

Separation of Quality Concerns in the Development of Distributed Real-time Systems

MohammedReza Mousavi, Giovanni Russello,
Michel Chaudron, Twan Basten, Michel Reniers

Eindhoven Embedded Systems Institute,
Eindhoven University of Technology, Eindhoven, The Netherlands
Email: {m.r.mousavi, g.russello,
m.r.v.chaudron, a.a.basten, m.reniers}@tue.nl

Abstract—This paper addresses the issue of separation of concerns in the design and implementation of distributed real-time systems. It gives an overview on the activities done in the theoretical and practical domains of the SACC (Software Architecture = Components + Coordination) project in order to provide a support for this idea.

Keywords—Distributed Real-time Systems; Middleware; Formal Specification; Aspect Orientation

I. INTRODUCTION

The design of distributed real-time systems is a complex and difficult task. For such systems many functional and non-functional quality requirements (reliability, timeliness, efficiency, and flexibility) need to be reconciled simultaneously. The difficulty arises from the fact that a solution for one of these requirements seems to complicate meeting other requirements. Hence the interaction between different partial solutions to the requirements is one important cause of the complexity of designing such systems.

To simplify the design of such systems, we propose a method that has the following novel features:

- A separation at the specification and design level of quality concerns from the functionality of the system.
- An architecture for the middleware of distribution system that supports separation of multiple quality concerns.
- A provably correct method for getting from the specification to an implementation.

In order to realize this goal in our project, we develop a suite of specification languages as well as

middleware that supports the separation of concerns in the implementation level. An automatic translation from the specification formalisms to the implementation languages will bridge the gap between the two domains. Thus, the project is organized along two themes: theory concerning specification techniques and practice involved with an implementation framework.

In this paper, after the introduction, Section 2 focuses on the current state of the theoretical part and Section 3 describes the main ideas of the implementation theme. Finally, Section 4 gives the concluding remarks and presents the future research directions.

II. CONCEPTS AND THEORY

In this section we give a brief introduction to the model that we use for system design. We explain why it provides better support for separation of concerns than traditional software programming and modeling languages.

A. GAMMA: A Computational Model for Separation of Concerns

GAMMA is an abstract language, based on multiset rewriting on a shared data-space. It is designed to support parallel execution of a program on a parallel and/or distributed architecture [1]. The basic and atomic piece of functionality in GAMMA is the rule. The calculus of GAMMA [2] contains some composition operators to compose rules into programs. However, in our approach, we eliminate all structuring decisions at the level of GAMMA programs, and use the abstract theory as our basic model of component functionality. Since GAMMA rules can interact with the shared data space independently, the models allows for temporal and

spatial decoupling (see Figure 1). Henceforth, our GAMMA model is only concerned with independent basic functionalities of a design and abstracts from several other concerns such as: relative ordering of functionalities, timing, distribution, hardware resources and the level of true concurrency.

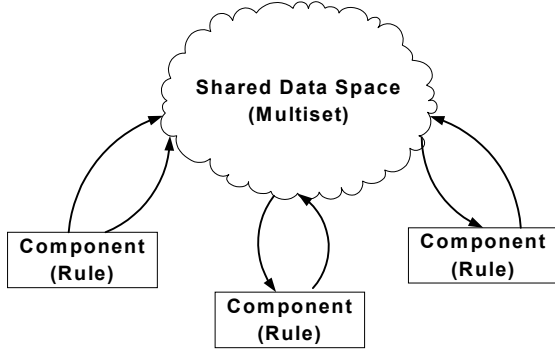


Figure 1. GAMMA shared data space model

B. Other Concerns: Timing, Coordination, and Distribution

In our design method, non-functional aspects are specified in their own tailor-made design language. These non-functional aspects include:

- **Coordination:** Coordination is concerned with ordering, concurrency and synchronization of independent functionalities. In particular, we take a process algebraic approach for formal specification of this aspect. Basic processes (building blocks) of our coordination expressions are GAMMA rules. Our coordination language, which is a slightly modified version of the scheduling language of [3], contains conditional, sequential, parallel, and recursion operators. This leads to a modeling framework that allows changing the modeling language of each layer and still exploiting the formal semantics of other layers in combination.
- **Timing:** We distinguish between two different attributes in the timing domain: individual performance metrics of components vs. end-to-end requirements of an entire system [4]. To model individual performance metrics, we relate single GAMMA rules with their estimated execution time as intervals. End-to-end requirements are specified using a real-time extension of Temporal Logic and are verified against the combination of aspect designs.
- **Distribution:** We specify a distribution pattern as a mapping from data types and rules to

physical locations. The distribution pattern may be static in time or may dynamically change over time.

