An Extension to State Machine Modeling: The Concept of Coupled State Machines

Dominikus Herzberg*
Ericsson Eurolab Deutschland GmbH
Ericsson Allee 1
52134 Herzogenrath, Germany
Dominikus.Herzberg@eed.ericsson.se

André Marburger
Aachen University of Technology
Department of Computer Science III
52074 Aachen, Germany
marand@i3.informatik.rwth-aachen.de

Abstract

A layer in a communication system has at least three main interfaces: (1) A Service Access Point (SAP) for providing services to a user of an upper layer, (2) an inverse SAP for accessing services provided by a lower layer, and (3) a logical interface for communication purposes to a remote peer entity, which we call Connection Endpoint (CEP). The CEP actually specifies the communication protocol. In its entirety, the full specification of the communication layer results in a huge, complex state machine. It would be beneficial to individually specify the different interfaces of the communication layer and smoothly couple the specifications. Current modeling languages like the Unified Modeling Language (UML) do not offer a satisfying solution. In this paper, an extension is presented, which provides an approach to couple finite state machines. It enables a modeler not only to specify different aspects of a communication layer but also to describe their coordination and synchronization. Furthermore, the concept of coupled state machines can be used for other purposes as well, e.g. for specifying Application Programming Interfaces and modeling the observer pattern.

1 Introduction

1.1 Background

On an architectural level, any data or telecommunication system can be structured according to two different directions of communication, "vertical" and "horizontal". "Vertical" communication refers to the exchange of information between layers. The "point" at which a layer publishes its services for access to an "upper" layer is called *Service Access Point* (SAP). In contrast, "horizontal" communication refers to the exchange of information between remote peers. Remote peers are physically distributed, they reside in different nodes, and communicate with each other according to a protocol. We call the "point" describing the protocol interface *Connection Endpoint* (CEP). Note that the concept of a protocol is well-known and generally defines a set of messages and rules (see e.g. [2, p.191]); however, it has a special meaning in data and telecommunications. Whereas software engineers associate a reliable, indestructible communication relation with the term "protocol", data and telecommunication engineers are faced with the "real" world: They have to add error correction, connection control, flow control and so on as an integral part to the protocol. A communication relation between remote peers can always break, be subject to noise, congestion etc. This is the reason why communication engineers introduced protocol stacks, with each protocol level comprising a dedicated set of functionality, thereby "stackwisely" abstracting the communication service. These stacks naturally give means to "vertically" divide a node into layers.

Consequently, three main interfaces completely describe the behavior of a node layer from an outer perspective, each interface covering a specific aspect of the communication relation, see figure 1. The SAP, denoted by a filled diamond symbol, provides layer (N) services by means of so-called *service primitives* to a service user, the upper layer $(N+1)$. Service primitives can be implemented as procedure calls, library calls, signals, methods etc., which is a design decision. The CEP, symbolized by a filled circle, describes the "horizontal" relation to another remote peer entity. A CEP holds the specification of a communication protocol such as the Transmission Control Protocol (TCP) [25] or the Internet Protocol (IP) [24]. In fact, we will exemplify the topic of discussion on TCP. Be aware that the CEP is purely virtual and represents a logical interface only. All proto-

*This work is being funded by Ericsson and run in cooperation with the Department of Computer Science III, Aachen University of Technology, Germany.

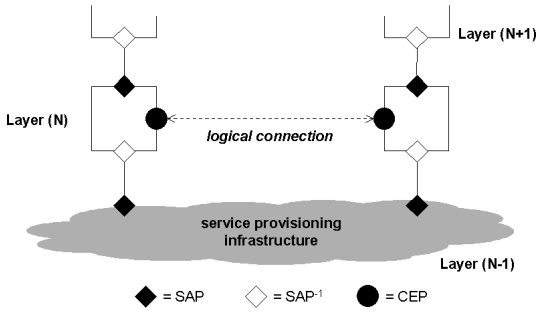

Figure 1. A simplified communication model according to OSI RM

col messages are transmitted using the services of a lower layer. This interface function is given by the inverse SAP (SAP^{-1}), which uses services from a lower layer $(N-1)$ by accessing the $(N-1)$-SAP; it is depicted by an "empty" diamond symbol.

