Combining SDL and Pattern-based Design for the Customization of Communication Subsystems [1]

Birgit Geppert and Frank Rößler
Computer Science Department,
University of Kaiserslautern
P.O. Box 3049, D-67653 Kaiserslautern, Germany
{geppert, roessler}@informatik.uni-kl.de

Abstract

Due to the large variety of modern applications and evolving network technologies, a small number of general-purpose protocol stacks will not always be sufficient. Rather, customization of communication protocols will play a major role. In this paper, we present an approach that has the potential to substantially reduce the effort for designing customized protocols. Our approach is based on the concept of *design patterns*, which is well-established in object-oriented software development. We specialize this concept to communication protocols, and - in addition - use formal description techniques (FDTs). Thus patterns are supplemented with precise rules for their instantiation and composition as well as semantic properties to specify their intent more precisely. The FDTs of our choice are SDL and MSC. We propose an SDL pattern description template and discuss the requirements concerning language support. A pattern-based configuration process comprising *selection, adaptation* and *composition* of SDL patterns is presented.

Keywords: SDL, design patterns, communication protocols, protocol engineering, reuse

1 Introduction

Today's communication systems are typically structured into several layers, where each layer realizes a defined set of protocol functionalities. These functionalities have been carefully chosen such that a wide range of applications can be supported and protocols work in a general environment of networks. Well-known architectures of this type are the OSI stack or the Internet protocol suite. However, due to evolving network technologies as well as increasing demands of modern applications general-purpose protocol stacks are not always adequate. In particular, varying demands on throughput and delay as well as on delay jitter, synchronization or multicasting are not well supported yet. Also, traditional communication subsystems are not designed to exploit the advantages of advanced transmission technologies (e.g., fiber optics) and high-speed networks (e.g., ATM), which combine high bandwidth with low error rates. Rather, traditional communication subsystems enforce the use of mechanisms that may actually not be needed by a given application (e.g., error control), which leads to reduced performance.

To achieve an improvement plenty of different customized communication subsystems would be necessary. Here, protocol configuring is a promising way to cope with the additional development effort. Actually, we suggest to provide a pool of reusable proto-

1. This work has been supported by the German Science Foundation (DFG) as part of the Sonderforschungsbereich 501 "Development of Large Systems with Generic Methods".

Appeared in:
A. Wolisz, I. Schieferdecker, A. Rennoch (Ed.): Formale Beschreibungstechniken für verteilte Systeme, GI/ITG-Fachgespräch, 19.-20. Juni 1997, Berlin (GMD-Studien Nr. 315) ISBN 3-88457-315-2

col building blocks from which the protocol designer may select components according to the specific communication requirements. After suitable adaptation these building blocks are ready for composition to build part of the customized communication subsystem. To assure high quality of the resulting communication subsystem we apply formal description techniques (FDTs) during all development stages. For protocol design the FDT of our choice is SDL (see [7] for further information on requirements engineering). Before going ahead two important assumptions have to be considered:

- Communication subsystems are decomposable into protocol building blocks.
  Several approaches dealing with the automatic configuring of protocol implementations (e.g., F-CCS [14], Da CaPo [8], CogPiT [15], ADAPTIVE [11], see [5] for an overview) give strong evidence for the existence of protocol building blocks. Though it seems obvious that communication protocols generally have a number of protocol functionalities in common, this research work was first to identify and employ actual protocol components by reverse engineering of many existing transport and network protocols. Note that contrary to these approaches we do not aim to configure subsystem implementations but formal subsystem design specifications in SDL. We therefore expect to learn more about inherent composition principles and to give a formal basis for subsystem validation.
- Design Patterns offer adequate flexibility to represent *reusable* protocol building blocks.
  The reuse of predesigned solutions for recurring design problems is of major concern in object-oriented software development. During the past few years *design patterns* have emerged as an especially fruitful concept from other well known approaches such as frameworks or toolkits (in the sense of object oriented libraries) [3]. It seems that SDL adapted patterns form a proper basis for the representation of reusable protocol building blocks [4]. In [13] early experience in reuse of protocol specifications with SDL was reported. Thereby an example protocol building block was designed as a reusable library class, however, according to the authors with limited success. As their conclusions were similar to those, which led from the toolkit approach to the design patterns concept, we argue that SDL patterns are better suited. In [6] we configure a reservation protocol as part of a real-time communication architecture, which successfully employs the same building block but represented as an SDL pattern.

