Towards an Experimental Evaluation of SDL-Pattern based Protocol Design

Raimund L. Feldmann, Birgit Geppert, Frank Rößler

SFB 501 Bericht 04/98

Towards an Experimental Evaluation of SDL-Pattern based Protocol Design

Raimund L. Feldmann[*], Birgit Geppert[+], Frank Rößler[+]
{feldmann, geppert, roessler} @informatik.uni-kl.de

Sonderforschungbereich 501
Technical Report 04/1998

[*]*Software Engineering Group*
[+]*Computer Networks Group*

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern
Germany

ABSTRACT

In order to improve the quality of software systems and to set up a more effective process for their development, many attempts have been made in the field of software engineering. Reuse of existing knowledge is seen as a promising way to solve the outstanding problems in this field. In previous work we have integrated the design pattern concept with the formal design language SDL, resulting in a certain kind of pattern formalization. For the domain of communication systems we have also developed a pool of SDL patterns with an accompanying process model for pattern application. In this paper we present an extension that combines the SDL pattern approach with the experience base concept. This extension supports a systematic method for empirical evaluation and continuous improvement of the SDL pattern approach. Thereby the experience base serves as a repository necessary for effective reuse of the captured knowledge. A comprehensive usage scenario is described which shows the advantages of the combined approach. To demonstrate its feasibility, first results of a research case study are given.

Keywords

experimental software engineering, experience base, design patterns, formal description techniques, communication protocols, SDL

Table of Contents

1 INTRODUCTION 1
2 THE CONSTRUCTION SET OF PROTOCOL BUILDING BLOCKS 2
2.1 The Formal Description Technique SDL 2
2.2 SDL Patterns 3
2.3 Configuration Process 5
3 THE EXPERIENCE BASE: SFB-EB 7
3.1 Logical structure of the SFB-EB 7
3.2 A usage scenario 10
4 FIRST EXPERIENCES: A CASE STUDY 13
5 CONCLUSION AND OUTLOOK 15

Appendix

A GQM PLAN ERRORS AND FAULTS 17
B AN EXAMPLE QUESTIONNAIRE FROM THE RTP CASE STUDY 23
REFERENCES 27

List of Figures

Fig. 1: Configuration process model (partial) 6
Fig. 2: Logical Structure of the SFB-EB regarding SDL pattern oriented development of communication protocols 8
Fig. 3: Excerpts from the GQM plan “errors and faults” 14

1 INTRODUCTION

Due to the increasing variety of modern applications and evolving network technologies, the communication services provided by today's general-purpose protocol stacks are not always adequate. It is expected that in order to increase flexibility and to support applications in the best possible way, customization of special-purpose protocols will also play a major role. Here, configuring communication protocols from reusable protocol building blocks seems to be a promising way for overcoming the additional development effort.

The reuse of predesigned solutions for recurring design problems is of major concern in object-oriented software development in general. During the past few years, design patterns have emerged as a particularly fruitful approach to software reuse [8] [14]. Contrary to the traditional paradigm of class and function libraries, which are solely concerned with code reuse, design patterns aim to focus on the invariant parts of a design solution and offer by far more flexibility for adaptation to the embedding context. That is, the potential of reuse is substantially increased. Additionally, design patterns proved helpful in guiding the instantiation and documentation of frameworks, which provide larger-scale reuse of the overall architecture and design from a certain application domain.

In [15] [17] [18] we present the SDL-pattern approach, which on principle adopts the pattern concept for the design of communication protocols. However, in order to assure high quality of the resulting communication subsystem, we apply a formal description technique (FDT) as design language. The FDT of our choice is SDL [20]. Thereby we benefit from the formal basis provided by SDL, so that tool support and validation of pattern application is possible. For instance, instantiation of a pattern can be defined by precise embedding rules in terms of the SDL syntax. Similarly, the semantic model of SDL allows to precisely state assumptions for adequate pattern application as well as resulting properties on the embedding context. This is a major improvement compared to conventional design patterns, which mainly rely on natural language based pattern description and, to a large degree, must still leave pattern application to the personal skills of the system designer. Note, however, that the enhancements work within the scope of an FDT with its formal syntax and semantics.

