ENVIRONMENT MODELLING IN CLOSED SPECIFICATIONS OF EMBEDDED SYSTEMS

Mika Katara
Software Systems Laboratory
Tampere University of Technology
Finland

Arto Luoma
Department of Mathematics, Statistics and Philosophy / Statistics
University of Tampere
Finland

An embedded system is in constant interaction with its environment. It can consist of several, possibly distributed, components communicating with each other using interfaces. Collective behaviour between the system and its environment may be nondeterministic or random, and can include continuous quantities. The effects of the collective behaviour to the architecture of the system are non-obvious and should be considered before defining the interfaces between system components. This calls for methods capable of expressing complex collective behaviour and providing proper structuring of complex specifications. In this paper we discuss such capabilities in conjunction with the DisCo method.

1. Introduction

Embedded systems are inherently *reactive*. They are in constant interaction with their *environment*, i.e., the context in which they operate. These systems can consist of several, possibly distributed, components communicating using interfaces.

Interaction with the environment may include nondeterministic, random or continuous behaviour. These sources of complexity are usually hidden behind external interfaces. Conventionally, the specification process of an embedded system starts by identifying the external interfaces as well as the components of the system together with the internal interfaces between the components. The process continues by defining each component separately. However, it is often the case that as the developers gain more insight into the collective behaviour of components, the internal interfaces have to be revised. This often leads to other changes as well.

To alleviate the problem, the process should be started at a higher level of abstraction and the definition of interfaces should be postponed. The collective behav-

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35409-5_23

B. Kleinjohann (ed.), *Architecture and Design of Distributed Embedded Systems*

iour should be captured before partitioning the system into components and defining the interfaces (Kurki-Suonio&Mikkonen, 1998, 1999). This leads naturally to *closed specifications*, which are self-contained in the sense that they describe the behaviour of the total system, including the environment, rather than individual components.

Closed modelling does not mean that specifications cannot be modularised. However, modularity in specification languages is unlikely to be the same as in programming languages (Maibaum, 2000). Instead of reflecting the implementation architecture, closed specifications should exploit *abstractions* of the final system as units of modularity (Kurki-Suonio&Mikkonen, 1999). Consequently, different sources of complexity in the environment can be isolated into specification modules, which potentially affect many implementation-level components.

The method used in capturing collective behaviour should support expression of nondeterministic, random and continuous behaviour. Moreover, it should address effectively separation of concerns at high levels of abstraction. Nondeterminism is a built-in feature in many specification formalisms, so in this paper, the focus is on environment modelling especially concerning continuous and random behaviour in closed specification of embedded systems. Furthermore, an example of a distributed water tank system is given to illustrate structuring of complex specifications. In the sequel we will use DisCo, which is a specification method for real-time reactive systems. The rest of this paper is structured as follows. Section 2 discusses specification of embedded systems. In Section 3 the DisCo method is introduced and in Section 4 the example is given. Section 5 contains some concluding remarks.

2. From structural view to collective behaviour

2.1. Specifying embedded systems

Embedded systems have some characteristics which distinguish them from other systems. One of these characteristics is that they are inherently *reactive*. Reactive means that they are in constant interaction with their environment, i.e., the context in which they operate. Embedded systems may consist of several, possibly distributed components. These components communicate with each other using *internal interfaces* and with the environment of the system using *external interfaces*.

Interfaces provide means to implement *collective behaviour* between the components and between the system and its environment. Interfaces play a very important role in decomposing complicated systems. Well-defined and documented interfaces enable concurrent work, facilitate maintenance and enable reuse of components. Once defined the interfaces should not be changed. However, components and interfaces describe the *static structure* of the embedded system, not the collective behaviour.

2.2. Why to consider properties of the environment?

An embedded system may interact with its environment in complex ways. This interaction may include nondeterminism, randomness, continuous quantities etc. By non-

determinism we mean that no statistical properties can be found and by randomness that there are some statistical properties which can be described. Different sources of