The remainder of the paper is organized as follows: we first introduce the main concepts of pattern-based design (Section 2) and subsequently discuss the requirements concerning language support (Section 3). The combined use of SDL with pattern-based design is presented in Section 4, where we actually propose an advanced SDL pattern description template, illustrate an example SDL pattern, and describe the process of pattern application more precisely. Current experience with our configuration approach and concluding remarks are given in Section 5.

2 The patterns concept for reuse of architecture and design

During the past few years design patterns have emerged as an especially fruitful concept for high-level reuse of architecture and design in object-oriented software development:

- *"Design Patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context." [3]*

- *"A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the ways in which they collaborate." [2]*

Design patterns capture solutions, which have evolved over time and serve as an elegant way to make designs more flexible, modular, reusable, and understandable. They reflect experience gained during prior developments and therefore help designers to reuse successful designs and architectures. As a consequence, the design process becomes faster and the number of design errors decreases. Design patterns even help improving the documentation and maintenance of existing software systems, because they focus on the essentials of a design solution and thus increase the probability to be matched with a given design. The patterns presented in [2] and [3] are described by the means of special description templates. Roughly speaking, a pattern is mainly described by its name, the recurring design problem it addresses, a corresponding well-proven design solution, and a short description when the pattern should/should not be applied. A main characteristic of conventional design patterns is their independence on a particular target language. Thus the design solution normally is described in natural language combined with an OMT object model that defines the participating components and their relations. Though sample implementations may be given, these are not strictly prescribed and only serve as implementation guidelines. The lack of a formal target language for pattern instantiation prevents the patterns semantics from being precisely specified.

A design pattern determines relevant components of an embedding context and describes how they should be adapted to solve the addressed design problem. This is expressed in terms of reuse mechanisms such as class inheritance and parameterized types [3]. Besides a formal foundation, a target language for pattern instantiation should therefore support structural concepts to define the pattern's constituent components as well as object-oriented reuse mechanisms.

Each pattern, once applied, is part of the embedding context into which other pattern instances may be integrated to progress further to the final system design. Thus care has to be taken to prevent a patterns intent from being destroyed unintentionally by the application of further patterns. This also suggests the use of a language with formal semantics for pattern-based design.

3 SDL language support for pattern-based design

A main goal of our approach is to capture protocol design patterns formally, which involves two major issues. Firstly, techniques are needed to specify context information for adequate pattern application. This includes, for instance, structural information about participating components, external interactions, pattern's intent, composition intelligence. We will propose an initial pattern description template to capture this kind of context information in Section 4.

Secondly, a formal target language for pattern instantiation is needed, which is mandatory for the specification of context information mentioned above. In the following, we will elaborate on this issue, where the FDT of our choice is SDL.

SDL is a formal description technique for describing reactive, discrete systems. The most recent version was standardized by the International Telecommunication Union (ITU) in 1992 [17] with some updates made in 1996. It offers a textual as well as a graphical notation. SDL is an object-oriented language suitable for description of architecture, behaviour, and data of distributed, communicating systems.

As mentioned in Section 2, a target language for pattern instantiation should support structural concepts in order to define the patterns participating components. The language elements of SDL concerning system architecture, namely blocks, processes, and services, turned out to be well-suited for pattern specification. An SDL system specification is structured into blocks where each block is composed either of a set of blocks or a set of processes. System behaviour is modelled as a set of communicating extended finite state machines (CEFSMs) each represented by an SDL process with an unbounded FIFO input queue. SDL processes run concurrently and communicate asynchronously by signal exchange. An SDL process may be further structured in SDL services, which themselves represent CEFSMs. In this case, an SDL process appears as the product automaton of its services. The services of a process run sequentially and share the input queue of the process. We found out that the design solution represented by a protocol design pattern may be independent on the architecture of the embedding context and interaction paradigm. Thus we propose to specify several versions of the same pattern (realization as SDL process, service or procedure, interaction by message passing or shared variables, etc.) when appropriate. This kind of versioning enables the design of monolithic or rather distributed systems. Concerning block structure we have additionally devised a small number of transformation rules stating how to modify the system structure without affecting external behaviour.

It is worth mentioning that SDL also enables the specification of abstract data types - i.e. passive objects - called *sorts,* which is valuable to specify pattern components. The behaviour definition of the operators is given informally, axiomatically, algorithmically or by the use of another notation such as ASN.1.