In [5], a general framework for the specification of these three aspects, together with GAMMA functionality, is sketched, and the framework is applied on an elevator case study. In [6], we give a precise semantics for the (GAMMA + Coordination) + Timing paradigm.

III. EXPERIMENTAL VALIDATION: ARCHITECTURE OF AN IMPLEMENTATION

In this section, we describe the current status of the part of the research that is involved in developing implementation techniques that support the design method presented in the preceding sections. The main goal in the implementation domain is to design and implement a distributed middleware that can deal with several aspects of design as orthogonal issues that can be uploaded according to their respective design models.

A. Aspects in the Middleware

In the specification / design domain, for each application several aspects such as functionality, timing, behavioural pattern (coordination), and the distribution, are defined. In the implementation domain, the functionality aspect is mapped into application components that are distributed (according to some initial distribution pattern) over available nodes making up the distributed system. The other aspects are transformed into policy descriptors that are downloaded into the middleware where various aspect managers interpret them at run time.

Aspects can be divided into application-specific and system-wide. Application-specific aspects are those concerned with a single application (combination of functionality, timing, etc.). In contrast, system-wide aspects effect the execution of all applications currently active in the middleware. According to this classification, the middleware itself is divided into two layers. In the highest layer, application-specific aspect managers are responsible for interpreting and enforcing an application's policies. In the lowest layer, global aspect managers deal with global policy descriptors and enforce system-wide policies.

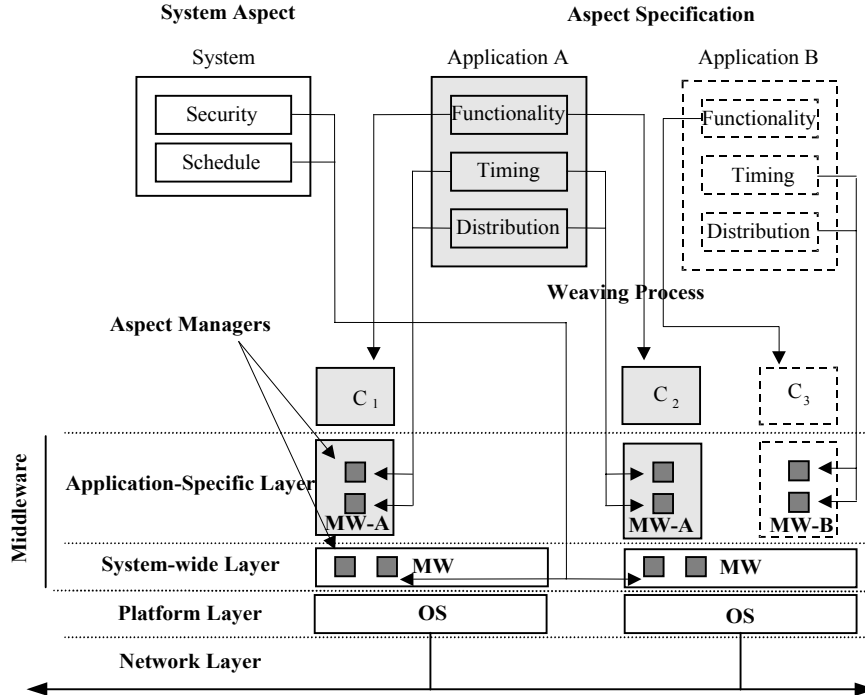


Figure 2: An Architectural View of the Implementation Domain

The separation into two layers has the important advantage that each application can define its own policies without interfering with policies of other applications. At the same time, it is still possible to define global policies to apply to the overall system.