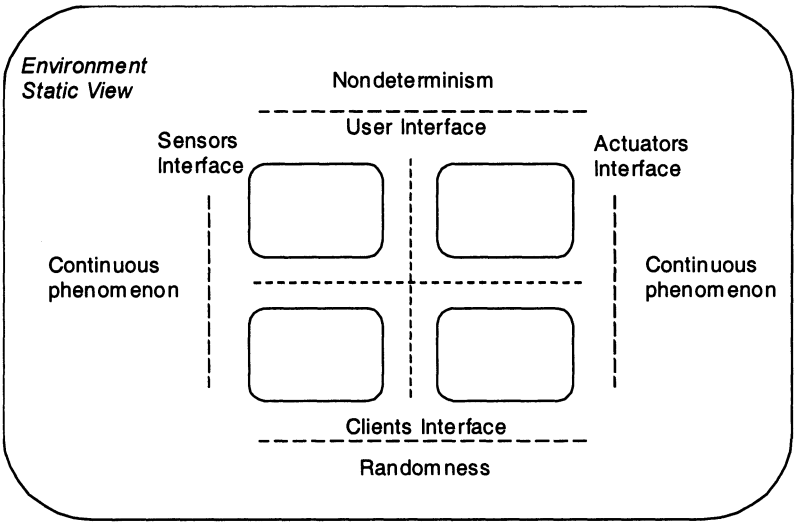

Figure 1. Structural view to an embedded system.

complexity in the environment may pose non-functional requirements to the architecture of the system. These requirements may involve, for example, timeliness, performance and reliability. The affects of these requirements to hardware/software partitioning, for example, are usually non-obvious.

As an example, a fly-by-wire or an anti-lock breaking system has interaction with a user (driver or pilot) and with physical phenomena. Consider a real-time requirement for bounded response requiring that within certain time interval from a user activity certain changes in the physical environment occur. Obviously, for the architecture of the system, effects of such a requirement are non-trivial.

2.3. From structural view to collective behaviour

In Figure 1, a structural view to an embedded system is depicted. This is usually the most abstract view to the system consisting of components, and internal and external interfaces. Nondeterministic behaviour of a user and random behaviour of clients requesting service is hidden behind interfaces. Moreover, continuous behaviour is hidden behind interfaces, which provide access to the sensors and actuators. Given this kind of view to the system, a designer has to come up with well-defined and preferable permanent internal interfaces. However, as the developers gain more insight into the collective behaviour between the components and between the components and the environment, there might rise a need to revise the interface definitions, which probably leads to other changes as well.

To alleviate the problem, after the requirements phase, the specification of the sys-tem should be started at a higher level of abstraction. Definition of interfaces

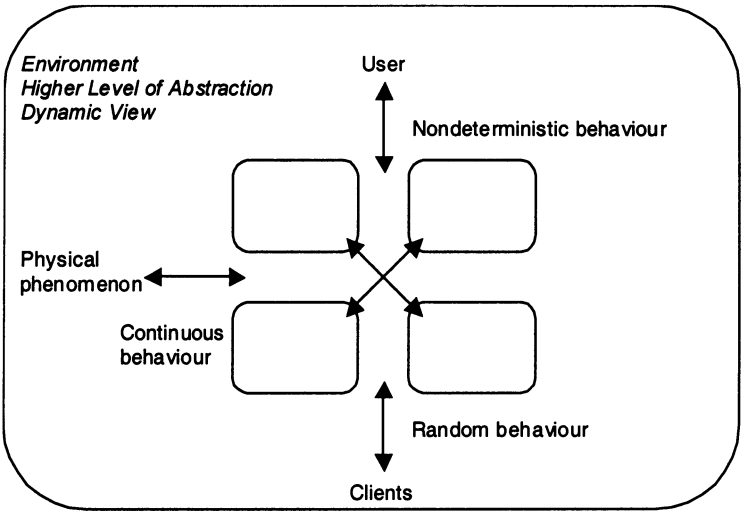

Figure 2. Collective behaviour view.

between components should be postponed. The collective behaviour should be captured before partitioning the system into components with fixed interfaces (Kurki-Suonio&Mikko-nen, 1998, 1999). After partitioning, the components can be further designed using conventional methods. Those parts of the closed specification which belong to the environment need no explicit implementation.

In Figure 2 the capturing of collective behaviour is depicted. Interfaces are not defined yet, but components of the system and entities of the environment, e.g. users, clients, and physical phenomena, are described in a very abstract way. This view does not describe static structure but dynamic co-operation. It may include nondeterministic behaviour with the user, random behaviour with the clients and continuous behaviour with the physical phenomenon. The focus is on *what* the collective behaviour is rather than *how* it is implemented (Kurki-Suonio&Mikkonen, 1998). After capturing this view, the definition of internal interfaces is more likely to succeed.

