State Machine Modeling: From Synch States to Synchronized 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@cs.rwth-aachen.de

Abstract: To synchronize concurrent regions of a state machine, the Unified Modeling Language (UML) provides the concept of so-called "synch states". Synch states insure that one region leaves a particular state or states before another region can enter a particular state or states. For some application areas, it is beneficial to synchronize not only regions but also state machines. For example, in data and telecommunications, a pure black box specification of communication interfaces via statechart diagrams gives no adequate means to describe their coordination and synchronization. To circumvent the limitations of the UML, this paper presents the concepts of Trigger Detection Points (TDP) and Trigger Initiation Points (TIP); it allows a modeler to couple state machines. The approach is generic, easy to extend and smoothly fits to the event model of the UML; it could also substitute the more specific concept of synch states.

1 Introduction

The problem of synchronizing concurrent state machines raised as an issue in a research project at Ericsson [9]. Concerned with architectural modeling of telecommunication systems, we developed a ROOM (Real-Time Object-Oriented Modeling) [20] like notation (see [8]) but were soon confronted with the question of coupling interfaces: How do we model the interaction between interfaces (or ports) of a single component without referring to its internals? The intention was to describe an architecture in a black box manner, though being capable to understand and simulate interface coordination and synchronization. In

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

other words, the question was how to properly couple the individual state machines, which specify the interface behavior of a single component.

Independent of this investigation, an Ericsson internal study on the use of modeling languages for service and protocol specifications exactly points out the same problem. It shows that the coupling problem is of theoretical as well as practical relevance. It is also one of the reasons, why modeling languages like the UML (Unified Modeling Language) [17] have not successfully penetrated the systems engineering domain, yet. System designers of data and telecommunication systems do not find reasonable support in today's modeling languages for their problem domain [7].

In the following two subsections, the telecommunication background is introduced and the problem is described in more detail. Subsequent sections discuss the proposed solution: In section 2 we elaborate on the model presented in subsection 1.1 in form of a case study. There, we study the TCP (Transmission Control Protocol) layer of a data communication system and show how the external interfaces can be described by a Finite State Machine (FSM) each. Section 3 discusses the coupling problem from different perspectives and demonstrates how FSMs can be synchronized via Trigger Detection Points (TDP) and Trigger Initiation Points (TIP). The implementation of a prototype, verifying the TDP/TIP concept, indicates how the UML could incorporate the TIPs and TDPs as extensions and supersede synch states; this is subject to section 4. Finally, section 5 closes with some observations and conclusions.

1.1 Background

On an architectural level, any data or telecommunication system can be structured according to two different directions of communication, "vertically" and "horizontally". "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). "Horizontal" communication, on the opposite, 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 "stackwise" abstracting the communication service. These stacks naturally give means to "vertically" divide a node into layers.

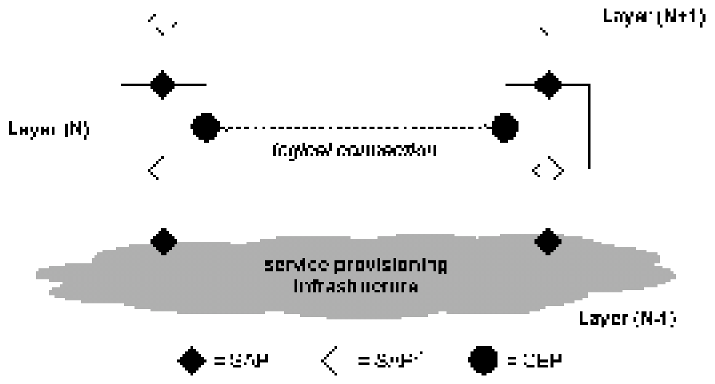

Figure 1: A simplified communication model based on OSI RM

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) [19] or the Internet Protocol (IP) [18]. 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 protocol 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 bases on the OSI (Open Systems Interconnection) Reference Model [12], which has laid a solid foundation for understanding distributed system intercommunication [3]. The notation used for the SAP and SAP^{-1} is an extension to ROOM; for a thorough discussion see [8].

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). For this discussion, we assume that Finite State Machines (FSM) according to the Unified Modeling Language (UML) [17] are the primary means to describe behavioral aspects. FSMs are a common tool for specifying protocols [11].

2 **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 that case, the CEP, the SAP, and the SAP^{-1} are specified by an FSM each, 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.

2 **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 clear.