The model described is based on the OSI (Open Systems Interconnection) Reference Model [16], which has laid a solid foundation for understanding distributed system intercommunication [3]. The notation used for the SAP and SAP^{-1} is an extension of ROOM (Real-Time Object-Oriented Modeling) [26]; for a thorough discussion see [13].

1.2 The Problem

Given the model presented, one faces some important problems in modeling the behavior of a layer in a communication system. There are in principle two alternatives for specifying layer (N) [6]. For this discussion, we assume that Finite State Machines (FSM) according to the Unified Modeling Language (UML) [23] are the primary means to describe behavioral aspects. FSMs are a common tool for specifying protocols [15].

- **Black box view:** Specifying a layer in a *black box* manner means that we give a complete description of the behavior of each and every external interface. In which case, the CEP, the SAP, and the SAP^{-1} are each specified by an FSM; which is a precise description of the remote peer protocol and the two interface protocols. Even though this view is ideal from a modeling point of view, the problem is that such a black box model can neither simulate nor explain the interface interaction without being wired with the internal behavior.

- **White box view:** Specifying a layer in a *white box* manner means that we define a more or less huge and complex FSM that gives a complete specification of the internals driving the external behavior. As a result, the communication at an external interface cannot be understood without looking inside the layer; at best, a list of messages (or service primitives) going in and out at the external interface can be declared. This corresponds to the notion of an interface in UML, see e.g. [4, p.155ff.]. Here, the problem is that the FSM is difficult to structure in a way, so that at least internally the behavioral aspects of the external interfaces are made visible.

Furthermore, the virtual CEP interface is some sort of a hybrid interface; it has to cover two modes of operation in the layer model:

- **Abstract view:** Without accessing any provisioning services of a lower layer, the CEP is needed to describe and possibly simulate node interaction; this is a purely *abstract* view of the protocol layer interconnection. In this case, the CEP is connected to a logical entity, which models the channel connection including transmission characteristics.

- **Concrete view:** On the other hand, if the provisioning layer is attached, which it always is from a *concrete* point of view, the CEP becomes "inactive"; its function is taken over by the SAP^{-1}. The CEP may just indicate, which protocol messages are virtually being sent out, it no longer plays an active role.

It is not a trivial exercise to completely reconcile the abstract and the concrete view in a single model. We are not going to solve this here. A formal treatment and discussion of compositional refinement can be found in e.g. [7, 20].

What the problems listed above have in common is that different views change scope and redefine how state machines are coupled with each other. In the case of *abstract vs. concrete*, the CEP FSM should be either coupled to a logical channel entity or to the inverse SAP. The *black box view* requires the decoupling of the external interface FSMs from the logic of the internal layer FSM and the coupling of the external FSMs to describe their interaction. And finally, the *white box view* suffers from structural means to divide internal FSMs as much as possible.

The coupling problem is of theoretical as well as practical relevance. An Ericsson internal study on the use of modeling languages for service and protocol specifications accurately pinpoints this problem. It is one of the reasons, why modeling languages like the UML have not yet successfully penetrated the systems engineering domain. System designers of data and telecommunication systems do not find reasonable support in today's modeling languages for their problem domain.

The solution we propose, in order to enable FSM separation along with smooth coupling, is based on the idea of Trigger Detection Points (TDP); they are introduced as a concept extending state machine modeling. TDPs were motivated by the concept of detection points in [1]. Our approach is mainly driven by a *black box view* on a communication layer. In section 2 we will elaborate on the model presented in figure 1 by using a practical example. There, we will specify the TCP layer of a data communication system and show how all external interfaces (the CEP, the SAP, and the inverse SAP) can each be described by an FSM. Section 3 demonstrates the coupling of FSM specifications via TDPs and Trigger Initiation Points (TIP). In section 4, related approaches are discussed. Finally, section 5 closes with some observations and conclusions.