In order to be useful in capturing complex collective behaviour, a specification method should allow expression of such behaviour. However, this may lead to specifications that are hard to understand and reuse. Therefore, the method should support separation of concerns in such a way that different sources of complexity in the environment can be specified modularly. Also, reuse of such modules should be possible.

In closed specifications, the natural units of modularity are not the units of the implementation architecture, but logical abstractions of the final system (Kurki-Suonio&Mikkonen, 1999). Consequently, different sources of complexity should be isolated into reusable specification modules.

3. The DisCo method

DisCo (Järvinen et al., 1990 & DisCo, 1999) is a state-based formal specification method for reactive and distributed real-time systems. It is based on the joint action

theory (Back&Kurki-Suonio, 1988, 1989) where the focus is on capturing collective behaviour at a high level of abstraction. The semantics of DisCo is defined in the Temporal Logic of Actions (TLA) (Lamport, 1994). Concurrency is modelled by interleaving and specifications have an operational interpretation.

The basic notions of the DisCo language are *classes* and *multi-object actions*. One or more objects, which are instances of classes, *participate* in actions. As an example, definitions of class C and action A are given below:

$$class\ C = \{i : integer\}$$

$$action\ A(c1, c2, c3 : C; r : real) :$$
$$r > 4.6$$
$$\rightarrow c1.i' = c2.i' = c3.i' = max(c1.i, c2.i, c3.i),$$

where the unprimed names refer to the values of variables in the state before the action execution and primed in the state following the action execution. If the *guard* of the action, $r > 4.6$ in the above, evaluates to true, the action is said to be *enabled*, i.e., it can be executed. After execution of action A the values of i attributes of the participating objects equal the maximum of values before the action was executed.

DisCo specifications are generic in the sense that an unbounded number of objects are assumed. Furthermore, the action to be executed next, its participants and parame-ter values are chosen nondeterministically. This gives basis for specifying reusable patterns of collective behaviour. Specifications are refined from a high level of abstraction towards implementation description using stepwise refinement. The refinement mechanism is *superposition*, which preserves safety properties ("something bad never happens") by construction. One refinement step is described in a *layer*, which is a unit of modularity. A system is always specified together with its assumed environment, i.e., specifications are closed. Also, specifications can be *composed* and actions *synchronized* to be executed in parallel. To illustrate the use of superposition, C is extended with attribute b, and new conjuncts are added to the guard and body of A:

$$C = C + \{b : boolean\}$$

$$action\ A(c1, c2, c3 : C; r : real) : refines\ A(c1, c2, c3, r)$$
$$3.5 < r$$
$$\rightarrow c1.b' = c2.b' = c3.b' = true.$$

Logical abstractions can be given as different *logical layers* (Mikkonen, 1999) affecting potentially many implementation-level components. Because DisCo specifications are operational, their finite instances can be validated using simulation. The method includes tool support for graphically animated simulation of specifications (Systä, 1991). Furthermore, the formal basis enables verification using theorem proving (Kellomäki, 1997) and model checking (Aaltonen et al., 2000).

3.1. Modelling time

Real time is usually considered to be the most important continuous quantity at least concerning embedded systems. It can be incorporated in the above scheme as follows (for more detailed discussion, see (Kurki-Suonio&Katara, 1999)). It is assumed that actions are executed instantaneously. A clock variable Ω belonging to the set of nonnegative reals and initialised as 0, records time from the beginning of a behaviour. In each action, the time when it is executed is given by the value of an implicit parameter τ. Furthermore, guards of all actions are implicitly strengthened by the conjunct $\Omega \leq \tau \leq min(\Delta)$, where Δ denotes a multiset of *deadlines*. Additionally, conjunct $\Omega' = \tau$ is added to bodies of all actions.

Both minimal separation and bounded response requirements can be expresses using these constructs. Minimal separation between executions of actions A and B can be required by strengthening the guard of action B by conjunct $\tau \geq \tau_A + d$ where τ_A denotes the most recent execution moment of A. Deadlines are needed for bounded response requirements. When a deadline $\tau + d$ is required a conjunct of the form $x' = \Delta_{on}(d)$ is given in the action body. It adds the deadline to Δ and stores it in a variable x. Until some action removes the deadline with $\Delta_{off}(x)$, an implicit conjunct $\tau \leq min(\Delta)$ in all guards prevents advancing Ω beyond this deadline. Initially, Δ can hold initial deadlines. A type time, a synonym type of real, can be used in timed specifications.