SDL patterns are expected to offer the same advantages as those commonly attributed to conventional design patterns: 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 experiences gained in prior developments and therefore help designers reuse successful designs and architectures. As a consequence, the design process becomes faster and the number of design errors decreases. Patterns even help improve the documentation and maintenance of existing software systems, because they focus on the essentials of a design solution and thus increase the probability of being matched with a given design. However, such statements often seem to be subjective in nature. That is, they are characterized as hypotheses that have to be validated.

In order to validate these hypotheses, we choose experimentation as known from the field of experimental software engineering, i.e., case studies (as defined in [22]) and controlled experiments (also denoted as ‘formal experiments’ [22]). Our experiments are planned, executed, and analyzed according to the Quality Improvement Approach (QIP) [4], supported by goal-oriented measurement based on the GQM paradigm [3]. The collected quantitative data serves as a basis for the improvement of the construction set of protocol building blocks. All gained experiences (i.e., all kinds of knowledge, such as measurement data, (process-) models, SDL-patterns, or lessons learned) are stored in a central repository: the Experience Base of the SFB 501 (SFB-EB) [12]. With the help of the SFB-EB we transfer the gained experiences into new projects and experiments in order to allow continuous improvement of the SDL-pattern approach.

The remainder of the paper is organized as follows: Section 3 shortly introduces the concepts underlying the proposed construction set of protocol building blocks, including a corresponding process model. In Section 3 the structure of the SFB-EB, as far as relevant to our approach, is described. Furthermore, a comprehensive usage scenario is given to show how this structure is used to support our approach. First experiences, gained in a research case study, demonstrate the feasibility of the combination of SDL-pattern based design and the experience base concept (Section 4). We summarize the results and conclude with an outlook in Section 5.

2 THE CONSTRUCTION SET OF PROTOCOL BUILDING BLOCKS

This section summarizes concepts for a construction set of protocol building blocks from which a protocol designer can select components and compose them into a customized, formal protocol specification. In order to get highly flexible building blocks and to increase the quality of resulting products, we combine the design patterns approach with SDL. For further details, the reader is referred to [15] [17] [18].

2.1 The Formal Description Technique SDL

SDL is a description technique with a formal syntax and semantics and is designed for the specification of reactive, distributed systems, in particular, communicating systems. The most recent version was standardized by the International Telecommunication Union (ITU) in 1992 [15] with some updates made in 1996. It offers a textual as well as a graphical representation. SDL is an object-oriented language capable of describing architecture, behavior, and data. It is in widespread use in industry and is well-supported by commercial and public domain development environments such as SDT [26], Object-GEODE [1], Cinderella, or SITE [27].

An SDL system specification is hierarchically structured into blocks. Each block is composed of either a set of blocks or a set of processes. System behavior is modeled as a set of communicating extended finite state machines (CEFSMs), each represented by an SDL process. SDL processes run concurrently and communicate asynchro-

nously 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. For the definition of data, SDL offers a number of predefined data types such as integer or boolean. New data types are either derived by using built-in constructs (e.g., structs or arrays) or by using the abstract data type concept. For the latter the behavior of the operators can be defined axiomatically or algorithmically as well as by interfacing to another language such as C.

As an object-oriented language, SDL allows the parameterized type definition of blocks, processes, services, and signals as well as their specialization by bounding parameters, adding properties (e.g., new input signals), or redefining virtual types or transitions. OO concepts not supported by SDL are multiple inheritance and dynamic binding.

One main advantage of SDL is that an SDL specification is already executable and can be used for simulation, validation against test cases, and validation of general properties (such as freedom from deadlock or implicit consumption). The possibility of simulating or formally analyzing a distributed system (especially its flow of control) before implementation is of great importance in order to detect design errors in early stages of development. Note that even code can be automatically derived from an SDL specification. Validation and simulation, as well as automatic code generation (rapid prototyping) are well supported by existing SDL development tools.

2.2 SDL Patterns

An SDL pattern describes a generic solution for a recurring context-specific design problem from the domain of communication protocols. It is assumed that the target language for pattern instantiation is SDL.

Contrary to conventional design patterns, SDL patterns add the advantages of a mathematical foundation. Instead of specifying and applying the patterns rather informally, a formal target language such as SDL offers the possibility of precisely specifying how the application of a specific pattern is performed, under which assumptions this will be allowed, and what properties result for the embedding context. This information is pattern specific and organized by means of a certain pattern description template. The main items of the SDL pattern description template are sketched in the following.

