Using Aspect-GAMMA in the Design of Embedded Systems

(Extended Abstract)

MohammadReza Mousavi, Giovanni Russello,
Michel Chaudron, Michel Reniers, Twan Basten
Technische Universiteit Eindhoven (TU/e)
P.O. Box 513, 5600 MB
Eindhoven, The Netherlands

Angelo Corsaro, Sandeep Shukla,
Rajesh Gupta, Douglas C. Schmidt

University of California at Irvine
Irvine, CA 9269, USA

Abstract

This paper proposes a design framework that takes advantage of the aspect-orientation paradigm. The proposed framework is based on the multi-set transformation language called GAMMA, used for the functional aspect, together with a set of modelling notations for other aspects of system design, namely coordination, timing, and distribution.

1 Introduction

Separation of concerns is one of the concepts at the core of modern system design and evolution. It has been advocated as a key principle for reducing the complexity of developing large-scale systems [10, 7]. In particular, in [7], *orthogonalization of concerns* has been illustrated in the context of embedded system design. It advocates capturing the design intent at the highest possible level of abstraction, and separating the timing, concurrency, and communication concerns. However, since the methodology is couched in the platform-based design methodology, the highest level of abstraction in most cases is some form of finite state machines. From our perspective, finite state machines already mix control and data-flow concerns in the system. Therefore, we propose raising the level of abstraction, and separating concerns from the very inception of the ideas of the system under design.

Separation of concerns (especially the cross-cutting ones such as timing) brings about the following benefits in the design, verification and implementation phases:

1. *More focused design and lighter verification.* One of the main difficulties in the design of some concerns is that they usually *crosscut* the responsibility of several encapsulation units (components, modules, or objects). Thus a design time separation of concerns allows for a more focused design method that gathers cross-cutting concerns in one dedicated place. Furthermore, properties related to a concern (set of concerns) can be verified by looking at the particular concern (combination of concerns) instead of the entire design.

2. *Localized change.* In this paradigm making and tracking changes is localized to the involved aspects. Subsequently, a (possibly automated) composition process (also referred to as *weaving* process) spreads the consequences of change to the composed behavior of a system.

3. *Abstractness from other modeling methods.* By using separate modeling techniques for different aspects, one may decide to change the modeling formalism of one aspect and still be able to use the other aspect models.

This paper combines a formal design framework, which is based on a multi-set transformation language called GAMMA [1, 2], with aspect-oriented development concepts from software engineering [4, 8]. We refer to design concerns as *aspects* following the software terminology. We illustrate how having a tailor-made formalism for each aspect, that is abstracted from other aspects, is a key benefit of such a design framework. To clarify our discussions, we sketch an architecture specification and design method for distributed real-time embedded systems. We propose separating the concerns of computation, coordination, timing, and distribution, through different simple and abstract notations for these aspects. We also propose a weaving process that maps combinations of these different aspects to a formal semantics domain.

The remainder of this paper is organized as follows: Section 2 discusses computation, coordination, timing, and distribution as different aspects of a design and suggests languages/notations to specify them. Section 3 proposes a simple model of weaving the functional and non-functional aspects in a single semantic model. Section 4 presents a small

$$
\begin{array}{lll}
\mathit{Program} & ::= & \mathit{ProgramName} = \{\mathit{Rules}\} \\
\mathit{Rules} & ::= & \mathit{Rule} \mid \mathit{Rule}, \mathit{Rules} \\
\mathit{Rule} & ::= & \mathit{RuleName} = \mathit{MultisetExp} \mapsto \\
 & & \qquad \mathit{MultisetExp} \Leftarrow \mathit{Condition} \\
\mathit{MultisetExp} & ::= & \epsilon \mid \mathit{BasicExp} \mid \mathit{BasicExp}, \mathit{MultisetExp} \\
\mathit{BasicExp} & ::= & \mathit{Term} \mid (\mathit{Term}, \mathit{Term})
\end{array}
$$

Figure 1. Basic GAMMA Syntax

case study that we work on using our design method. Section 5 provides concluding remarks and research directions. For sake of brevity, we do not present detailed formalities of aspect-modeling languages in this abstract and sketch the overall design criteria.