3.2. Modelling other continuous quantities

In TLA values of state variables can change only in actions. However, we can think of storing *samples* of a continuous quantity in a state variable. Periodic sampling can be modelled as a periodic action with non-deterministic parameter value representing the value of the quantity at a moment when the action is executed. The samples and time can be used to give an approximation of the quantity (Kurki-Suonio, 1993).

Modelling an action that has to be executed at a moment when the quantity reaches some limit poses a problem in determining the right moment of time when the execution should happen. Because the moment the quantity reaches the limit is not known beforehand, a time deadline for the action cannot be given.

Time deadlines can be generalized to other continuous quantities as well. It is assumed that between executions of actions each continuous quantity of interest changes *monotonically*. This means that between executions of actions their values are either non-increasing or non-decreasing. For each continuous quantity q of interest, multisets Δ^{q-} and Δ^{q+} are introduced to hold lower and upper *hybrid deadlines* concerning the value of the quantity. Additionally, all guards are strengthened by conjunct

$$max(\Delta^{q-}) \leq p_q \leq min(\Delta^{q+}),$$

where p_q is the parameter corresponding to the quantity. In (Katara, 2000) these constructs were used to model physical mobility.

3.3. Modelling stochastic behaviour

In order to capture randomness occurring in the environment, it should be possible to address statistical properties at the specification level. One way to introduce ran-

domness is to let action parameters exhibit some mean, variance and distribution. As an example consider a generic Poisson process modelled as follows:

$$\begin{aligned} &\textit{class Poisson} = \{\lambda : \textit{constant real};\\ &\qquad\qquad t : \textit{time}\}, \end{aligned}$$

where λ is the rate of the process and t is used to store a deadline for the next Poisson arrival. Poisson arrival at rate λ is modelled as an action:

$$\begin{aligned} &\textit{action Poisson\_Arrival}(p : \textit{Poisson}; d : \textit{real}(\textit{exp}, 1/p.\lambda)):\\ &\qquad \tau \geq p.t\\ &\qquad \rightarrow \Delta_{off}(p.t)\\ &\qquad \wedge\ p.t' = \Delta_{on}(d). \end{aligned}$$

The values of parameter d are exponentially distributed with mean $1/p.\lambda$, where p is a participant belonging to the class Poisson. The value of the parameter is used to determine the next moment the action is executed.

However, TLA does not support expression of statistical properties. This means that stochastic parameter values lack formal semantics. Nevertheless, statistical analysis can be employed. Also, stochastic parameter values could be used in simulation.

4. Example of separating concerns

As an example of separating concerns at a high level of abstraction a simple example of an embedded system with a complex environment is presented. The system consists of distributed water *tanks* and *customers,* which visit tanks and consume water. As customers consume water from a tank, the tank's water level decreases and it can order more water from a *truck.* However, there is a considerable delay before the truck arrives and filling may be started. Furthermore, tanks may leak. The objective is to develop a system which guarantees that the tanks never run out of water. Because of lack of space, only a brief summary of the specification is given. For the full specification, the reader is referred to (DisCo, 1999).

In Figure 3 the structure of the specification is depicted. The specification starts from a simple layer modelling only the functional aspects of the collective behaviour. Layer Functional Specification includes the definition of classes tank, customer and truck, associated relations, and some initial conditions which have to hold in the initial state. There are also five actions: *Start_Service* and *Stop_Service*, which model arrival and departure of customers, *Order*, which models ordering of a water delivery, and *Start_Filling* and *Stop_Filling* modelling filling of a tank.

There are also three generic layers. Layer Aperiodic Events uses constructs introduced in Section 3.1. to define the generic class and actions to model scheduling an aperiodic event (*A_Schedule*) and triggering it (also Kurki-Suonio&Katara, 1999). The time interval between scheduling and triggering is given as a parameter d in ac-

tion *A_Schedule*. Actions $A\_Trigger_{\leq}$ and $A\_Trigger_{=}$ model triggering. The former is used when the action corresponding to the event can be executed before *d* has passed and the latter when exactly time *d* must have passed from scheduling.

