General Terms

Performance, Design, Standardization, Languages, Verification.

Keywords
Coordination Requirements, Behavior Simulation, Accordance Checker, Coordination Patterns.

1 INTRODUCTION
The need to develop more and more complex systems has enabled the improvement in languages and models to manage the coordination constraints between system components, promoting reusability, and the flexibility to change the interaction policies between components by means of the separate treatment of functional and coordination concerns. However, a serious limitation of these models, with regard to their usability, is that they do not provide support to manage the coordination constraints from the early stages in the software life cycle. That makes the adoption of a particular coordination model or language more difficult during the detailed design or implementation phases, due to the fact that coordination aspects are implicit and dispersed along all the components present in the system model, and now it is necessary to express explicitly the coordination aspect in a separate way from the functional one, to be able to apply a specific coordination model. Dealing with the specification of complex systems, coordination models should be encompassing a methodology that supports the separation of concerns throughout the whole software development process. Such methodology should be based on the use of formal techniques providing strictness and allowing the demonstration of properties that the specifications must satisfy.

Among the varieties of logics on which the formal specification techniques are based, rewriting logic has been revealed as a well suited base to express the system specifications in concurrent and state transition environments. Moreover, rewriting logic supports a wide spectrum of applications for developing prototypes, parallel execution and transformations. In addition, the reflective capability of this logic makes it a powerful tool to express the system specifications and to develop applications by transforming, checking and manipulating the logic itself.

In particular, this work focuses on the execution of specifications from the early stages of the life cycle, to validate and test the system behavior imposed by the coordination constraints between components. The use of Maude language [1] supporting rewriting logic is proposed. This choice is motivated by Maude's capabilities not only to specify and execute the system coordination requirements, but also to construct tools that manipulate the system specifications and transform them from a representation which is closer to designers to a more detailed and

[1] This work has been supported by the project CICYT under grant TIC 02-04309-C02-01.

complex representation that specifies the mechanisms needed to perform the coordination concerns.

This supposes accepting specifications which adopt the syntax of a coordination model, and which transform them into representtations detailing how the coordination mechanisms act to simulate the coordinated behavior by executing this last representation.

As the detailed representation can be refined along the software development process with features of design and implementation, a tool to check the accordance between representations of different abstraction levels is required. This tool manipulates specifications, and it is developed in the rewriting logic itself, making use of the reflective capability.

With these aims, we have developed COFRE (Coordination Formal Requirements Environment) [2], a set of tools considering all the above features, providing a methodology to make the system specification easy based on formal and graphic techniques, and dealing with coordination constraints from the early stages in the software development process. As regards benefits, changing and reusing components and coordination patterns from requirements are more systematic and pleasant tasks; in addition, the system behavior can be simulated by executing the formal specifications, simplifying the validation process, and the use of a model checker allows for the verification of the accordance between specifications in different abstraction levels.

In order to provide a standard notation to specify the system coordination constraints, we have developed a set of basic coordination patterns in UML that can be combined to express any kind of coordination constraints between the components of a system, even the most complex. Although these patterns are proposed to complete COFRE, they can also be used independently of this set of tools and the coordination model or language in which the system will be implemented.

The structure of this paper is organized as follows: In Section 2 the motivation for this work is presented. The steps of our proposal which makes use of a language based on rewriting logic to represent the system model along the software development process are described in Section 3. Section 4 gives an overview of the work in progress, describing the design of coordination patterns and their integration into the proposal. Finally, Section 5 provides the conclusions.

2 MOTIVATIONS

In recent years a wide range of tools combining both graphical and formal techniques have been developed with the aim of making software development easier. These tools can express in a detailed way the static and dynamic aspects of the system. But, initial requirements making use of these techniques are expressed in a global way which makes it difficult to adopt an architectural or design technique based on the separation of concerns (including coordination constraints). Particularly, in coordination environments, the adoption of a coordination language requires the separation of the specification of the coordination behavior from the functional one. But this task is delegated to designers or programmers starting from a conceptual model where these concerns are mixed. By avoiding this problem, the development processes are made more agile and consistent. Consequently, we proposed a set of tools named COFRE, based on the use of a rewriting logic language that tries to separate the functional concerns described for each component from concerns related to the interactions between components starting from the requirements definition in the development process.