B. Implementation Architecture

Figure 2 presents the case in which two applications, A and B, are defined and deployed in our system. We use light grey blocks for application A and dashed line blocks for application B. For each application, the set of aspects is defined. The functionality aspect is mapped into components C₁ and C₂ (deployed on different physical nodes) for application A and component C₃ for application B. The other aspects for the two applications are downloaded into the middleware in the application-specific layer, in the form of policy descriptor. In this way, each application can define its own policies regarding several aspects independent from other applications. Also, application-independent aspects are defined and then downloaded as policy descriptors in the middleware at the system-wide layer.

A prototype implementation of this architecture is done based on the JavaSpace shared data space system. On top of this prototype middleware, a case-study about a Traffic Management Pont is analysed by

examining the application of several distribution policies for different timing and bandwidth constraints. Details of the prototype implementation and the result of this analysis are presented in [7]; the results suggest that the idea of separation of concerns helps adjusting the best distribution policy to specific application settings.

I. CONCLUSION AND FUTURE DIRECTIONS

In this paper, we presented the main directions taken in the SACC project in order to realize separation of concerns in the development of distributed real-time systems. These directions include the development of a suite of specification languages for functionality (GAMMA-based) and various non-functional aspects such as coordination, timing, distribution etc, and an aspect-oriented distributed middleware to support separation of concerns in the implementation domain.

The main challenges in the theoretical part of the project, apart from developing a sound aspect-oriented theory, include mechanizing the proof system for the developed theory, and providing a refinement / synthesis method to extract concrete system representations from general requirements. In the practical part, developing an adaptive strategy for

finding and maintaining an optimal distribution policy is on the list of future work.

The activities in the theoretical track of the project so far were more focused on timing aspects whilst in the implementation track the focus was more on distribution policies. We are aiming at bridging this gap by both providing a formal semantics of distribution and implementation support for timing constraints. Currently, the translation of GAMMA functionality into implementation components (JavaSpace programs) is done manually; providing an automatic translation from specifications to programs (for functionality as well as other aspects) is another important issue in the future work. Studying an industrial case in our framework is another planned activity to provide a proof of concept for our approach.

REFERENCES

- [1] Jean-Pierre Banatre and Daniel Le Metayer, Programming by Multiset Transformation, *Communications of the ACM (CACM)*, 36(1):98--111, January 1993.
- [2] Chris L. Hankin, Daniel Le Metayer and David Sands, A Calculus of Gamma Programs, *Proceedings of the Fifth International Workshop on Languages and Compilers for Parallel Machines*, New Haven, Connecticut, Lecture Notes in Computer Science, vol. 757, pp. 342--355, Springer-Verlag, Berlin, 1993.
- [3] Michel R.V. Chaudron, *Separating Computation and Coordination in the Design of Parallel and Distributed Programs*, PhD thesis, Department of Computer Science, Rijksuniversiteit Leiden, Leiden, The Netherlands, 1998.
- [4] Lynne Blair, Gordon Blair and Anders Andersen, *Separating Functional Behavior and Performance Constraints: Aspect-Oriented Specification*, Technical Report MPG-98-07, Computing Department, Lancaster University, May 1998.
- [5] MohammadReza Mousavi, Giovanni Russello, Michel Chaudron, Michel Reniers, Twan Basten, Angelo Corsaro, Sandeep Shukla, Rajesh Gupta, Douglas C. Schmidt, Using Aspect-GAMMA in the Design of Embedded Systems, to appear in *the Proceedings of 7th IEEE International Workshop on High Level Design and Validation*, Cannes, France, IEEE Computer Society Press, 2002.
- [6] MohammadReza Mousavi, Twan Basten, Michel Reniers, Michel Chaudron, Giovanni Russello, *Separating Functionality, Behavior, and Timing in the Design of Reactive Systems: (GAMMA + Coordination) + Time*, Technical Report, Department of Computer Science, Eindhoven University of Technology, 2002.
- [7] Giovanni Russello, Michel Chaudron, Maarten van Steen, *Separating Distribution Policies in a Shared Data Space System*, Internal Report IR-497, Department of Mathematics and Computer Science, Vrije Universiteit, June 2002.