Apart from the various structuring concepts other important properties of SDL for pattern-based protocol engineering are its object-oriented features. As an object-oriented language SDL supports the type definition of systems, blocks, processes, and services. It also supports the reuse of type definitions by parameterization and specialization. In case of a *parameterized* type, formal context parameters can be set to actual context parameters by instantiation. Example parameters are signals or variables. *Specialization of types* results from the reuse of an existing type definition by bounding some context parameters, adding properties or redefining virtual types or transitions. Table 1 gives an overview of possible specializations for different SDL types. As mentioned in Section 2, a design pattern determines relevant components of an embedding context. It is now possible to describe in terms of SDL reuse mechanisms how they should be adapted to solve the addressed design problem (see Section 4 for the composition of SDL patterns).

For systems engineering SDL is often combined with other languages such as OMT object model [10] or MSC [18]. Though the combined use of OMT and SDL is not yet standardized it is commonly agreed on that "the use of the OMT object model notation in combination with MSC and SDL is a powerful combination that covers all aspects of system engineering"[1]. Actually, several SDL design methodologies exist, which combine OMT and/or MSC for analysis and SDL for system design [1] [9] [12] [16]. The integration of SDL with these techniques is also supported by existing development tools (see, e.g., [12]). As we use a combination of MSC, OMT object model, and SDL for patterns specification (Section 4.1) this is another advantage, which favours SDL for formal pattern-based engineering.

1. borrowed from the Homepage of the SDL-Forum, which can be found at *http://www.sdl-forum.org/*

	Specialization by adding or redefining the following:
system type	block type/instance, process type, service type, procedure, gate, signal, channel, sort
block type (composed of blocks)	block type/instance, process type, service type, procedure, gate, channel, signal, sort
block type (composed of processes)	process type/instance, service type, procedure, gate, signal route, signal, sort
process type (composed of services)	service type/instance, procedure, variable, timer, signal route, signal, gate, sort
service type, process type (behaviour definition)	transition, procedure, variable, timer, signal route, signal, gate, sort
procedure	transition, signal, timer, parameter, procedure
signal	signal parameter
sort	literal, operator

Table 1: possible type specializations in SDL

4 Pattern-based protocol configuring with SDL

As already mentioned, we aim to produce an SDL specification by reusing predesigned protocol building blocks, where we do not employ low-level reuse of code but high-level reuse of architecture and design. We therefore combine the design patterns concept with SDL for definition of reusable protocol building blocks.

4.1 SDL patterns as reusable protocol building blocks

An SDL pattern describes a generic solution for a recurring context-specific design problem from the domain of communication protocols. Furthermore, it is assumed that SDL is the target language for pattern instantiation. This definition of SDL patterns is similar to those of conventional design patterns (Section 2). But contrary to conventional design patterns, SDL patterns are dedicated to a specific application domain, namely communication systems, and combine the advantages of the formal description technique SDL with the patterns‘ concept. Instead of specifying and applying the patterns rather informally, SDL offers the possibility to specify what the application of a specific pattern precisely means, under which assumptions this will be allowed, and what the consequences are. For the specification of SDL patterns we propose an enhanced description template with the items listed below. A comparison to existing description templates of conventional design patterns is given in [6].

Name The name of the pattern, which should intuitively describe its purpose.

Intent A short informal description of the particular design problem and its solution.

Motivation An example from the area of communication systems, where the design problem arises. This is appropriate for illustrating the relevance and need of the pattern.

Structure A graphical representation of the structural aspects of the design solution using an *OMT object model*. This defines the involved components and their relations.

Message scenario Typical scenarios describing the interactions between the involved objects (e.g. protocol entity, service user) are specified by using *MSC diagrams*.

SDL-fragment The mere syntactical part of the design solution is defined by a generic *SDL-fragment*, which is adapted and syntactically embedded when applying the pattern. If more than one SDL **versions** of the design solution are possible (realization as SDL service or procedure, interaction by message passing or shared variables, etc.), fragments for the most frequent versions are included. We plan to substitute versioning by a special kind of pattern parameterization.
For each fragment, corresponding **syntactical embedding** rules are defined:

- Rules for <u>renaming</u> of the abstract identifiers of the SDL-fragment.
- Rules for <u>specialization</u> of embedding SDL superclasses in order to integrate the instantiated pattern. This could, for instance, result in
    - the *addition* of new transitions or SDL services
    - the *redefinition* of existing virtual types or transitions.

Semantic properties Properties of the resulting specification that are introduced by the embedded pattern. This also includes a description of assumptions under which the properties hold. The semantic properties define the patterns intent more precisely.