Simulation by means of the execution of the model is the technique that best permits the observation and testing of the system dynamic properties. Often the simulation techniques by formal specifications execution, named *animation* techniques, required the translation of the specification to an imperative programming language like C++ or Java to be executed [10,11]. However, the different abstraction levels of the languages used to specify and to animate the model can provoke lack of precision and fidelity between both representations. This justifies the use of formal techniques allowing the execution of specifications to check the system behavior when implementation details have not yet been described.

Particularly, in coordination systems, special attention must be paid to guaranteeing that the final behavior obtained in the composed application is semantically coherent. That means verifying whether gluing together a coordination policy and a set of components in an application (which can have been coded and checked separately) will produce the expected behavior, and whether the addition or change of coordination constraints will produce conflicts with the current behavior.

Because the adoption of a coordination model means specifying the system constraints in a more detailed way, it is necessary to guarantee the accordance of this representation with regard to the initial requirements definition by means of a verification process checking that the interaction between the system components are maintained.

3 COFRE: SPECIFYING COORDINATION SYSTEMS

In this section, the above topics are focused on, proposing COFRE to make the specification and validation process easier for both software engineers and users. This proposal is based on the use of the formal language Maude to express the different representations of the system specifications and as a language in which the tools that manipulate and transform these specifications are implemented.

Maude is an executable algebraic language based on rewriting logic, that describes the specifications in a concurrent and non-deterministic way. The specifications can be executed by means of reducing terms in equations and rewrite rules performed by the interpreter provided by the language, which facilitates its use for prototyping and for checking the specifications behavior.

Maude allows for the definition of functional modules containing operations and equations, system modules also containing rewrite rules and object modules that are system modules allowing the specification of features concern to O-O paradigm. The use of rewrite rules is particularly appropriate to represent state transitions in systems of concurrent objects, where a configuration formed by the object instances and the current messages present in the system determine the system state in each moment. Maude provides specific definitions and operations to efficiently manage system configurations.

These are the capabilities that make the use of Maude appropriate to represent the different abstraction level specifications of the

system as well as to manipulate the specifications themselves, being the language in which part of the tools composing COFRE are implemented.

The methodology under COFRE proposes the use of IRD diagrams to represent the system components and their interactions from requirements analysis. IRDs have a corresponding specification in Maude language. However, this representation is not executable, because a specific coordination model needs to be adopted, but it is necessary to check the accordance of subsequent detailed specifications with regard to the initial requirements. COFRE adopts a specific coordination model and generates, starting from this initial specification, the equivalent specification making use of the coordination model syntax. This specification can be transformed in a more detailed Maude representation, expressing all the artifacts to represent the coordination aspects. This more detailed representation can be executed to simulate the system coordinated behavior [3], contributing to the system validation. Moreover, the specification adopting the coordination model can be verified with respect to the initial formal representation of the IRD to determine their accordance making use of the accordance checker, developed for this purpose. Figure 1 shows a schematic representation of the method.

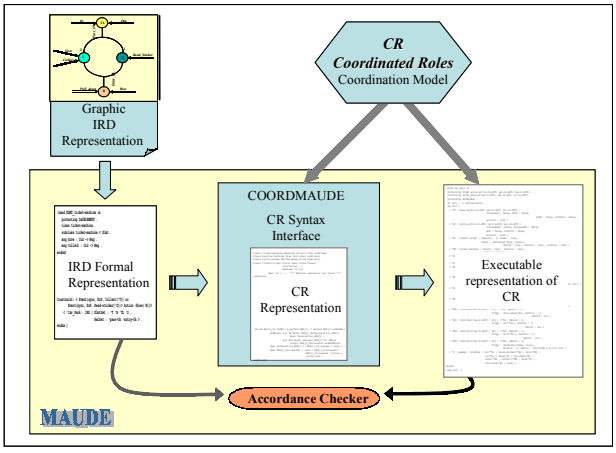

Figure 1. Schematic representation of COFRE method steps

Figure 1 shows the main steps of COFRE:

1. The main system components, their external interface and the interaction rules of the system are expressed using Interelement Relation Diagrams (IRDs). IRDs are used to specify the cooperation rules (coordinated interactions) between components in a graphical way.

2. The system IRD has a Maude representation allowing for verification of the agreement of a detailed Maude specification with regard to the original requirements expressed in the IRD.

3. The system specification can adopt a specific coordination model making use of its syntax, and this specification has a Maude representation of the coordination details permitting the model execution.

4. The behavior simulation of the system can be tested and validated after each iteration in the refinement of the development process.

5. The accordance with regard to the requirements expressed in the system IRD can be checked.

The transition to the design stage is made by adopting the exogenous coordination model Coordinated Roles (CR) [4]. CR is inspired by *IWIM* model [5] and based on the *Event Notification Protocols* (*ENP*) mechanism. This mechanism allows a coordinator component to ask for the occurrence of an event in another component, and the notifications can be asked for in a *synchronous* and an *asynchronous* way. The process must be transparent for the components to be coordinated.

Each coordination component imposes a coordination pattern structured as a set of roles. A role represents each of the characters that can be played in a coordination pattern. Behavior components will have to adopt these roles in order to be coordinated. For each role, coordination components specify the set of events required to represent the desired coordination constraints. The binding between coordinators and components to be coordinated is done at run-time via composition syntax.

4 COORDINATION PATTERN DESIGN

Design patterns [6] are common solutions accepted as being correct for specific design problems. They constitute an appreciated tool to improve of the quality of software development.

In order to take the advantage of using design patterns and applying them to the coordination aspects, we are working on the definition of a set of coordination patterns. These coordination patterns are specified using some diagrams of UML [7], to provide a standard representation that facilitates their use. We are starting with the definition of patterns representing basic coordination events and patterns representing the coordinators to develop the solution to problems in coordinated environments, in a level of abstraction independent of the programming language or the platform used.

The coordination patterns are developed with the aim of being widely applied, and with the purpose of being integrated in developing tools making use of UML. However, we propose to integrate coordination patterns in COFRE, to provide a standard notation that improves this set of tools and facilitates its use and its comprehension.

The coordination pattern integration is divided into three phases that are shown in figure 2:

Phase 1: This phase includes several steps:

- Specification of the events and the coordinators in a Class Diagram. Thus, it will define the static part of the system, the functional and singular aspects of each of its components of it. Also, the coordination aspects will be defined, that is, the system coordinated behaviour with the Interaction Diagrams of UML, composed of two kind of diagrams:
  - The Sequence Diagrams, showing the messages temporary ordination of the different objects in the system. The behavior and the role of the coordinator or coordinators in an action will be shown in these diagrams.

– The Collaboration Diagrams, showing the structural organization of the objects, that is, the set of messages sent and received between different objects in a collaboration.

- Generation of the formal definition of the system in Maude. Starting from this representation the sequence of steps proposed in COFRE can be applied to obtain a representation of the system in CoordMaude and to simulate the system behaviour.
- A detailed system specification is obtained independent of the platform and the programming language to be implemented. This solution will be saved in a repository together the documentation generated. The coordinated patterns are proposed to be reused and the documentation generated could be exploited to help in their reutilization.

Phase 2: This phase will consist of the development of a tool (black arrow number 1 in the figure 2) that transforms the Class and the Interaction Diagrams of Phase 1 to the IRD diagrams of COFRE. In this way, the development of the coordinated system is performed in the analysis phase with COFRE and in the design phase with the tools developed in phase 1. Thus, it will be necessary to develop a new tool (black arrow number 2 in figure 2) to check the accordance between the results of both phases.