2 Exploring Aspects

This section focuses on the specification of computation (the functional aspect) and the three non-functional aspects coordination, timing, and distribution. Computation is somehow in the center of our design method in that other aspects build their semantic model on top of functionality. We use a subset of GAMMA for specifying basic component functionalities (computations) and present its distinguishing features. We then present some ideas about specifying other aspects.

2.1 Modeling Computation with GAMMA

GAMMA is an abstract language, based on chaotic rewriting of a multi-set, designed to support scalable parallel execution of a program on parallel and/or distributed architectures [2, 1]. The basic and atomic piece of functionality in GAMMA is the *rule*.

In this paper, we focus on a subset of GAMMA involving the specification of basic rules. We thus factor out structuring decisions (present in the calculus of GAMMA [5]) and make them a separate aspect model, namely the coordination model.

The syntax of a simple GAMMA program is given in Figure 1. A GAMMA program consists of a non-empty set of rules, each rewriting the content of the shared multi-set of data items. Execution of a program consists of applying rules to the multi-set in arbitrary orders (sequential or parallel). Each rule consists of a set of terms valuated by multi-set content values (this replacement is not necessarily unique for a specific rule and multi-set). If a certain valuation of variables satisfies the condition in a rule, applying the rule results in removing the left-hand side valuations from the multi-set and replacing them by the valuation of the right-hand side expression. We use predicate logic formulas for the condition part throughout this paper. In [9], a formal operational semantics for GAMMA is given in the style of Plotkin [11].

Henceforth, the GAMMA model is only concerned with basic functionalities in the form of a simple input-computation-output pattern that abstracts from the following details:

1. *Relative ordering of actions (coordination).* Basic functionalities (rules) are specified independently of each other. Hence, no special ordering of actions (control structure) is imposed on this particular specification.

2. *Timing.* The basic GAMMA model does not include any information about timing. Since it abstracts from ordering of actions, even a qualitative (causal) notion of time is not present in the GAMMA model.

3. *Distribution.* For any distributed system, the shared data-space is an abstraction that eases the programming, yet must be distributed in the implementation.

4. *Hardware resources.* Chaotic behavior of GAMMA programs includes all possible levels of true concurrency. This means that the semantic model of a GAMMA program is general enough to be adopted to any particular hardware architecture that allows a certain level of true concurrency. In other words, adding information about hardware resources as an aspect will refine the GAMMA model to a particular platform-dependent program.

5. *Fault tolerance.* The GAMMA execution model requires programs to be designed in such a way that duplicated execution of atomic actions of a program cannot affect the functionality. Hence, replication of actions can be added transparently to the functional model.

This abstraction is the key issue in our approach since it allows several ways of restriction (refinement) of the basic functionality model by adding different (combinations of) aspects.

2.2 Coordination

The goal of our coordination language is to restrict behavior of GAMMA programs to certain execution orders. Hence, the language should provide composition operators to structure execution and restrict behavior of GAMMA rules.

The syntax of our coordination language is given in Figure 2. A coordination expression is basically composed of GAMMA rules. Also, simpler schedules can be composed using rule-conditional ($r \curvearrowright s$ meaning that if r can be

$$
\begin{array}{rcll}
Schedule & ::= & RuleName & | \\
 & & RuleName \curvearrowright Schedule & | \\
 & & Schedule \; ; \; Schedule & | \\
 & & Schedule \; || \; Schedule & | \\
 & & \mu RecursionVar.\; Schedule & | \\
 & & RecursionVar &
\end{array}
$$

Figure 2. Coordination Language Syntax

scheduled then s is selected for execution, otherwise the schedule terminates), sequential composition (;), parallel composition ($||$), or recursion ($\mu x.s$). The recursion operator $\mu X.$ bounds the recursion variable x in the expression s.

Our coordination language is a process-algebraic formalism that provides a formal framework for different composition models on top of a functionality specification. In [3], the relationship between the composition models and computational complexity is studied in detail.

2.3 Timing

Timing constraints can be associated with the functionality specification to provide assertions regarding the execution time of GAMMA rules. This time is relative to the point from which the rule is selected for execution. Hence, there is an inter-dependency between overall end-to-end timing behavior of the design and both timing and coordination aspects. We return to these inter-dependencies in the weaving section.