Redefinition An embedded pattern instance can be further redefined, e.g., by the embedding of another pattern instance in subsequent development steps. Redefinitions compatible with the patterns intent and semantic properties are specified.

Cooperative usage Possible usage with other patterns of the pool is described. This is feasible and especially useful for a specific application domain as in our case.

The mere syntactical part of the design solution is defined by a generic *SDL-fragment*, which must be adapted and subsequently composed with a given embedding SDL specification. Possible specializations are constrained by *syntactical embedding* rules. An example for such a constraint would be that a redefined transition only adds a procedure call to the original (virtual) transition and keeps the same otherwise. A main advantage of using an FDT for the specification of patterns is that *semantic properties* can be formally specified defining the patterns intent more precisely. This includes desired properties and necessary assumptions, which have to be satisfied to ensure the intended use of the pattern. This is important for validation of the resulting communication protocol. Because an SDL pattern is part of a predesigned pool of protocol building blocks (each represented by an SDL pattern) it can be related to other pool components by describing

Name: TimerControlledRepeat

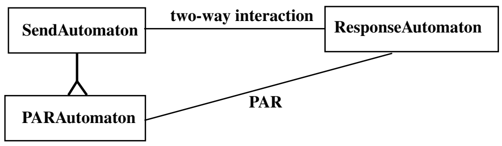
Intent:

TimerControlledRepeat extends a two-way interaction between two automata *SendAutomaton* and *ResponseAutomaton* for the case of possible message losses during data transfer. If an expected response does not arrive before the expiry of a timer, the message is repeated (Positive Acknowledgement with Retransmission). This pattern does not deal with the problem of message disruption or duplication.

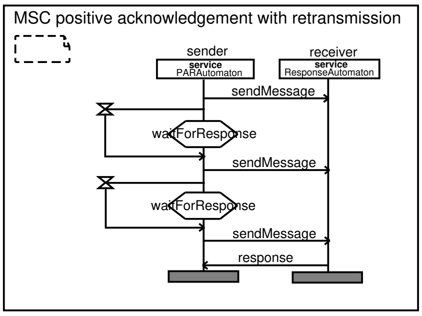
Motivation:

For a BlockingRequestReply pattern [4] instance the requester will deadlock, if the reliable transmission of the request or reply signal is not guaranteed. Therefore replies are observed by TimerControlledRepeat in case of an unreliable basic service.

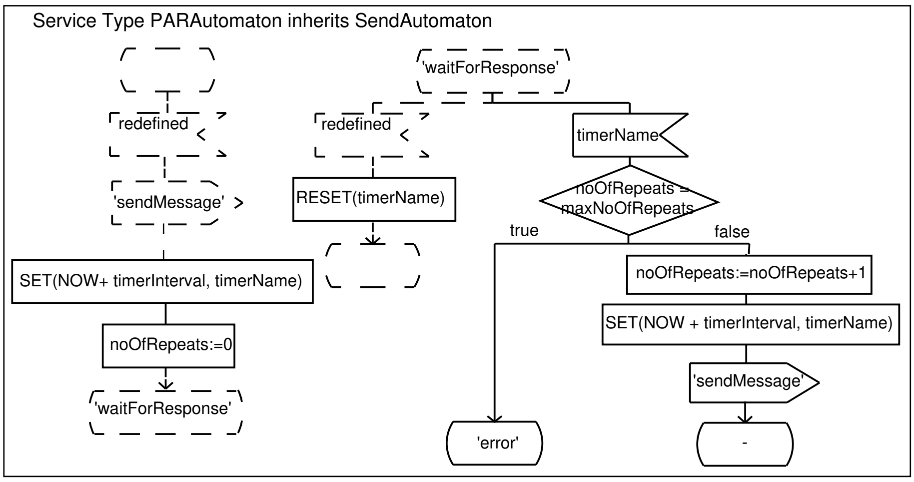
Structure:



Message Scenario:



SDL-fragment:



Semantic properties:

Property C.1.: *If the assumptions stated below hold, PARAutomaton will eventually receive a response from ResponseAutomaton after sending a sendMessage, or PARAutomaton will enter the error state after maxNoOfRepeats unsuccessful retransmissions. The assumptions are:*

- *The communication channel between* PARAutomaton *and* ResponseAutomaton *for transmission of* sendMessage *and corresponding response signals neither disrupts nor creates messages.*
- *The communication channel may lose messages but* timerInterval *is greater than the maximum round trip time of* sendMessage *and corresponding response.*
- ResponseAutomaton *merely discards duplicate* sendMessages *or reacts on duplicates the same way (from the perspective of* PARAutomaton*) as on the original* sendMessage.