Phase 3: In this phase, a tool will be developed to allow the inverse process to the explained one in phase 2; that is, a tool that converts the IRD diagram to the corresponding UML diagrams.

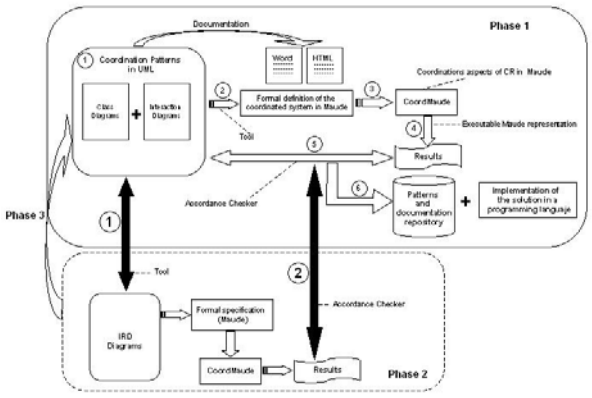

Figure 2. Integration of the Coordination Patterns in COFRE

5 CONCLUSIONS

The system requirements representation must be done performed making use of formal techniques allowing the validation of system requirements from the early stages in the development process, by executing specifications. This is especially important in coordination environments, where it is necessary to guarantee that interactions between system components work properly.

COFRE has been developed for this purpose, providing these advantages:

1. The coordinated interactions between components can be specified independently from component functionality By using IRDs. The formal representation of IRDs in Maude avoids ambiguity, adding precision to the model.
2. The adoption of a specific coordination model is facilitated from early stages in the development process due to the separation of functional and coordination concerns.
3. The features and event notification protocols are represented in Maude to make use of the rewrite engine of the language and with the aim to simulate the system behavior with different configurations.
4. The system representation making use of the CR syntax is possible, using *CoordMaude* that extends Maude, to accept specifications made on CR and generating all the mechanisms needed for the representation of the coordination model in Maude, in order to execute the system specifications.
5. Formal representation of IRDs can be checked with the specifications resulting from applying the coordination model to determine whether the accordance between both representations is maintained.

The definition of basic and generic coordination patterns in a standardized notation like UML allows the system specification to be expressed in an independent way from the tools used for the developing process and the coordination model adopted for the implementation of the system. The specification of complex coordination constraints can be made by combining basic coordination patterns, maintaining the separation of the coordination and the functional aspects of the systems. This separation contributes to making the software development and modification easier.

6 REFERENCES

[1] Clavel, M. Durán, F. Eker, S. Lincoln, P. Martí-Oliet N. Meseguer, J. and Quesada, J. Maude: Specification and Programming in Rewriting Logic. Computer Science Laboratory. SRI International. March'99.

[2] Sánchez-Alonso, M. Murillo, J.M. and Hernández J. COFRE: Environment for Specifying Coordination Requirements using Formal and Graphical Techniques. Journal of Research and Practice in Information Technology, Vol. (36) pp: 231-246, Australian Computer Society Inc. 2004.

[3] Sánchez-Alonso, M. and Murillo, J.M. Specifying Cooperation Environment Requirements using Formal and Graphical Techniques. In *WER'2000, 5th. Workshop on Requirements Engineering,* 2000.

[4] Sánchez-Alonso, M. Clemente, P.J. Murillo, J.M. and Hernández, J. CoordMaude: Simplifying Formal Coordination Specifications of Cooperation Environments. *2nd Workshop on Languages Description Tools and Applications (LDTA'03)*. ENTCS nº 82.

[5] Arbab, F. (1996): The IWIM Model for Coordination of Concurrent Activities.. *1st Int. Conf. on Coordination'96.* LNCS 1061. Springer-Verlag.

[6] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley Professional, 1995.

[7] J. Rumbaugh, I. Jacobson, G. Booch. The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.