We propose to add the timing aspect to a GAMMA specification by associating an interval to each rule name (denoted by $T_{RuleName} = Interval$). This timing representation keeps the syntactic specification of timing separate from rule definitions, and hence allows independent change of both aspects. This method also allows a rule to have no timing assertion, which will be replaced by a default interval ($[0, \infty]$) in the weaving process.

Since GAMMA rules assume a shared access to data, the timing aspect does not specify any assumptions about the cost of accessing the data items in a distributed setting. The above estimation is therefore only related to the computation time for each computation.

2.4 Distribution

Distribution as a separate concern contains information about mapping sorts of data objects and rules to different locations. Adding the distribution aspect introduces interconnections with the timing aspect in that a cost of access is associated for remote data access. Thus, timing of individual component execution is changed when taking the distribution aspect into account.

The distribution aspect is modelled as a mapping between multi-set terms (present in the GAMMA rules) to the physical locations. To specify distribution, we assume a set R containing rule names and a set T containing data types. Data types are used to categorize data items used/produced by different rules. We do not specify how to assign this typing to terms but assume that there is a function from the sets of terms (structures over variables and constants) to types. The set of locations is denoted by P. Static distribution is defined as a function $StaticDist : (R \cup T) \rightarrow \mathcal{P}(P)$, representing the locations of the data objects and rules of each type. Note that we do not restrict locations to contain both data and processing (rules) and hence, a location may represent a storage node or a processing unit, or both.

This general specification of distribution can be used to model more specific distribution policies, such as push and pull models. For example in a push model, the distribution mapping should map any data type to its consumer side. In a pull model, however, the data type remains on the producer side and should be accessed (fetched) from the producer by the consumer.

3 Weaving Aspects

Weaving refers to the merging of different aspects of design into a complete system model. In our case, we want to relate functionality, coordination, timing, and distribution, and present them in one semantic model. The orthogonality of non-functional aspects allows the designer of each aspect to abstract from the others. Consequently, the weaving process should allow change or even absence of one aspect in the whole semantics. Hence, an ideal weaving process provides formal semantics for any (meaningful) combination of aspects.

Our proposal for a formal semantics of weaving consists of a timed transition system [6] with transitions of a GAMMA program and timing consisting of computation time plus communication time.

We denote the computation time of a rule r by $comp(r)$. This function induces a *by-name* weaving method to relate GAMMA rules and their respective timing estimations. In this paper, we assume that $comp(r)$ works as a function returning the execution time estimation of a rule, if available ($T_{RuleName}$), or otherwise $[0, \infty]$. Nevertheless, this assumption could be relaxed by allowing several intervals associated to a rule, and hence letting $comp(r)$ return one of the intervals non-deterministically (or a set of intervals). This could be used to model the situation where a rule has multiple possible execution times, depending e.g. on varying implementation environments.

To represent communication costs resulting from the distribution policy, we use the function $comm(r)$, which returns the time cost for making local copies of the data items needed for the execution of rule r. For a rule r, $comm(r)$ is computed by taking the maximum of communication costs for all variables (of data items) v present in rule r, that reside in a different location than the location of r. If all the data needed for the execution of a rule is available at the location of the rule itself, we assume the communication cost to be 0.

The simple time weaving function presented here can be extended by adding estimations for failed attempts to execute a rule, or by defining the timing estimation as a function of multi-set size or contents. In GAMMA, rule implementations, computation time and failure time may depend on the time for searching the multi-set to find the appropriate valuation. These two extensions thus add to the practical value of the proposed method. Such extensions can illustrate the profit of the separation of concerns in the method outlined in this paper.

4 Case Study: Designing the Control of an Elevator System

We design the control software of an elevator system using our method and show how design concerns and correctness criteria can be localized using this method.

The elevator system consists of an elevator moving up and down between floors (numbered from 0 to $MaxFloor$) of a building to service requests. On each floor there is a push button to announce a request for an elevator when turned on. When an elevator arrives on a floor, the request flag is turned off automatically. The same setting works for the push buttons inside the elevator, which indicate the requested stops for passengers inside.

To model this system we propose a multi-set containing events requesting an elevator stop represented by $((inStop, i),\ status)$ and $((extStop, i), status)$ that show the status of the request button for the i'th floor, inside and outside the elevator, respectively. The tuple (cf, i), shows where the elevator currently resides. The GAMMA program presented in Figure 3, represents the functionality aspect of the elevator system.