What both problems have in common is that different views change scope and redefine how states or state machines are coupled with each other. In case of white box specifications, the UML offers the concept of composite states, which can be decomposed into two or more concurrent substates, also called *regions*. In order to enable synchronization and coordination of regions, the UML introduced *synch states*. However, synch states do not sufficiently support enough synchronization means as the case study presented below shows, nor do they solve the problem of synchronizing states of distinct state machines. Driven by a black box view, we propose the idea of Trigger Detection Points (TDP) to enable FSM separation but smooth coupling. TDPs together with Trigger Initiation Points (TIP) are introduced as a concept extending state machine modeling; they were motivated by the concept of detection points in [1].

2 Case Study: The TCP 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 [19] sums up to less than one hundred pages)[1], public available, and – most important – it is widespread and one of the most used protocols world-wide. Together with IP, the TCP/IP protocol suite forms the backbone of the Internet architecture.

Looking at how the TCP standard [19] 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 is the result of a white box view. 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 side, the client user

[1]Clarifications and bug fixes are detailed in [5], extensions are given in [14].

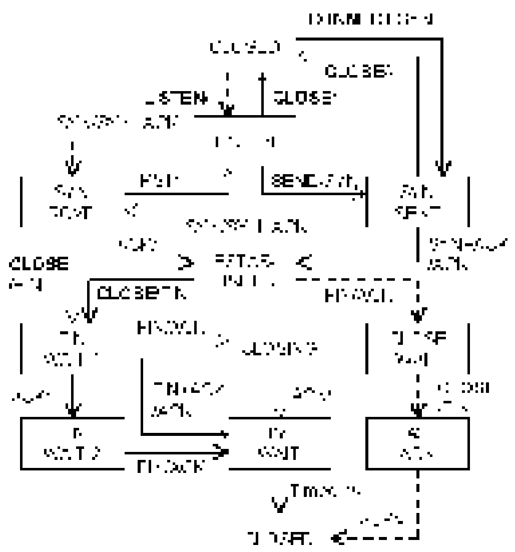

Figure 2: The TCP FSM figure is derived from [21, p.532]. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events. User commands are given in bold font.

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 completes successfully, both parties end up in state ESTABLISHED and are ready to send and receive data respectively. This short description of figure 2 neglects a lot of details of TCP (e.g. timeouts, which are important to resolve deadlocks and failures) but is sufficient for the purpose of our discussion. The interested reader may consult [21] for more information.

In order to structure TCP according to its interface functions (figure 1) the FSM in figure 2 needs to be partitioned. The result of this step is shown in figure 3 in UML notation. The left hand side of figure 3 displays the FSM, which corresponds in functionality to the SAP. Instead of using the TCP service commands LISTEN, CONNECT, SEND etc., the commands have been converted to service primitives, which are more narrative. Again, the client and the server side are combined in a single SAP FSM. From a user's viewpoint the communication with the client/server SAPs looks like follows: When a user requests a connection (Conn.req), the client's SAP changes to state C-PENDG. The server gets notified by 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

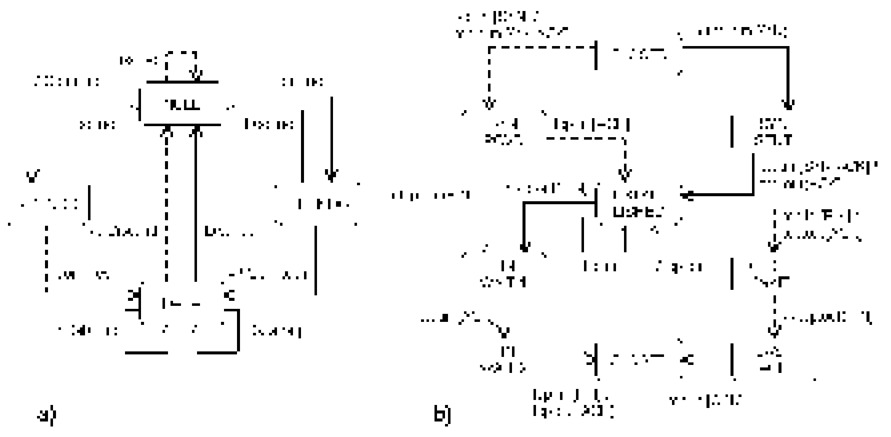

Figure 3: The FSMs of the SAP and the CEP of the TCP layer. The shortcuts stand for connect and disconnect; the postfixes stand for request, confirmation, indication, and response

SAP interface; the layer and its use of TCP is hidden.