2 Specifying the External Interfaces of a Communication Layer

The TCP protocol serves as an excellent example for discussing layer design and specification problems. It is simple to understand, easy to read (the technical standard [25] is summed up in less than one hundred pages)[1], publicly available, and – most important – it is widespread and one of the most used protocols worldwide. Together with IP, the TCP/IP protocol suite forms the backbone of the Internet architecture.

Looking at how the TCP standard [25] specifies the protocol, unveils a typical problem: It presents the whole layer by a state machine and does not clearly separate the TCP protocol from its user (or application) interface. Both are combined, see figure 2. It stems from a white box view (as mentioned previously). The figure uses a compact notation and shows both the server FSM and the client FSM. It reads as follows: When a user in his role as a server submits a LISTEN command, the state changes from CLOSED to LISTEN. If, on the other hand, the client user submits a CONNECT, the TCP protocol sends out a message with the synchronization bit SYN set to one, and the client's state changes to SYN SENT. On receipt of the TCP message with SYN equal to one, the server sends out a TCP message with SYN and ACK (the acknowledgment bit) set to one and changes to state SYN RCVD. When the three-way handshake is successfully completed, both parties end up in the state ESTABLISHED and are ready to send and receive data. This short description of figure 2 neglects a lot of details of TCP (e.g. timeouts, which are important to resolve deadlocks) but is sufficient for the purpose of our discussion. The interested reader may consult [27] for more information.

Figure 3 and 4 demonstrate the division of the TCP layer into two separated FSMs, each covering the functionality of its related external interface. It is the prerequisite to change from a white box view to a black box view. Figure 3 displays the SAP of TCP. Instead of using the TCP service commands LISTEN, CONNECT, SEND etc., we decided to convert them to service primitives, which are more narrative. Again, the client and the server side are combined in the SAP FSM. From a user's viewpoint the communication with the client/server SAPs looks as follows: When a user requests a connection (Conn.req), the client's SAP changes to state C-PENDG. The server receives notification of the connection request via a connection indication (Conn.ind) and may respond with Conn.res, accepting the request. This is confirmed to the client via Conn.con and finally, the SAPs end up in state DATA. Note that neither the user of the client SAP nor the user of the server SAP see the underlying TCP protocol being used. They only see the SAP interface; the layer and its use of TCP appears as a black box.

The logical CEP holds the protocol specification of TCP, see figure 4. It also follows the black box principle. That is why there is for example no indication as to what might have triggered the transition from CLOSED to SYN SENT at the client's side; but when the transition is triggered it sends out a TCP message with the SYN bit set. Otherwise, figure 4 is similar to figure 2 but with the difference that all the numerous SAP related details have been stripped off. To

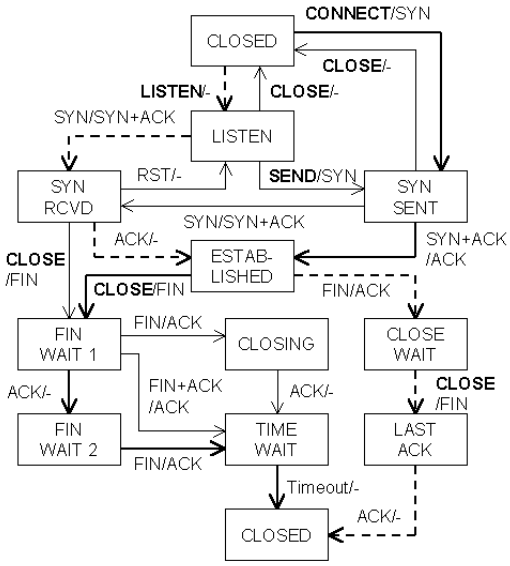

Figure 2. The TCP FSM figure is derived from [27, p.532]. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events. User commands are given in bold font.

[1] Clarifications and bug fixes are detailed in [5], extensions are given in [18].