The initial multi-set for this system is defined as:

$$\begin{aligned} State = [&((inStop, 0), off), \ldots, ((inStop, MaxFloor), off),\\ &((extStop, 0), off), \ldots, ((extStop, MaxFloor), off),\\ &(cf, 0)] \end{aligned}$$

which shows that the elevator is at the ground floor initially and that there are no requests for the elevator. Note that the proposed functionality model does not impose any control strategy. In other words, the execution of the GAMMA model allows for any possible execution of rules (including, for example, going up and down between floors without servicing the requests).

The coordination aspect defines how rules are composed to define different control strategies. A simple strategy is going up and down if there exists any request, and servicing them in order.

$$\begin{aligned} ElevatorSchedule =& (\mu X.inRequest \mid\mid X) \mid\mid\\ & (\mu X.extRequest \mid\mid X) \mid\mid\\ & (\mu X.ServiceUp\ ; ServiceDown; X)\\ ServiceUp =\ & \mu X.((load || unload)\ ;\\ & (moveUp \curvearrowright (moveUp\ ;\ X)))\\ ServiceDown =& \mu X.((load || unload)\ ;\\ & (moveDown \curvearrowright (moveDown\ ;\ X))) \end{aligned}$$

The informal description of the above coordination strategy is as follows. Events of pushing the request button cannot be controlled by the system and hence they happen in an arbitrary rate in parallel with the rest of the system. The elevator goes sequentially up and down by performing the $ServiceUp$ and $ServiceDown$ schedules. $ServiceUp$ (respectively, $ServiceDown$) includes servicing the possible requests at the current floor and trying to go up (or down) if there is any request at/for the higher (lower) floors. If the attempt to go up (down) fails, since there is no pending request there, then the direction changes and the elevator tries to go down (up).

The timing aspect of the case study consists of performance measures related to each basic operation in the elevator system (consisting of going up and down, and servicing requests). Suppose that the following timing information is given about the elevator system:

- Pushing an internal or external button does not take time at all:
  $T_{inRequest} = T_{extRequest} = [0, 0]$.
- Going up and down between floors takes $StepTime$ for each floor:
  $T_{moveUp} = T_{moveDown} = [StepTime, StepTime]$.
- The elevator will be loaded/unloaded within $MinService$ and $MaxService$ amount of time, depending on the number of people and goods waiting for it:
  $T_{load} = T_{unload} = [MinService, MaxService]$.

The timing information allows us to verify the timeliness of a functional specification, possibly for a given coordination, assuming the aspects are appropriately weaved together.

Finally, we consider the distribution aspect. Suppose that sensors for request buttons on each floor are connected to the elevator via a field-bus network. In this case, accessing the physically distributed locations (from the elevator) will take some time. To specify this model of distribution, we assume a location for the elevator and its internal buttons

$$
\begin{aligned}
\mathit{ElevatorSystem} = \{ \quad & \mathit{inRequest} = ((\mathit{inStop}, i), \mathit{off}) \mapsto ((\mathit{inStop}, i), \mathit{on}), \\
& \mathit{extRequest} = ((\mathit{extStop}, i), \mathit{off}) \mapsto ((\mathit{extStop}, i), \mathit{on}), \\
& \mathit{moveUp} = (\mathit{cf}, i) \mapsto (\mathit{cf}, i+1) \Leftarrow \\
& \quad i < \mathit{MaxFloor} \wedge \exists j; i < j \wedge (((\mathit{extStop}, j), \mathit{on}) \in \mathit{State} \vee ((\mathit{inStop}, j), \mathit{on}) \in \mathit{State}), \\
& \mathit{moveDown} = (\mathit{cf}, i) \mapsto (\mathit{cf}, i-1) \Leftarrow \\
& \quad i > 0 \wedge \exists j; j < i \wedge (((\mathit{extStop}, j), \mathit{on}) \in \mathit{State} \vee ((\mathit{inStop}, j), \mathit{on}) \in \mathit{State}), \\
& \mathit{load} = ((\mathit{extStop}, i), \mathit{on}) \mapsto ((\mathit{extStop}, i), \mathit{off}) \Leftarrow (\mathit{cf}, i) \in \mathit{State}, \\
& \mathit{unload} = ((\mathit{inStop}, i), \mathit{on}) \mapsto ((\mathit{inStop}, i), \mathit{off}) \Leftarrow (\mathit{cf}, i) \in \mathit{State}\}
\end{aligned}
$$