The mere syntactical part of the design solution is defined by a generic *SDL-fragment*, which has to be instantiated and textually embedded into the context specification when applying the pattern. SDL-fragments represent context invariant parts of a design solution. Instantiation and embedding of SDL-fragments is prescribed in terms of *syntactical embedding rules*, which, e.g., guide renaming of abstract identifiers or specialization of embedding design elements. Usually, pattern semantics is not completely captured by an SDL-fragment. Due to language constraints

this would otherwise result in an overspecification of the design solution and reduce the potential of reuse. Thus, additional *semantic properties* are included, specifying preconditions for pattern application as well as behavioral changes of the embedding context. Though semantic properties are currently stated in natural language, it is possible to express them precisely in a temporal logic. Also, restrictions on the *redefinition* of pattern instances are specified in order to prevent a pattern's intent from being destroyed by subsequent development steps. A comparison to existing description templates for conventional design patterns is given in [15].

The current pool of protocol building blocks contains SDL patterns that deal, for instance, with interaction behavior of distributed objects, error control (lost or duplicated messages), lower layer interfacing, or dynamic establishment and closing of connections. To further illustrate the functional scope of SDL patterns, we briefly introduce some examples. Note that the SDL patterns below are not completely specified. We basically summarize a pattern‘s intent and skip the description items explained above. For completely defined SDL patterns, the reader is referred to [16].

- **MultipleRequestsMultipleReplies:**

The MultipleRequestsMultipleReplies pattern introduces a confirmed 1:n interaction between one sending entity and multiple receiving entities. Being triggered, the sender will initiate a request and waits until all corresponding replies are received. After reception of a request each receiver sends at least one reply.

- **Codex:**

The Codex pattern provides mechanisms to allow two (or more) entities, which interact directly through SDL channels, to cooperate by means of a given communication service. In general, the introduction of a basic service involves many specialties. Among others, these are segmentation, reassembly, upgrade of basic service quality (e.g., in case of loss, disruption, or duplication of messages), lower layer connection setup, or routing decisions. The Codex pattern is only concerned with a minimal subset of these functionalities, namely interfacing with the basic service by means of service primitives. That is, Codex essentially provides a mapping from protocol data units to basic service primitives and vice versa.

- **DynamicEntitySet**:

Consider a given server entity that is capable of providing its service exactly one time and terminates thereafter. In order to offer this service several times (e.g., to more than one client), the DynamicEntitySet pattern is introduced. Thereby an administrator dynamically creates a new server entity for each service request and subsequently acts as a proxy.

2.3 Configuration Process

For the design of SDL protocol specifications we have defined a configuration process supporting the reuse of protocol building blocks represented as SDL patterns (Fig. 1). The configuration process suggests incremental protocol engineering, where the whole set of communication requirements is first decomposed, i.e., partitioned and (where appropriate) simplified. Decomposition classifies as an analysis task that identifies separate protocol functionalities. Thereby it is possible to consider a protocol functionality under different assumptions. For instance, interaction sequences for connection establishment are less complex on top of a reliable basic service rather than an unreliable basic service. Experience has shown that protocol functionalities can often be specified one after the other and - in addition - be completed stepwise (e.g., adapted to the non-ideal properties of an underlying basic service; see, e.g., [15] [23] [19]). This suggests that we perform an individual development step in order to incorporate an additional protocol functionality or relax a corresponding simplification. Thereby, each development step divides into analysis, design, and validation and yields an executable SDL design specification. In the following, the different activities within a development step are sketched.

First, an object-oriented analysis of the current protocol functionality is performed. This results in an analysis model updated from the previous development step. It is suggested to provide an OMT [24] or UML [5] object model and an MSC [21] use case model which together identify participating objects and typical interaction scenarios.

The analysis model is realized in the following design activity. Here, SDL patterns come into place. Starting point is the context SDL specification, i.e., the SDL design specification obtained from the previous development step[1]. This may, e.g., be a protocol specification, which relies on reliable basic service. Hence the design problem (stated in the analysis model) could then be to suit the protocol to an unreliable basic service. In order to meet the new requirements, a number of design steps are performed where individual SDL patterns are applied to the context specification. Note that for some design problems the pool of predefined protocol building blocks may not contain an adequate solution, so that an adhoc solution must be found. The selection of an SDL pattern is supported by several items of the SDL pattern description template, namely *intent, motivation, structure, message scenario, semantic properties,* and *cooperative usage*. As patterns represent generic design solutions, the corresponding SDL-fragment has to be adapted in order to seamlessly fit the embedding context. This is instructed by the *renaming parts* of the *syntactical embedding rules*. Finally, the resulting pattern instance has to be composed with the embedding context, which is prescribed by the specialization part of the syntactical embedding rules and also by *redefinition rules* of embedding pattern instances.