The logical CEP holds the protocol specification of TCP, see the right hand side of figure 3. Since we have not introduced any coupling yet, the CEP FSM is strictly separated from the SAP FSM. That is why there is for example no indication what might have triggered the transition from CLOSED to SYN SENT at the client's side; but when the transition is triggered, no matter how it happened, then it sends out a TCP message with the SYN bit set. Otherwise, figure 3b is similar to figure 2; just all the numerous SAP related details have been stripped off. To 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 usually is IP and is accessed via the inverse TCP SAP^{-1}. To avoid cluttering up the discussion and distracting the reader with too many FSMs, we intentionally left out an example figure. For the sake of brevity, the SAP^{-1} is not considered and supposed not to exist; that is we assume the logical connection between the CEPs to be for real. Consequently, we can restrict the discussion on the interaction between the SAP and the CEP; this simplifies and eases the topic under discussion.

3 The Concept of Coupled State Machines

We managed to partition TCP according to its layer interfaces, which already is an achievement. All further details of TCP like flow control and buffering, congestion control, fragmentation, error control, window flow control etc. are hidden and subject of a refined view. As was mentioned above: If we prefer a white box view, the two state machines could be interpreted as concurrent regions in a “higher level” statechart and synchronized via synch states. If we, on the opposite, demand a rigid black box view (as is often the case for architectural modeling), the SAP and the CEP are described by two separate FSMs specifying the “horizontal” and “vertical” communication behavior; there are no coupling capabilities. 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 3a) needs to have an activity attached that sends a signal to the CEP (see figure 3b). 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. Obviously, 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 figure 4. 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 figure 4. *Passive* TIPs, on the other hand, have a locking mechanism and can be meaningfully used with “normal” transitions only, 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 it fire. Passive TIPs behave like a logical “and” to synchronize a transition, whereas active TIPs realize a logical “or”. An example of a passive TIP can be found in figure 4a; 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 4a and 4b. 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,

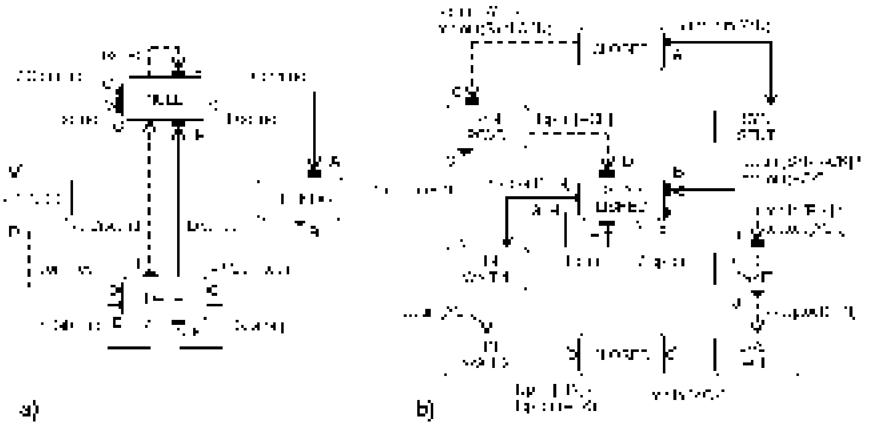

Figure 4: The FSMs of the SAP and the CEP of the TCP layer coupled via TDPs and TIPs

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 it 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 peacefully coexist without blurring the difference between both views.

4 Extending the UML

Synch states as known from the UML correspond in their behavior to what we called *passive* TIPs: A synch state is used in conjunction with forks and joins to insure that one

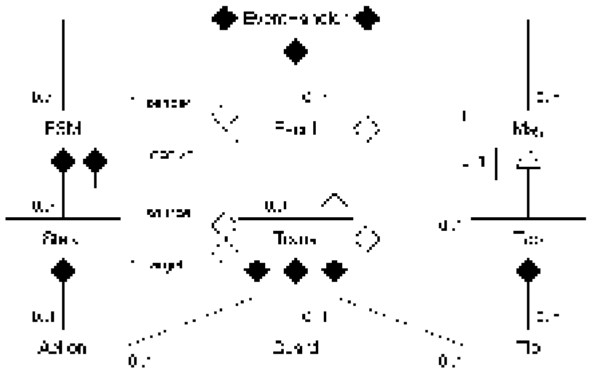

Figure 5: The design of the FSM prototype

region leaves a particular state or states before another region can enter a particular state or states [17]. Clearly, synch states do not support other synchronization means between regions like TIPs do and they are not suited for inter-FSM synchronization. Good reasons to think about integrating TIPs and TDPs in the UML and to substitute synch states.