As described in Section 3.2, time deadlines can be generalized to other continuous quantities as well. Layer Continuous Behaviour describes generic continuous quantity

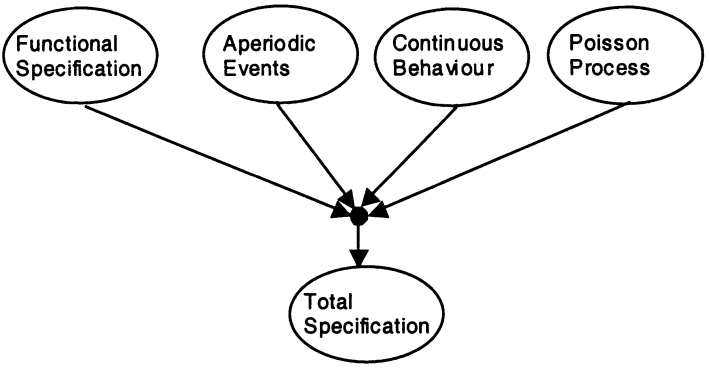

Figure 3. Layers of the distributed water tank example.

including scheduling and triggering actions for both lower ($Schedule^-$ and $Trigger^-$) and upper ($Schedule^+$ and $Trigger^+$) hybrid deadlines concerning the values of the quantity. Furthermore, layer Poisson Process contains the generic class and action introduced in Section 3.3.

The total specification is obtained by composing the functional layer with the three generic layers and applying superposition to the composition. Instantiation of the generic parts is done in the final layer. Moreover, it is required that in the initial state there is a lower hybrid deadline corresponding to a notice level where more water should be ordered. Action *Start_Service* is synchronized with *Poisson_Arrival* and *A_Schedule*, and action *Stop_Service* with $A\_Trigger_{=}$. Using stochastic parameter values, mean and variance for the service time are given.

Moreover, action *Order* is synchronized with $Trigger^-$ and *A_Schedule*, modelling reaching the notice level and scheduling a water delivery. *Start_Filling* is synchronized with $A\_Trigger_{\leq}$ and $Schedule^+$, to model arrival of the water delivery and setting an upper hybrid deadline that equals the capacity of the tank. Furthermore, *Stop_Filling* is synchronized with $Trigger^+$ and $Schedule^-$, conforming to the water level reaching the capacity of the tank and setting a lower hybrid deadline for the notice level again.

To find a correct value for the notice level, we cannot solely resort to the worst-case scenario. An unbounded number of customers could arrive at the moment action *Order* has been triggered and consume all the water at once. To illustrate how statistical analysis can be used in this context, we have modelled water consumption using a *compound Poisson process* (see, for example Ross, 1997). Water consumption by time *t* is given by the formula

$$X(t) = \sum_{i=1}^{N(t)} Y_i \qquad ,t \geq 0,$$

where $N(t)$ is the number of customers who arrive by time t and Y_i is the amount of water the ith customer takes. We assume that $\{N(t), t \geq 0\}$ is a Poisson process and $\{Y_i, i \geq 1\}$ a sequence of independent and identically distributed (IID) variables, independent also of the process $\{N(t)\}$.

The compound Poisson variable $X(t)$ has the mean $\lambda t E[Y_1]$ and variance $\lambda t(Var(Y_1)+[E(Y_1)]^2)$ where λ is the rate of the Poisson process $\{N(t)\}$, and $E(Y_1)$ and $Var(Y_1)$ are the mean and variance of Y_1. The mean and variance of the water consumption of one customer can be easily obtained from the mean and variance of the time they receive service, if we assume that the rate of water flow from the tank to the customer is constant.

Using Chebyshev's inequality it can be estimated that the probability of water running out before the water delivery arrives is less than $p^*(x)=Var(X(t))/[2(x-E(X(t)))^2]$, where x is the notice level of the tank and t is the maximum delay before refilling the tank. The number of fillings of the tank during a longer time period T is approximately $k(x)=E(X(T))/[V-(x-E(X(t)))]$, where V is the capacity of the tank. Thus, the probability that the water does not run out in time T is approximately

$$p = (1 - p^*(x))^{k(x)}.$$

It can be seen that the probability p is a function of the notice level x. If p is given, the suitable notice level x can be solved numerically.