Fig. 1: an example SDL pattern (incomplete description)

their *cooperative usage*. This is strongly supported by restricting to design problems of a certain domain. The item *redefinition* of our description template takes into account that actual pattern instances may be the embedding context for further pattern applications. Possible redefinitions of the already existing pattern instance are restricted such that its intent is not destroyed by the application of further patterns.

SDL patterns deal, for instance, with two-way interaction between communicating peers, error control (lost or duplicated messages), lower layer interfacing, or dynamic establishment and closing of connections. For illustration purposes an example SDL pattern *TimerControlledRepeat* is shown in Fig. 1.

4.2 Configuration process

Pattern-based configuring of an SDL protocol specification involves three steps: *selection, adaptation*, and *composition* (Fig.2).

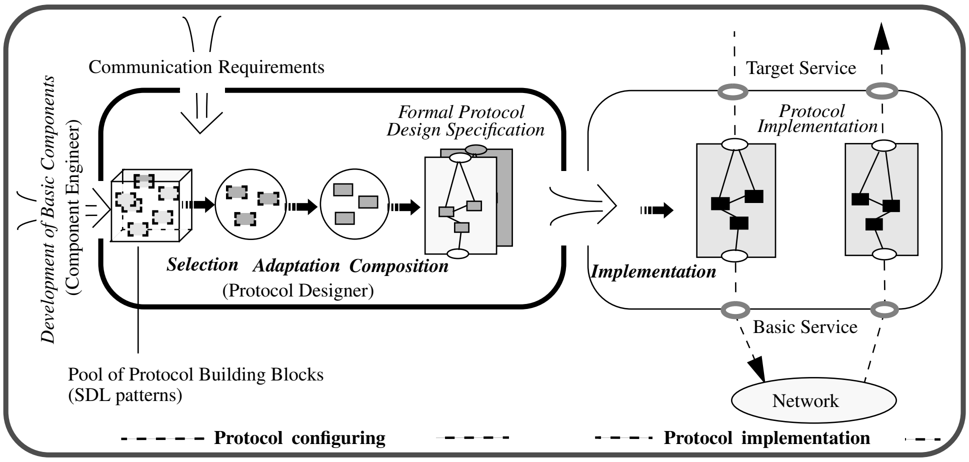


Fig. 2: pattern-based protocol configuring

- **selection:**
  as a prerequisite the communication requirements have to be analysed to identify the inherent design problems. Pattern selection can then be guided by comparing the resulting analysis model with the patterns descriptions. We assume that an object-oriented analysis is performed resulting in an OMT object model and MSC diagrams describing typical and exceptional interaction scenarios. Actually, this assumption is satisfied by existing SDL methodologies such as [1], [9], [12], or [16] into which pattern-based configuring can be seamlessly integrated. As we assume identical techniques (OMT, MSC) for analysis and pattern description we reduce the semantic gap between analysis and design models. By comparing OMT and MSC diagrams with the *structure* and *message scenario* descriptions of the SDL patterns and by further examination of the patterns' *intent*, *semantic properties,* and *motivation* protocol building blocks are to be selected.

- **adaptation**:
  as SDL patterns describe generic design solutions they have to be adapted before composition. Depending on the embedding SDL specification a suitable *version* of the pattern has to be selected. The chosen version additionally must be adapted by *renaming* the abstract identifiers in order to seamlessly fit the embedding SDL specification.

This is guided by the *syntactical embedding* rules. The result is a pattern instance ready for composition with the embedding SDL specification.

- **composition**:
  the pattern instance, i.e. the adapted SDL-fragment, finally has to be composed with the embedding SDL specification. This is done according to the *specialization* part of the *syntactical embedding* rules resulting in a specialization of the embedding SDL specification. Thereby, possible redefinitions of virtual constructs are constrained by the *syntactical embedding* rules.

The resulting SDL specification may be further refined in order to get an executable version. Therefore additional *redefinition* steps as far as allowed by the embedded patterns may be necessary. Examples are the declaration of new signals, sorts or channels. The *semantic properties* of an embedded pattern may also impose additional assumptions on the environment. They have to be taken into account in further development steps and must therefore be added to a list of assumptions [6].

5 Conclusions

We have presented an approach that has the potential to substantially reduce the effort for designing customized protocols. Our approach combines the advantages of formal description techniques with pattern-based design. We have discussed the use of SDL as a target language for pattern instantiation and proposed an advanced description template for specifying SDL patterns. To fill in this template OMT object model, MSC, and SDL as well as natural language are employed. Additionally, we have illustrated an example SDL pattern and sketched pattern application in the context of configuring communication protocols.

