On Methodologies for Constructing Correct Event-Based Applications *

Pascal Fenkam, Mehdi Jazayeri, Gerald Reif
Technical University of Vienna, Distributed Systems Group
A-1040 Vienna, Argentinierstrasse 8/184-1
{p.fenkam, m.jazayeri, g.reif}@infosys.tuwien.ac.at

Abstract

Various application domains exist where the advantages of the event-based paradigm make it a key technology. In general, this architectural style allows better control of the structural and behavioral complexity of applications: components can be developed independently and loosely integrated. The computational behavior of this paradigm, however, remains poorly understood. This position paper argues on the necessity of a new methodology for constructing event-based applications as well as a new logic that clarifies the computational behavior of such applications. For this, the paper presents some factors that make the event-based style so troublesome and discusses the (non-) adequacy of existing formal techniques for the construction of event-based applications.

1 Introduction

The different facets of the event-based (EB) architectural style (also called *publish/subscribe* or *implicit invocation*) are currently widely investigated. In this paradigm, some components (called *subscribers*) specify their interest in some kind of data (called *event notification* or simply *event*) that other components publish (or announce). The publishing component is called the *publisher*. The specification stating what kind of events a subscriber is interested in is called a *subscription*. The communication between the publishers and the subscribers is done through the *event-based infrastructure*. Events are sent to this infrastructure which then matches them against existing subscriptions and invokes interested subscribers. The dispatching of events to subscribers is completely transparent to the publishers, leading to loose coupling of components. This makes the event-based paradigm a key technology for many application domains.

*This work is supported by the Austrian Research Foundation (FWF) project RAY (Number P16970-No4).

Despite the wide acceptance of the EB concept, the development of applications based on it remains an ad hoc and informal process. Consequently, as EB systems proliferate, including in safety-critical domains, it remains difficult [20, 21] to reason about the reliability of such systems. This is especially unfortunate in the case of the EB paradigm because the loose component coupling it supports naturally leads to a component based approach to the construction of distributed applications and therefore a consequent expectation of increased reliability.

The aim of this paper is to shed some light on the strange behaviors of the EB paradigm. We present three behavioral aspects of this style that we have distilled and that make us strongly believe that at least a new logic is needed that captures the quintessential properties of this style. First, we show that the traditional meaning of the sequential composition of programs does not hold anymore. This is related to the second issue, where we claim that the associativity of the sequential composition is not present in the context of EB applications. The third strange behavior is called the "assertion fleeing effect": the result of a subscriber is always postponed to the "end" of the post-conditions of programs. Finally, we discuss the applicability of existing formal techniques to the construction of EB applications. We claim that although a new formal specification language may not be required, a logic must be developed that clarifies the behavior of the EB architectural style and renders it unambiguously understood.

The remainder of the paper is organized as follows. We condition methodologies for the development of correct EB applications with a set of requirements that we present in the next section. Section 3 summarizes existing approaches for constructing EB applications and discusses their shortcomings. In section 4, we briefly present an abstract semantics of the concept of EB system. This abstract semantics is the basis for the elicitation of event-based systems's behaviors in Section 5. We pursue the discussion of the need of a new methodology for constructing EB applications in Section 6. Section 7 concludes the paper.

2 Requirements for a Methodology

We present some requirements for a methodology for developing correct EB applications. For the elicitation of these requirements, we consider three application areas: component based software engineering (CBSE), mobile computing, and coordination languages. Our requirements are clearly not exhaustive as the EB paradigm is applied in many other domains.

2.1 Generic Requirements

Before going to the specific requirements of the above areas, there are generic requirements that a software development methodology must strive for, namely the scalability to large systems and the ability to be used to consistently produce maintainable and high-quality software at low cost and with a small turnaround time [25]. Four issues are identified in this requirement: scalability, quality, cost and schedule, and maintenance issues. Since the degree to which a methodology supports these issues is in general hard to measure, it may be argued that any development method supports them more or less. The goal is therefore, to find methods that better respond to these requirements than those currently available.

2.2 CBSE related Requirements

CBSE is an emerging methodology that promises to adequately address many of the above issues. And, indeed, many researchers are working on making it a reality.