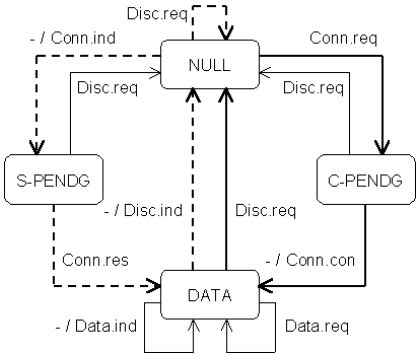

Figure 3. The SAP of the TCP layer. The shortcuts stand for connect and disconnect; the postfixes stand for request, confirmation, indication, and response

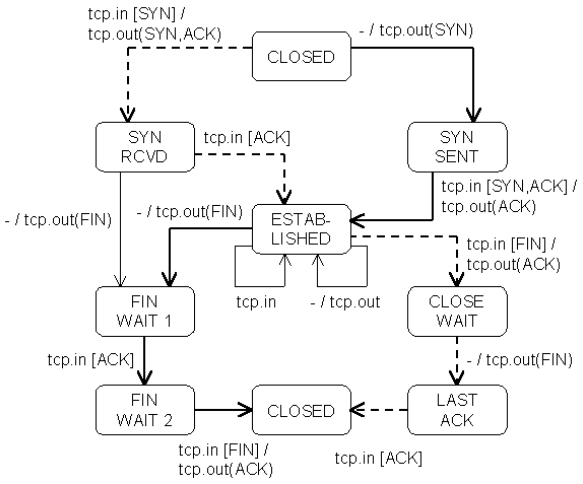

Figure 4. The CEP holding the protocol specification of TCP

reduce complexity, we slightly simplified the TCP protocol specification and added transitions to the data transfer state ESTABLISHED.

As described, TCP calls on a lower level protocol module to actually send and receive information over a network. This lower level protocol module is usually IP. To avoid cluttering up the discussion and distracting the reader with too many FSMs, we intentionally left out an example figure. Since IP is a connectionless protocol and basically consists of just two commands (SEND and RECV, see [24, p.31ff.]), the TCP SAP^{-1} could be modeled with one up to three states. In the case that TCP does not sit on top of IP but rather on top of a connection-oriented lower level protocol, the SAP^{-1} would look pretty much the same as its SAP interface. Note that the following discussion on coupling the SAP with the CEP applies without restriction to the SAP^{-1} as well.

3 The Concept of Coupled State Machines

We managed to specify TCP from an external point of view, which in itself is already an achievement. All internals of TCP like flow control and buffering, congestion control, fragmentation, error control, window flow control etc. are hidden and subject of a detailing white box view. However, for model understanding it would be beneficial to show how the different interfaces of the communication layer interact with each other without referring to any internals. As was shown by Ericsson's language study, it is usually the "inside", which drives the "outside". We are looking for a way that allows the modeler to keep a purely external view.

One way to couple the individual FSMs is by the usual event messaging mechanism provided by UML, that means by signals and/or call events. The drawback of this approach is that one would again tightly connect the FSMs. For example, the Conn.req transition of the SAP (see figure 3) needs to have an activity attached that sends a signal to the CEP (see figure 4). This signal would then represent the CLOSED/SYN SENT transition that triggers the tcp.out message. As a result, the FSM of the CEP would more or less turn out to be the original TCP FSM and finally look like figure 2. In other words, the modeler would not be better off, and splitting of the TCP FSMs seems to be an academic exercise only. Clearly, another technique is needed.

Our solution to this problem is the introduction of so-called Trigger Detection Points (TDPs) and Trigger Initiation Points (TIPs). A TDP can be attached at the arrow head of an transition in a statechart diagram; it detects whenever this specific transition fires and broadcasts a notification message to all corresponding TIPs. TDPs are notated by small filled boxes, see e.g. figure 5. A TIP can be attached at the beginning of the transition arrow and triggers the transition to fire. An *active* TIP stimulates the transition to fire on receipt of a TDP notifier independent of the transition's event-signature. That means, that either the event specified by the transition's event-signature *or* the TIP can trigger the transition. Active TIPs are visualized by small filled triangles, see e.g. figure 6. Conversely, *passive* TIPs have a locking mechanism and can be meaningfully used only with "normal" transitions, i.e. the transition explicitly requires an event-signature. The transition cannot fire unless the TIP's corresponding TDP has been passed *and* unless the transition's event has been received. The order of occurrence is irrelevant, it is just the combination of the TIP event *and* the transition event, which unlock the transition and let