So far, we have applied our approach for re-engineering of modern protocols of the Internet protocol suite such as ST2+ or IPv6 and have also conducted a more complex case study, which considers the engineering of a real-time communication platform for distributed control of an inverted pendulum. The system comprises a conventional Token Ring network (5 Pentium-PCs running under QNX and connected with IBM 16/4 Token Ring Network Adapter II), which we have upgraded to offer a real-time communication service. For this purpose we have developed a real-time Token Ring driver including mechanisms for traffic policing, packet scheduling, connection admission control, and user interfacing as well as a resource reservation protocol. Roughly speaking, this protocol supports connection setup in conjunction with the reservation of sufficient network resources to guarantee a specified quality of service during data transfer. Its service is offered to a feedback control protocol tailored to the needs of the distributed application (controlling of the inverted pendulum). The task of this application-oriented protocol is to periodically distribute the sensor and control values. Though this case study was originally started by using “conventional” engineering methods, we applied pattern-based configuring by stepwise re-engineering specific parts. Compared to the SDL specifications developed the usual way, the specifications of the configured protocols are more readable, which is due to the more systematical design. Among other things, this results in improved maintainability and less design errors, since the design decisions are well founded and documented. Furthermore, the case study yielded a promising way for incremental design of communication subsystems [6].

Note that identification, investigation, and description of suitable protocol building blocks is a challenging and time consuming task, but provides payoff for subsequent pattern-based protocol engineering.

References

[1] R. Bræk and Ø. Haugen, *Engineering Real Time Systems - An object-oriented methodology using SDL*, Prentice Hall, 1993

[2] F. Buschmann et al., *Pattern-Oriented Software Architecture - A System of Patterns*, John Wiley & Sons, 1996

[3] E. Gamma et al., *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995

[4] B. Geppert, R. Gotzhein, and F. Rößler, *Configuring Communication Protocols Using SDL Patterns*, accepted for the 8th SDL Forum, Paris, France, September 1997

[5] B. Geppert and F. Rößler, *Automatic Configuration of Communication Subsystems - A Survey*, SFB 501 Report 17/96, University of Kaiserslautern, Germany, 1996

[6] B. Geppert and F. Rößler, *Pattern-based Configuring of a Customized Resource Reservation Protocol with SDL*, SFB 501 Report 19/96, University of Kaiserslautern, Germany, 1996

[7] R. Gotzhein, B. Geppert, C. Peper, and F. Rößler, *Generic Layout of Communication Subsystems - A Case Study*, SFB 501 Report 14/96, University of Kaiserslautern, Germany, 1996

[8] T. Plagemann, B. Plattner, M. Vogt, and T. Walter, *Modules as Building Blocks for Protocol Configuration*, Proceedings of ICNP`93, International Conference on Network Protocols, San Francisco, 1993

[9] R. Reed, *Methodology for Real Time Systems*, Computer Networks and ISDN Systems 28 (1996)

[10] J. Rumbaugh et al., *Object-Oriented Modeling and Design*, Prentice Hall, 1991

[11] D.C. Schmidt, D.F. Box, and T. Suda, *ADAPTIVE: A Dynamically Assembled Protocol Transformation, Integration, and eValuation Environment*, Concurrency Practice and Experience, Vol. 5, No. 4, 1993

[12] SDT 3.1 Methodology Guidelines: The SOMT Method, Telelogic, 1996

[13] A. Sinton and M. Crowther, *SDL-92 support for re-use in protocol system specifications - some early experience*, Proceedings of the 7th SDL Forum, 1995

[14] B. Stiller, *Flexible Protokollkonfiguration zur Unterstützung eines diensteintegrierenden Kommunikationssubsystems*, PhD thesis (in German), VDI-Verlag, Reihe 10, Nr. 306, 1994

[15] B. Stiller, *CogPiT - Configuration of Protocols in TIP*, Technical Report TR 368, University of Cambridge, Computer Laboratory, England, U.K., June 1995

[16] D. Witaszek et al., *A Development Method for SDL-92 Specifications Based on OMT*, Proc. 7th SDL Forum, 1995

[17] Z.100 (03/93) *CCITT Specification and Description Language (SDL)*, ITU-T, 1994

[18] Z.120 (10/96) *Message Sequence Chart (MSC)*, ITU-T, 1996