One of the issues that need to be solved is the composition and integration of components; the EB paradigm is currently intensively deployed for this purpose. Any software development methodology that supports the development of EB applications must, therefore, also explicitly support CBSE. That is, such an approach must be compositional, both at the abstract level (conceptual level) as well as at the implementation level. At the abstract level, it must be possible to compose component specifications as well as proofs about properties of these specifications. In particular, the methodology must allow building models in a clearly defined and intuitive way such that one can understand the whole by understanding the parts and recombining them in predictable ways [13].

2.3 Mobility related Requirements

Mobile computing is a paradigm in which users are able to communicate and use their applications while they are moving [9]. The difficulty in such computational models is that one can not rely on the permanent availability of the network. The resulting connectivity intermittence has led to the argument that traditional client/server middleware are not suitable for mobile computing [3, 9, 10, 23]. The EB paradigm is viewed as a valuable replacement to client/server; it does not require subscribers to be present at the time of announcement. This asynchrony in the communication is mapped one-to-one to mobile computing. A subscriber represents a mobile device that comes to and leaves the network at will.

From this short analysis, we derive that an approach for the construction of EB applications must support components that leave and come intermittently. In addition, to support easy maintenance of systems, or to be applicable to pervasive computing, such an approach must be able to foresee the integration of new components in a running system. In the EB terminology, this implies that the reasoning should not be based on a pre-defined static set of subscribers.

2.4 Coordination related Requirements

Coordination includes the specification, analysis, and control of the cooperation between components of a system. A significant amount of work is currently undergoing on this topic (see [4, 5, 29]). Many of the proposed solutions are based on the EB architectural style. The requirement that are placed upon methodologies for constructing EB applications by coordination languages are in essence similar to those of component based engineering and mobile computing. In particular, it is argued that compositionality, evolvability, and easy handling of volatile business requirements must be supported [4].

Due to the similarities in the goals of the coordination paradigm and the many uses of the EB architectural style, it is important to notice that neither subsumes the other; the EB paradigm is used for other purposes than coordination while there are coordination techniques that are not based on the EB technique (e.g. [5, 7]).

3 Related Work

Despite the wide acceptance of the EB paradigm, not much work has been presented on building correct applications using it.

3.1 Formalizing Architectural Styles

Several researchers have attempted to provide formal techniques that can support the treatment of EB applications. Although the ultimate goal of such works is reliability, the developed approaches have not been successful. Examples of such approaches are that of Garlan and Notkin [22], as well as that of Abowd, Allen, and Garlan [1, 2] who propose frameworks for formalizing architectural styles in general and implicit invocation in particular. These

approaches primarily focused on taxonomic issues, and do not provide an explicit computational model that permits compositional reasoning about the behavior of EB applications [12, 11]. This claim is justified by observing that the semantics of a method (also called operation or function) is not defined. It is, therefore, not possible to reason about the behavior of these methods (at least not without extending the framework). Moreover, it is not said how a method must be specified. For instance, a key question in formally specifying EB applications is how does a designer specify that a method m announces an event e whenever the state satisfies the condition Q? The next issue in such approaches is that there is no indication on whether the specifications are realizable or not. Given such a specification, what is the next step in building an application? How does a designer show that a given application satisfies such a specification?

3.2 Verification of EB Applications

Dingel et al.[12] propose an approach for verifying the correctness of EB applications. Since this is an a-posteriori approach, the components of the completed program are verified in isolation and then put together where general properties are attempted to be proved. The scalability to large applications can not be achieved, since erroneous design decisions taken in early steps are propagated until the system is implemented and attempted to be proven correct [26, 16]. Further, given that this approach assumes a static set of subscriptions the evolvability and the maintainability of systems are therefore hard to support.

3.3 Model checking EB Applications

Model checking EB applications is interesting in that a significant part of the process is carried out automatically. In [8, 20, 21], attempts to apply model checking to the verification of EB applications are discussed. The authors provide generic frameworks to be reused by modelers in the process of defining the abstract structure related to their systems. Indeed, the authors succeeded in factoring the work such that, for instance, the event delivery policy is now a pluggable element with various packaged policies (prepared by the authors) that can be used off-the-shelf.

In general, in addition to being an a-posteriori approach, model-checking EB applications suffers from the fact that there is no known adequate way for specifying the announcement of events in presence of interference. Ideally, it should be possible to specify components, verify their properties, implement them, and integrate them in a predictable way.

The discussion in this paper results from our early attempts to construct a methodology for the stepwise development of event-based applications [16, 15, 17, 14].