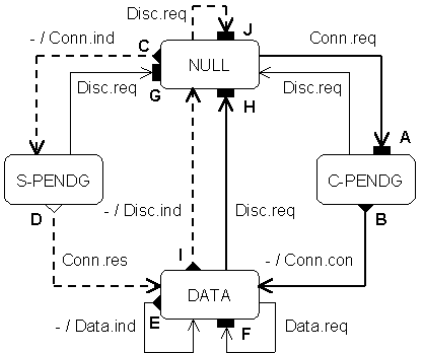

Figure 5. The SAP of TCP extended by TDPs and TIPs

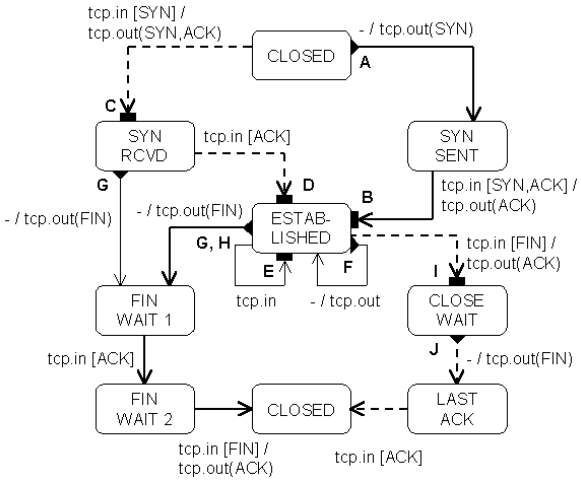

Figure 6. The CEP holding the protocol specification of TCP, extended by TDPs and TIPs

it fire. Passive TIPs behave like a logical “and” to synchronize a transition, whereas active TIPs realize a logical “or”. An example of an passive TIP can be found in figure 5; it is pictured by a small, “empty” triangle. In general, the relation of a TIP and a TDP is given by a name consisting of a single or more capital letters. Note that one or more TIPs may be related to a single TDP.

Now, the coupling of the SAP and the CEP can be easily described, see figure 5 and 6. For example, when a client user sends a Conn.req to the SAP, TDP A detects the transition NULL to C-PENDG firing and broadcasts a notifier event to all corresponding TIPs. The notifier event causes the CEP to fire the CLOSED/SYN SENT transition and results in sending out a TCP message with the SYN bit set to one; the rest of the scenario is straight forward. However, some explanations should help understand the purpose of a passive TIP. Let us assume, that the protocol at the server side has just entered state SYN RCVD, which triggers TIP C at the server SAP and results in a connection indication (Conn.ind) to the SAP user. Now there are two concurrent and competing threads. The user of the server SAP may either accept the connection indication and answer with Conn.res or, alternatively, the user may deny the request and answer with a Disc.req. Concurrently, on the protocol thread, the server’s CEP enters state ESTABLISHED at some point in time. It is the passive TIP D that prevents the SAP FSM entering DATA on Conn.req unless the protocol has reached ESTABLISHED. On the other hand, if the user has decided to reject the connection indication (Conn.ind) via Disc.req, the CEP starts the disconnect procedure based on the TDP G trigger. All this could not be done using conventional messaging without changing the FSMs.

The advantage of using TDPs and TIPs is that the FSMs remain autonomous but get coupled. They can notify each other about important state changes and use them for synchronization purposes; there is no need to introduce new event messages and modify transitions. TDPs and TIPs could be interpreted as some sort of annotations (with precise semantics), which specify FSM interaction and coordination. The modeler does not need to modify the original interface specification or reference to any internal “engine” driving the whole. If the broadcasting mechanism of TDP events can be directed, it is possible to couple external interface FSMs with layer internal FSMs reusing the same set of TDPs and TIPs. That means, that a black box and a white box view could harmoniously coexist without blurring the difference between both views.

4 Discussion of Related Approaches

It can be shown that TDPs and TIPs can be smoothly integrated in an event driven execution model for FSMs. The prototype we developed at Ericsson (programmed in Python [21]) treats TDPs as a specialization of messages, see figure 7, and adds notifier events to the event stack. The implementation of active and passive TIPs required only a few modifications to the event handler. Based on this implementation a comparison to other alternatives in modeling finite state machines is straight forward.