Figure 3. GAMMA Program for the Elevator System

and a location for each external button. The distribution function for the elevator system then looks like the following:

$$
\begin{aligned}
&\mathit{StaticDist}(\mathit{type}((\mathit{extStop}, i), \mathit{status})) = \{\mathit{floor}_i\} \\
&\mathit{StaticDist}(\mathit{type}((\mathit{inStop}, i), \mathit{status})) = \{\mathit{elevator}\} \\
&\mathit{StaticDist}(\mathit{type}(\mathit{cf}, i)) = \{\mathit{elevator}\} \\
&\mathit{StaticDist}(\mathit{extRequest}) = \{\mathit{floor}_i \mid 0 \le i \le \mathit{MaxFloor}\}
\end{aligned}
$$

and for each rule $rule$ other than $\mathit{extRequest}$:

$$\mathit{StaticDist}(\mathit{rule}) = \{\mathit{elevator}\}$$

This distribution policy defines where the GAMMA rules $\mathit{moveDown}$ and moveUp must look for remote copies of external request values from distributed locations.

From a verification point of view, some basic un-timed properties of design can be verified using the semantics of GAMMA and coordination. For example, it can be shown that the elevator eventually honors all requests, using only the functionality and the coordination aspect model. For more complicated timed properties however, the reasoning requires that the weaved model contains timing and distribution.

In Figure 4, a fragment of the timed transition system is given that results from weaving the computation, coordination, timing and distribution of this system as described above. The transitions are labelled by the name of the rule(s) that are executed, the timing estimation of the execution, and the communication cost. For simplicity, only the relevant elements of the multi-set contents are represented in this figure. It is assumed that the time cost for communicating data from one node to another is CT. In this transition system, communication time is only associated with rules concerning moving up and down since they are the only rules that have to fetch their required data from other distributed nodes (e.g. request button status at other floors) over the field-bus network.

The example given in this section illustrates how a distributed real-time embedded application can be developed systematically. The proposed method is amenable to formal verification techniques which increase the robustness of the design. When the design is complete and verified, the system engineer can proceed with an implementation.

5 Conclusion and Future Directions

In this paper, we presented the main ideas behind our method for separation of concerns in the design of distributed real-time embedded systems. The proposed method consists of separating the aspects of functionality, coordination, timing and distribution in the design phase, and providing a weaving mechanism to provide a formal semantics for composed aspects. This weaving method enables us to have localized reasoning about properties of aspect models and their inter-relationships.

The main challenges in our future research are the following:

- Extension of the method sketched in this paper to other aspects such as hardware resources, power-awareness, fault-tolerance, persistence, etc.
- Developing/studying logics for expressing properties of the aspect models and the weaving of those.
- Performing case studies to validate the method.
- Developing automated design methods and tools that support the aspect weaving process, the reasoning in the aspect models, and the refinement from aspect specifications towards system implementation.

References

[1] J.-P. Banâtre, P. Fradet, and D. Le Métayer. Gamma and the chemical reaction model: Fifteen years after. In C. S. Calude, G. Paun, G. Rozenberg, and A. Salomaa, editors, *Multiset Processing: Mathematical, Computer Science, and Molecular Computing Points of View*, volume 2235 of *Lecture Notes in Computer Science*, pages 17–44. Springer-Verlag, Berlin, 2001.

[2] J.-P. Banâtre and D. Le Métayer. Programming by multiset transformation. *Communications of the ACM (CACM)*, 36(1):98–111, Jan. 1993.

[3] M. R. V. Chaudron. Separation of Correctness and Complexity in Algorithm Design. Technical Report 94-36, Leiden, The Netherlands, 1994.