4 Abstract Semantics of the EB Paradigm

To make clear the differences in the behavior of the EB paradigm, we need to clarify the programming language and the semantics that we assume. We will attempt as far as possible to remain informal in this paper. Let us assume the following while-parallel language that also allows announcement of events.

$$P ::= x := e \mid P_1; P_2 \mid \textbf{if } b \textbf{ then } P_1 \textbf{ else } P_2 \textbf{ fi} \mid \textbf{skip} \\ \mid \{P_1 \| P_2\} \mid \textbf{while } b \textbf{ do } P \textbf{ od} \mid \textbf{announce(e)}$$

The semantics of the constructs in this language seem to have the traditional meaning. Indeed, the semantics of the announcement construct is simple: the set of subscribers to an event are executed in parallel with the remainder of the announcing program. That is, if we assume that there is no event announcement in the program z_1 and that the program z_e is subscribed to the event e, then $z_1; \mathbf{announce}(e); z_2$ is a program that behaves as z_1 and follows as $\{z_e \| z_2\}$ if z_1 terminates. In this semantics, we abstract from the EB infrastructure and simply model its behavior: the invocation of subscribers. For simplicity, we have removed the await construct that we propose [16, 15, 17] for synchronization and mutual exclusion in EB systems. The await construct gives more power to the language and allows it to simulate more types of EB systems. The aspects related to this construct will not be investigated in this paper. Further, we have ignored passing the event to the subscribers; this is discussed in [14].

5 Challenges of the EB Paradigm

We take a close look at the above semantics and present some aspects that make reasoning about EB applications difficult.

5.1 The meaning of sequential composition

In traditional systems (e.g. Hoare Logic [30], VDM [27]), a program $z_1; z_2$ is a program that behaves as z_1 and follows as z_2 if z_1 terminates. If the result of z_1 (respectively z_2) is captured by means of its post-condition E_1 (respectively E_2), then the result of $z_1; z_2$ is E_2 if the pre- condition of z_2 follows from E_1. This is expressed in Hoare's style as:

$$\frac{\{E_0\}\ z_1\ \{E_1\} \quad \{E_1\}\ z_2\ \{E_2\}}{\{E_0\}\ z_1; z_2\ \{E_2\}} \tag{1}$$

Let us now consider the case with an event-announcement. If the subscribers of e, say z_e, is such that $\{E_1\}\ z_e\ \{E_e\}$ holds, then the following rule follows from a naive double application of the above rule:

$$\frac{\begin{array}{c}\{E_0\}\ z_1\ \{E_1\}\\ \{E_1\}\ z_e\ \{E_e\}\\ \{E_e\}\ z_2\ \{E_2\}\end{array}}{\{E_0\}\ z_1; \mathbf{announce}(e); z_2\ \{E_2\}} \quad (2)$$

This rule is, however, unsound because the computational model of the EB paradigm requires to interpret the announcement of the event e as the parallel execution of z_e and z_2. If we assume that z_e and z_2 coexist, that is, there is no interference of one with the other, then the following rule must hold.

$$\frac{\begin{array}{c}\{E_0\}\ z_1\ \{E_1\}\\ \{E_1\}\ z_e\ \{E_e\}\\ \{E_1\}\ z_2\ \{E_2\}\end{array}}{\{E_0\}\ z_1; \mathbf{announce}(e); z_2\ \{E_2\ \wedge\ E_e\}} \quad (3)$$

Hence, sequential composition of programs when the EB paradigm is involved does not always have the traditional meaning. As a natural consequence, the iteration rule looses its traditional properties.

5.2 Non-associativity of sequential composition

A natural consequence of the above rules is that the associativity of the sequential composition of programs may no longer be assumed. Without event announcement, $z_1; \{z_2; z_3\}$ has the same results as $\{z_1; z_2\}; z_3$. This does not hold when z_2 announces an event. Let us replace z_2 with the announcement of the event e. The naive way of composing the three programs z_1, $\mathbf{announce}(e)$, and z_3 is to start by composing z_1 with $\mathbf{announce}(e)$ which results in the program $z_1; z_e$ whose post-condition is E_e (if the pre-condition of z_e follows from E_1). Next, we compose $z_1; \mathbf{announce}(e)$ with z_3 which results into a program satisfying E_2 if the pre-condition of z_3 follows from E_e.