One of the most prominent modeling approaches for finite automatons are Harel’s *statecharts* [10]. Statecharts provide a variety of concepts, which include the infrastructure needed to capture the concept of TIPs and TDPs. There are three types of states in statecharts: OR-states, AND-states, and basic states [12]. AND-states are also called *orthogonal states* and relate their components as separate concurrent parts. Modelers can use orthogonal states to break

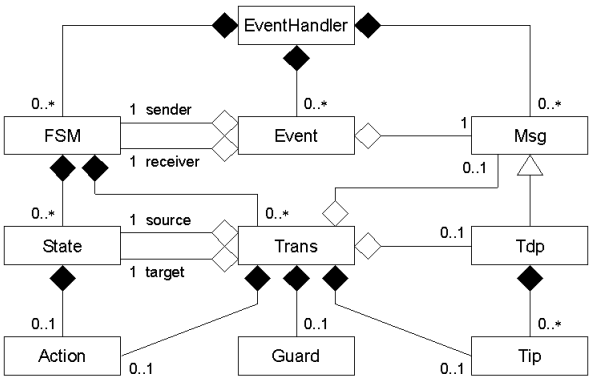

Figure 7. The design of the FSM prototype

down large state spaces naturally into independent (or almost independent) parts [11]. This supports structuring the TCP protocol into two statecharts, as was shown in figures 3 and 4. TDPs could be replaced by an event generation statement in the action list of the corresponding transition. TIPs, on the other hand, could be implemented by using trigger conjunction (event X *and* the TDP event have to be on the event stack) for passive TIPs, and parallel transitions (event X *or* TDP event) for active TDPs. Note that "in state" guards (which is a condition to a state reference) may offer a shortcut replacement for TIPs and TDPs in some cases. In general, the triggering TDP transition needs to be specified.

The situation is slightly different for another prominent modeling language, the UML. Statechart diagrams in the UML have experienced some restrictions. In its early version [22] there was no way of synchronizing concurrent regions. Meanwhile, so-called *synch states* have been introduced [23], which may be used as a substitute for TDPs and passive TIPs. However, due to the extension mechanisms available for the UML [14], there is the option to explicitly introduce TDPs and TIPs. In its simplest form, notifier events can be subclassed to the event metaclass using stereotypes (see section 2.12 of the UML [23] semantics), and TDPs and TIPs can be attached as properties to the transition metaclass. In addition to that, some changes to the execution semantics are required to process notifier events.

To conclude, the instruments needed to couple orthogonal regions of a state machine are available. The implementation of TDPs and TIPs may vary, but their main advantage is their notational clarity and expressiveness. Additional events in action lists, trigger conjunctions, parallel transitions, synch states etc. tend to clutter up diagrams and even hide the fact that coupling is an important aspect. The notation of TDPs and TIPs gives a clear understanding of where and how two or more separated FSMs strive for synchronization.

Note that the discussion above applies to white box specifications only. From a black box point of view there is no "superstate" that could serve as a container of orthogonal states. Instead, the capsule with its attached FSM interfaces (e.g. SAP, SAP^{-1}, CEP) provides the context of broadcast for notifier events. In this respect, TDPs and TIPs specify a broadcast protocol over a memoryless broadcasting channel for synchronization purposes, with the capsule defining the broadcasting scope. Such an interpretation of TDPs/TIPs requires different semantics, which go beyond the scope of state machine modeling.

5 Conclusions

Actually, the TDP/TIP concept relates very much to the observer pattern [9]; it allows the modeler to notify other FSMs about state changes. Because of the distinction in active and passive TIPs, the concept of coupled state machines implements an extended observer pattern. This lifts the observer pattern from its use in the design domain in form of class diagrams to the modeling domain with an explicit notation for coupling, which is a quite interesting aspect. Furthermore, it is an interesting thought, if TIPs and TDPs could be of use in sequence diagrams or Message Sequence Charts (MSC) [17].