Suppose that the capacity of the tank is 10 000 litres, the maximum delay of water delivery is 10 hours and we want the probability of water not running out during one year to be 0.9. If two customers come per hour on the average and the amount of water one customer takes in litres has mean 20 and variance 20, it can be numerically solved that the notice level should be at 1665 litres.

5. Discussion

Collective behaviour between a system and its environment affects the architecture of the system. Therefore, it should be captured before defining the interfaces between components of the system. The interaction with the environment may include complex behaviour. However, many specification methods lack support for expressing properties of the environment such as stochastic or continuous behaviour. In this paper we have investigated how to incorporate stochastic and continuous modelling into the DisCo method, which uses logical layering to modularise specifications.

We have presented some ways to model complex environment behaviour. However, as the underlying temporal logic does not provide direct means to express statistical properties, advantages of formal specification are easily lost. Yet, this does not rule out the use of classical statistical analysis and simulation as ways to validate system behaviour. Furthermore, the example given shows some evidence that logical layering provides useful separation of concerns also in the presence of complex behaviour.

There is a lot of ongoing work in the area of hybrid stochastic systems. One future direction would be trying to find a mapping between DisCo and some formalism capable of expressing hybrid stochastic behaviour but not necessarily focusing on

collective behaviour. Moreover, there is a lot to be done to develop user-friendly tools to support modelling of continuous and stochastic behaviour.

Acknowledgements

Helpful comments and creative ideas by the members of the DisCo project as well as funding from Tampere Graduate School in Information Science and Engineering (TISE) and Academy of Finland (project 57473) are gratefully acknowledged.

References

Aaltonen, T., Katara, M., Pitkänen, R. (2000). Verifying real-time joint action specifications using timed automata. In *16th World Computer Congress 2000, Proc. Conference on Software: Theory and Practice*, pp. 516-525, Beijing, China, August 2000. IFIP.

DisCo (1999). The DisCo project WWW page. At URL http://disco.cs.tut.fi, 1999.

Back, R.J.R, Kurki-Suonio, R. (1988). Distributed cooperation with action systems. *ACM Transactions on Programming Languages and Systems*, 10(4):513-554, October 1988.

Back, R.J.R., Kurki-Suonio, R. (1989). Decentralization of process nets with centralized control. *Distributed Computing*, 3:73-87, 1989.

Järvinen, H.-M., Kurki-Suonio, R., Sakkinen, M., Systä, K. (1990). Object-oriented specification of reactive systems. In *Proc. 12th International Conference on Software Engineering*, pp. 63-71. IEEE Computer Society Press, 1990.

Katara, M. (2000). Hybrid models for mobile computing. In *Proc. COORDINATION 2000*, number 1906 in LNCS, pp. 216-231. Springer-Verlag, 2000.

Kellomäki, P. (1997). Verification of reactive systems using DisCo and PVS. In *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods*, number 1313 in LNCS, pp. 589-604. Springer-Verlag, 1997.

Kurki-Suonio, R. (1993). Hybrid models with fairness and distributed clocks. In *Hybrid Systems*, number 736 in LNCS, pp. 103-120. Springer-Verlag, 1993.

Kurki-Suonio R., Katara, M. (1999). Logical layers in specifications with distributed objects and real time. *Computer Systems Science & Engineering*, 14(4):217-226, July 1999.

Kurki-Suonio R., Mikkonen, T. (1998). Liberating object-oriented modeling from programming-level abstractions. *Object-Oriented Technology, ECOOP'97 Workshop Reader*, number 1357 in LNCS, pp. 195-199. Springer-Verlag, 1998.

Kurki-Suonio, R., Mikkonen, T. (1999). Harnessing the power of interaction. In *Information Modelling and Knowledge Bases X*, pp. 1-11. IOS Press, 1999.

Lamport, L. (1994). The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872-923, May 1994.

Maibaum, T. (2000). Mathematical foundations of software engineering: a roadmap. In *Future of Software Engineering*, pp. 163-172. ACM, 2000.

Mikkonen, T. (1999). *Abstractions and Logical Layers in Specifications of Reactive Systems*. Doctoral dissertation. Tampere University of Technology, 1999.

Ross, S.M. (1997). *Introduction to Probability Models*. Academic Press, sixth edition, 1997.

Systä, K. (1991). A graphical tool for specification of reactive systems. In *Proc. Euromicro'91 Workshop on Real-Time Systems*, pp. 12-19, Paris, France, June 1991. IEEE Computer Society Press.