This result is, however, unsound. The semantics of the announce construct requires that the program z_e be executed in parallel with the remainder of the announcing program, i.e. z_3. The sequential composition operator is, therefore, right associative in the presence of event announcement.

5.3 Fleeing Assertions

As shown in the previous section, in the presence of event announcement, sequentially composing a program z_1 with the program z_2 does not result in a program that behaves as z_1 and follows as z_2. In absence of interference, the result of the subscribers to e (an event announced by z_1) will always hold in the final state of the program. Let us explain this in the light of the following figures.

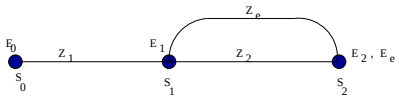

Figure 1. Behavior of $z_1; \mathbf{announce}(e); z_2$

Figure 1 captures the behaviors of $z_1; \mathbf{announce}(e); z_2$ as expressed by Rule 3. The first program z_1 is executed in a state satisfying E_0. Next an event announcement is executed in the state S_1 satisfying E_1, resulting in the parallel execution of z_e and z_2. The programs z_1 and z_2 both terminate in the state S_2 which satisfies E_2 and E_e.

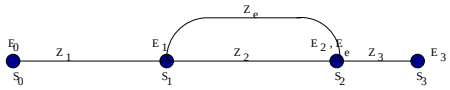

Figure 2. Naive Interpretation of $z_1; \mathbf{announce}(e); z_2; z_3$

We now perform a sequential composition of the above program with z_3. Figure 2 illustrates the case of a naive sequential composition: z_3 starts after z_2 and z_e are terminated. This is the wrong interpretation discussed in the previous section.

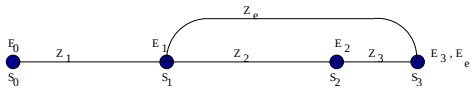

Figure 3. Fleeing Assertion

Figure 3 shows the sound behavior of $z_1; \mathbf{announce}(e); z_2; z_3$. The subscriber z_e is executed in parallel with $z_2; z_3$. The post-condition of z_2 which held in the state S_2 in Figure 1 now instead holds in the state S_3. In general, the result of a subscriber can not be " caught" by a sequentially composed program. We say that the post-condition of such a subscriber is a fleeing assertion; it keeps "fleeing" to the final state of any computation (provided there is no interference).

5.4 Similarity with the π-Calculus

The semantics that we have given to the announce construct in this paper (and also in [16, 15, 17, 14]) has some resemblance to asynchronous communication. In this communication mechanism, in fact, a process/agent sends a message to the other process and both continue their exe-

cution running in parallel with each other. The behavior advocated can, therefore, also be observed in such systems.

One has, however, to be careful when analyzing this behavior in the context of formal frameworks for asynchronous communication. Considering for instance the π-calculus [28], the agent $(u(a).Q_1 \| \overline{u}(b).Q_2).P$ is an agent such that $\overline{u}(b).Q_2$ sends the message b to $u(a).Q_1$ and evolves as Q_2 while the recipient evolves as $Q_1[b/a]$ and run in parallel with Q_2. And, once the agent $(u(a).Q_1 \| \overline{u}(b).Q_2)$ is terminated, the agent P can be executed. This is completely different from event-based applications where it is not possible to constrain a program to start after the termination of a subscriber. If we replace for instance $\overline{u}(b).Q_2$ with $\mathbf{announce}(b).Q_2$ and let $u(a).Q_1$ be the subscriber to the event b, then, the above agent evolves into $(Q_2.P) \| Q_1[b/a]$, which is completely different from the result obtained when there is no event announcement.

5.5 Shared Variable Interference

The EB style is intended for the loose coupling of components. This inspires that shared variables are not a primary issue in constructing methods for developing such applications. Yet, we argue that the issue of shared variables can not be ignored.

Let us illustrate a simple case in which loose coupling and shared variable are present. Two Java classes *Stack* and *Counter* are assumed where *Stack* contains the methods *push* and *pop* while *Counter* contains *increment* and *decrement*. The example is from [12, 11]. The meaning of these methods is indeed what the reader may expect. We require that *push* (resp. *pop*) announces an event of type *EventA* (resp. *EventB*) after performing its supposed task. Next, we construct an instance *somestack* of *Stack* and an instance *somecounter* of *Counter*. Using an EB middleware, we subscribe *somecounter.increment* (resp. *somecounter.decrement*) to events of type *EventA* (resp. *EventB*). The reader should agree that this is indeed a loosely coupled system: *Stack* knows nothing about *Counter* and vice versa.