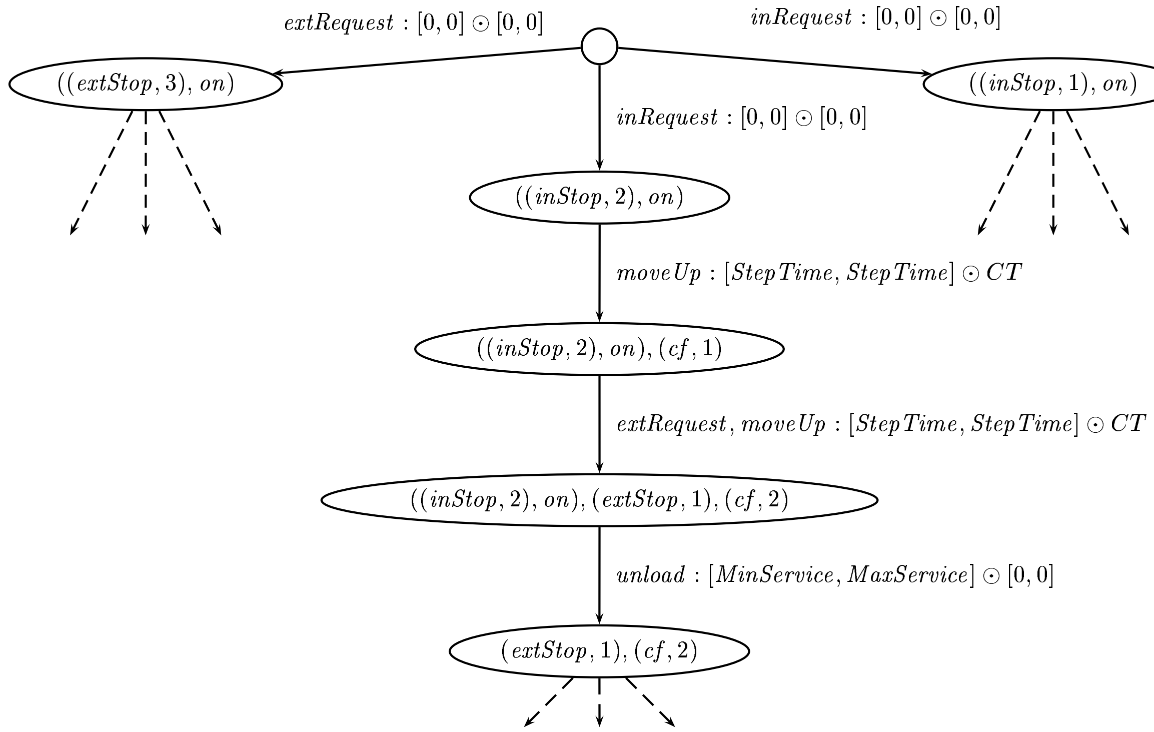

Figure 4. Fragment of the Timed Transition System after Weaving.

[4] T. Elrad, R. E. Filman, and A. Bader, editors. Special issue on aspect oriented programming. *Communications of the ACM (CACM)*, 44(10). ACM Press, 2001.

[5] C. L. Hankin, D. Le Métayer, and D. Sands. A calculus of Gamma programs. In U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua, editors, *Proceedings of the Fifth International Workshop on Languages and Compilers for Parallel Machines*, volume 757 of *Lecture Notes in Computer Science*, pages 342–355. Springer-Verlag, Berlin, 1993.

[6] T. A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In J. W. de Bakker, K. Huizing, W. P. de Roever, and G. Rozenberg, editors, *Proceedings REX Workshop on Real-Time: Theory in Practice,* Mook, The Netherlands, June 1991, volume 600 of *Lecture Notes in Computer Science*, pages 226–251,.Springer-Verlag, Berlin, 1992.

[7] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1523–1543, December 2000.

[8] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Aksit and S. Matsoka, editors, *Proceedings of the 11th European Conference on Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242, Berlin, June 1997. Springer-Verlag.

[9] M. Mousavi, T. Basten, M. Reniers, M. Chaudron, and G. Russello. Separating functionality, behavior and time in the design of reactive systems: (GAMMA + coordination) + time. Technical Report CSR 02-09, Department Computer Science, Eindhoven University of Technology, 2002.

[10] P. Tarr, H. Ossher, W. Harrison and S.M. Sutton. N Degrees of Separation: Multi-Dimensional Separation of Concerns. In *Proceedings of the International Conference on Software Engineering (ICSE99), Los Angeles, CA*, pages 107–199. ACM Press, May 1999.

[11] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, Sept. 1981.