TDPs and TIPs can be smoothly integrated in an event driven execution model for FSMs. The prototype we developed at Ericsson (programmed in Python [16]) treats TDPs as a specialization of messages, see figure 5, and dispatches notifier events to the event queue. The implementation of TIPs required only a few modifications to the event processor.

If one compares the prototype design and the metamodel for state machines (see section 2.12 of the UML [17] semantics), the required extensions to the UML can be easily identified: First, the notifier event needs to be subclassed to the event metaclass;[2] this can be achieved by using stereotypes. Then, it is to be decided how TDPs can be attached to the transition metaclass. Since transitions are restricted to have not more than one event trigger, it is not possible to add TDPs as a second trigger. Rather, the transition metaclass can be extended by some few properties. A TDP property is needed referring to the notifier event, optionally added by a property holding a list of state machines the notifier event is selectively broadcasted to. Another property are the TIP and the TIP type, which hold the notifier reference and the value active or passive, respectively. The required changes to the execution semantics of state machines are uncritical, since the UML is relatively open to adaptations. To conclude, the extensions described are the simplest form to introduce TDPs and TIPs to the UML using its extension mechanisms [10].

Note that TDPs and TIPs make synch states superfluous. TDPs/TIPs contain the concept of synch states but allow much more semantic variations and extensions. Synch states are an oddity in the UML with no clear conceptual roots; TDPs and TIPs are their generalization but they are put in a meaningful semantical context of transitions and events. In fact, TIPs and TDPs specify a synchronization protocol between states machines or regions. Such a protocol does not only seem more appropriate to capture complex interactions of

[2]Regarding events, the UML is a bit different designed than our message based prototype.

synchronization but also semantically cleaner. That is why we propose to remove synch states from the UML metamodel and instead introduce the notifier event subclass, insert metaclasses for TDPs and TIPs and associate them to the transition metaclass. This would enable flexible semantic extensions via stereotypes to the UML user.

5 Conclusions

Actually, the TDP/TIP concept relates very much to the observer pattern [6]; 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 question, if TIPs and TDPs could be of use in sequence diagrams or Message Sequence Charts (MSC) [13].

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 [15]. 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. As was shown in the case study, the design of communication protocols gains a lot of clarity from the separation of logical concerns. In short, it looks like 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 types of TDPs and TIPs required for coupling FSMs in an efficient manner. Extensions or specializations are conceivable. However, TDPs and TIPs appear to be a powerful modeling concept, they substitute synch states, and put a modeler in a better position especially for modeling the coordination and synchronization of concurrent systems.

Acknowledgements: Many thanks to Andreas Witzel, who triggered 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, August 1999.

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

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

[4] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

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

[6] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[7] Dominikus Herzberg. UML-RT as a Candidate for Modeling Embedded Real-Time Systems in the Telecommunication Domain. In Robert France and Bernhard Rumpe, editors, *≪UML≫'99 – The Unified Modeling Language: Beyond the Standard; Second International Conference, Fort Collins, CO, USA, October 28–30, 1999*, LNCS 1723, pages 330–338. Springer, 1999.

[8] Dominikus Herzberg and André Marburger. The Use of Layers and Planes for Architectural Design of Communication Systems. In *The Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2001), Magdeburg, Germany; May 2–4, 2001*, pages 235–242. IEEE Computer Society, May 2001.

[9] Dominikus Herzberg, André Marburger, and Tony Jokikyyny. E-CARES Research Project: Understanding Complex Legacy Telecommunication Systems. 2. Workshop Software-Reengineering, Bad Honnef, Germany, 11.-12. May, 2000.

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

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

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

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

[14] Van Jacobson, Bob Braden, and Dave Borman. TCP Extensions for High Performance. Standard RFC 1323, Internet Engineering Task Force, May 1992.

[15] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, LNCS 1241. Springer, June 1997.

[16] Mark Lutz. *Programming Python*. O'Reilly, 1996.

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

[18] Internet Protocol. Standard RFC 791, Internet Engineering Task Force, September 1981.

[19] Transmission Control Protocol. Standard RFC 793, Internet Engineering Task Force, September 1981.

[20] Bran Selic, Garth Gullekson, and Paul T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley & Sons, Inc., 1994.

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