Since the approach presented gives means to specify and separate aspects of a modeling entity, one could also investigate to which extend TDPs and TIPs enable aspect-oriented modeling in extension to aspect-oriented programming [19]. It also allows the modeler to specify APIs (Application Programming Interface) much more elegant; for instance, the TCP SAP could be seen as an API to TCP. In short, it is perceivable that many application areas could benefit from using coupled state machines.

Due to the specific nature of the application domain (data and telecommunications) we study, we cannot claim that we have identified all the types of TDPs and TIPs required for coupling FSMs in an efficient manner. Extensions or specialization are conceivable. However, TDPs and TIPs appear to be a powerful modeling concept, which puts a modeler in a better position especially for modeling distributed communication systems.

Acknowledgements: Many thanks to Andreas Witzel, who sparked off the idea of coupled state machines. Furthermore, the authors would like to thank Jörg Bruß and Dietmar Wenninger (all Ericsson) for their support.

References

[1] Customised Applications for Mobile network Enhanced Logic (CAMEL) Phase 3 – Stage 2. Technical Specification 3G TS 23.078 version 3.1.0, 3rd Generation Partnership Project, Valbonne, France, Aug. 1999.

[2] H. Balzert. *Lehrbuch der Software-Technik: Software Entwicklung*. Spektrum Akademischer Verlag, 1996.

[3] H. W. Barz. *Kommunikation und Computernetze: Konzepte, Protokolle und Standards*. Hanser, 1991.

[4] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[5] R. Braden. Requirements for Internet Hosts – Communication Layers. Standard RFC 1122, Internet Engineering Task Force, Oct. 1989.

[6] M. Broy. (Inter-)Action Refinement: The Easy Way. *Program Design Calculi, Series F: Computer and System Sciences*, 118, 1993.

[7] M. Broy. Compositional Refinement of Interactive Systems Modelled by Relations. In de Roever et al. [8], pages 130–149.

[8] W.-P. de Roever, H. Langmaack, and A. Pnueli, editors. *Compositionality: The Significant Difference; International Symposium, COMPOS'97, Bad Malente, Germany, September 8-12, 1997*, LNCS 1536. Springer, 1998.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[10] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, pages 231–274, July 1987.

[11] D. Harel and E. Gery. Executable Object Modeling with Statecharts. *IEEE Computer*, 30(7):31–42, July 1997.

[12] D. Harel and A. Naamad. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, Oct. 1996.

[13] D. Herzberg and A. Marburger. The Use of Layers and Planes for Architectural Design of Communication Systems. accepted for publication and presentation at the 4th International Symposium on Object-Oriented Real-Time Distributed Computing, ISORC 2001, Magdeburg, Germany; May, 2–4, 2001.

[14] D. Herzberg and L. von Wedel. Erweiterungsmechanismen der UML. *OBJEKTspektrum*, (4):56–59, Juli/August 1999.

[15] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[16] Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model. ITU-T Recommendation X.200, International Telecommunication Union, July 1994.

[17] Message Sequence Chart (MSC). ITU-T Recommendation Z.120, International Telecommunication Union, Nov. 1999.

[18] V. Jacobson, B. Braden, and D. Borman. TCP Extensions for High Performance. Standard RFC 1323, Internet Engineering Task Force, May 1992.

[19] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, LNCS 1241. Springer, June 1997.

[20] L. Lamport. Composition: A Way to Make Proofs Harder. In de Roever et al. [8], pages 402–423.

[21] M. Lutz. *Programming Python*. O'Reilly, 1996.

[22] Unified Modeling Language Version 1.1. Technical Specification, Object Management Group (OMG), Sept. 1997.

[23] Unified Modeling Language Specification, Version 1.4. Technical Specification, Object Management Group (OMG), Feb. 2001.

[24] Internet Protocol. Standard RFC 791, Internet Engineering Task Force, Sept. 1981.

[25] Transmission Control Protocol. Standard RFC 793, Internet Engineering Task Force, Sept. 1981.

[26] B. Selic, G. Gullekson, and P. T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley & Sons, Inc., 1994.

[27] A. S. Tanenbaum. *Computer Networks*. Prentice Hall PTR, Upper Saddle River, New Jersey 07458, 3rd edition, 1996.