To be convinced of the presence of shared variables interference, one observes that the result of *somestack.push* depends on whether it is executed alone or in parallel with *somestack.pop*, in which case there is indeed interference between *somestack.push* and *somestack.pop*. This kind of interference is indeed very difficult to master in the case of EB systems since the two methods *somestack.push* and *somestack.pop* may be invoked in a non-deterministic way by the environment.

6 Do we need "Yet another formalism"?

We have presented the requirements that we expect from a methodology that supports the EB architectural style. We have also presented some behavioral properties that contribute to making reasoning about the correctness of EB applications a non-trivial task. We are now armed to address the question of whether we need a new formalism for constructing such applications.

In general, we plead that there is the need for a new methodology that allows the development of reliable event-based applications. While a new formalism may not be required, we claim that an adequate logic is required that renders the computational behavior of the event-based paradigm unequivocally understood. We expect such a logic to illuminate the derivation and the inference of programs that conform to the EB computational model. Such a logic does not need to be orthogonal to existing logics, but should ideally be based on them such that existing tools can be leveraged.

Although such logics are often not widely used in practice, we believe that they give rise to sound and well founded less formal frameworks such as model checking and formal testing that are indeed more used in practice. Further, as acknowledged by Dingel et. [12, 11], the intuition gained in the development of formal techniques are of particular significance to software engineers.

7 Conclusions

The increasing use of the EB paradigm motivates the need for methodologies to support not only the construction of systems but also reasoning about their correctness. While some argue that reasoning about the EB paradigm is hard [20] other instead claim that no new methodology is required for constructing EB applications [19, 18]. The aim of this paper was to bring light on this starting "dispute."

Based on the requirements put on the EB style by its application domains, we deduce that there is need for a suitable methodology for EB applications. On the other hand, based on some identified behavioral aspects of this style, we plead that this methodology must be supported by a logic that allows a clear understanding of the EB paradigm.

Given that there are many factors that contribute to making it difficult to construct a methodology for the developing correct EB applications, following Michael Jackson [24], we further suggest that the EB architectural style be refined into architectural types [6] which are obtained from architectural styles by fixing some of their parameters.

This conclusion is guiding our work in the area of event-based applications [16, 15, 17, 14].

References

[1] G. D. Abowd, R. Allen, and D. Garlan. Using style to understand descriptions of software architectures. *ACM Software Engineering Notes*, 18(5):9–20, 1993.

[2] G. D. Abowd, R. Allen, and D. Garlan. Formalizing styles to understand descriptions of software architecture. *ACM Transactions on Software Engineering and Methodology*, 4(4):319–364, 1995.

[3] E. Anceaume, A. K. Datta, M. Gradinariu, and G. Simon. Publish/subscribe scheme for mobile networks. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 74–81. ACM Press, 2002.

[4] L. Andrade and J. Fiadeiro. Coordination – the evolutionary dimension. In *Proceedings of TOOLS Europe 2001*, pages 136–147. IEEE Computer Society Press, 2001.

[5] F. Arbab, M. Bonsangue, and F. de Boer. A coordination language for mobile components. In *Proceedings of the 2000 ACM Symposium on Applied Computing (SAC 2000)*, pages 166–173. ACM press, 2000.

[6] M. Bernardo, P. Ciancarini, and L. Donatiello. On the formalization of architectural types with process algebras. In *Proceedings of the 8th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 140–148. ACM Press, 2000.

[7] M. Bonsangue, J. Kok, and G. Zavattaro. Comparing coordination models based on shared distributed replicated data. In *Proceedings of the 1999 ACM Symposium on Applied Computing (SAC '99), San Antonio, Texas, USA*, pages 146 – 155. ACM press, February 1999.

[8] J. S. Bradbury and J. Dingel. Evaluating and improving the automatic analysis of implicit invocation systems. In *Proceedings of the 9th European software engineering conference held jointly with 10th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 78–87. ACM Press, 2003.

[9] G. Cugola and E. D. Nitto. Using a Publish/Subscribe Middleware to Support Mobile Computing. In *Proceedings of the Workshop on Middleware for Mobile Computing, in association with IFIP/ACM Middleware 2001 Conference, Heidelberg, Germany*, November 2001.

[10] G. Cugola, E. D. Nitto, and G. P. Picco. Content-based dispatching in a mobile environment. In *Proceeding of WSDAAL 2000, Ischia (Italy)*, Septenber 2000.

[11] J. Dingel, D. Garlan, S. Jha, and D. Notkin. Reasonning about Implicit Invocation. In *Proceedings of the 6th International Symposium on the Foundations of Software Engineering, FSE-6, Lake Buena Vista, FL*, pages 209–221. ACM, November 1998.

[12] J. Dingel, D. Garlan, S. Jha, and D. Notkin. Towards a formal treatment of implicit invocation using rely/guarantee reasoning. *Formal Aspects of Computing*, 10:193–213, 1998.

[13] D. F. D'Souza and A. C. Wills. *Objects, Components, and Frameworks with UML, The Catalysis Approach*. Addison Wesley Longman, Inc., 1998.

[14] P. Fenkam. *A Systematic Approach to the Development of Event-Based Applications*. PhD thesis, DSG, Technical University of Vienna, October 2003.

[15] P. Fenkam, H. Gall, and M. Jazayeri. A Systematic Approach to the Development of Event-Based Applications. In *Proceedings of the 22nd IEEE Symposium on Reliable Distributed Systems (SRDS 2003), Florence, Italy*. IEEE Computer Press, October 2003.

[16] P. Fenkam, H. Gall, and M. Jazayeri. Composing Specifications of Event Based Applications. In *Proceedings of FASE 2003 (Fundamental Approaches to Software Engineering 2003), Warsaw, Poland*, LNCS, pages 67–86. Springer Verlag, April 2003.

[17] P. Fenkam, H. Gall, and M. Jazayeri. Constructing Deadlock Free Event-Based Applications: A Rely/Guarantee Approach. In *Proceedings of FM 2003: the 12th International FME Symposium, Pisa, Italy*, LNCS, pages 632–657. Springer Verlag, September 2003.

[18] L. Fiege, M. Mezini, G. Muhl, and A. P. Buchmann. Engineering event-based systems with scopes. In *Proceedings of the 16th European Conference on Object-Oriented Programming (ECOOP 2002)*, volume 2374, pages 309–333. LNCS, 2002.

[19] L. Fiege, G. Muhl, and F. Gartner. A modular approach to building structured event-based systems. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC'02)*, pages 385–392, Madrid, Spain, 2002. ACM Press.

[20] D. Garlan and S. Khersonsky. Model checking implicit-invocation systems. In *Proceedings of the 10th International Workshop on Software Specification and Design, San Diego, CA*, pages 23–30, November 2000.

[21] D. Garlan, S. Khersonsky, and J. S. Kim. Model checking publish-subscribe systems. In *Proceedings of the 10th International SPIN Workshop on Model Checking of Software (SPIN 03), Portland, Oregon*, pages 166–180, May 2003.

[22] D. Garlan and D. Notkin. Formalizing design spaces: Implicit invocation mechanisms. In *Proceedings of Fourth International Symposium of VDM Europe: Formal Software Development Methods*, Noordwijkerhout, Netherlands, October 1991. LNCS 551.

[23] Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile enviroment. In *Second ACM international workshop on Data engineering for wireless and mobile access*, pages 27–34. ACM Press, 2001.

[24] M. Jackson. *Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices*. Addison-Wesley, 1995.

[25] P. Jalote. *An Integrated Approach to Software Engineering*. Springer Verlag, 1997.

[26] C. Jones. Tentative steps towards a development method for interfering programs. *Transactions on Programming Languages and Systems*, 5(4), October 1983.

[27] C. B. Jones. *Systematic software development using VDM*. Prentice-Hall International, 1990. 2nd edition.

[28] R. Milner. *Communicating and Mobile Systems: the pi-Calculus*. Cambridge University Press, May 1999.

[29] G. A. Papadopoulos and F. Arbab. Coordination models and languages. *Advances in Computers*, 46, 2001.

[30] D. von Oheimb. Hoare logic for mutual recursion and local variables. In V. R. C. Pandu Rangan and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *LNCS*, pages 168–180. Springer, 1999.