1 For the first development step, the initial context specification is either empty or given by an instantiated SDL framework.

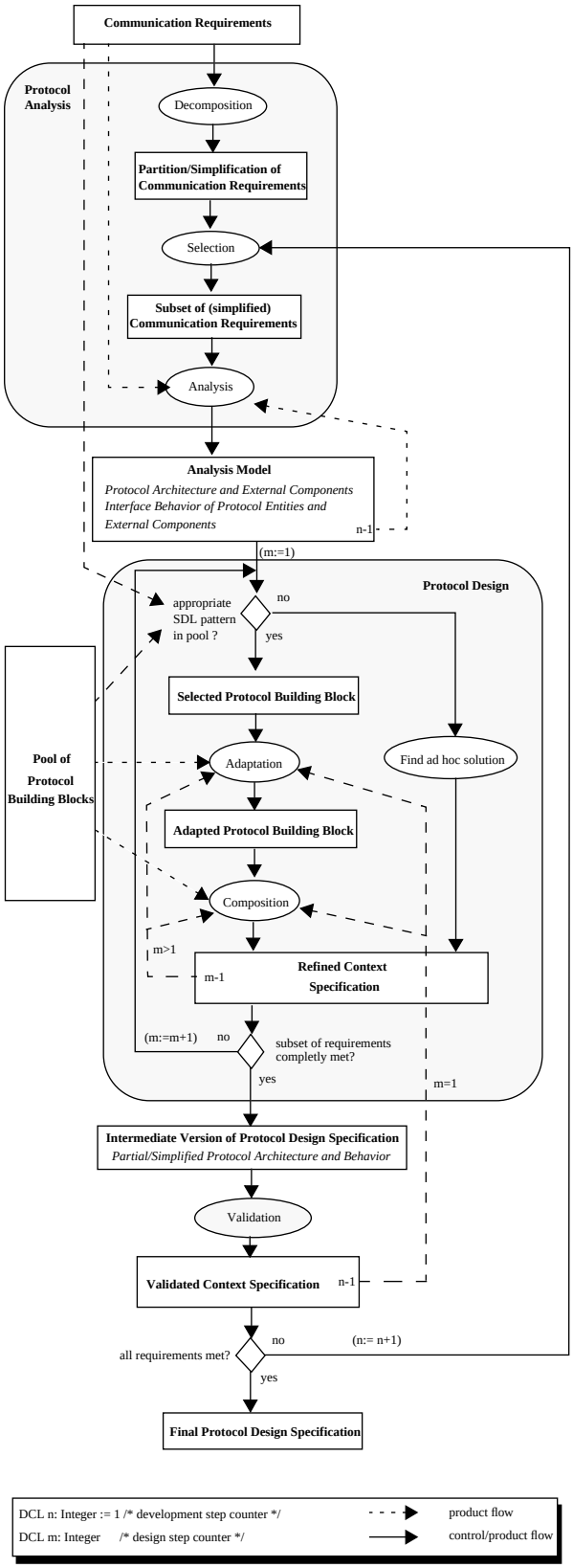

Figure 1. Configuration process model (partial)

The result of this design activity is an intermediate SDL design specification which is subsequently validated against the analysis use case model by performing MSC-based validation and simulation. The correctness of the SDL specification concerning general properties such as freedom from deadlocks is also checked. If any faults are discovered, a return to one of the previous development or design steps is needed (not shown in Fig. 1). Otherwise the validated specification serves as the context specification for the next development step. If all simplifications are eliminated and all requirement subsets are implemented, the final design specification is given by the validated design specification of the last development step.

3 THE EXPERIENCE BASE: SFB-EB

This section describes the current instantiation of the Experience Base of the SFB 501. First, the logical structure of the Experience Base is sketched, as far as it is relevant to the SDL-pattern based design of communication protocols (Section 3.1). Then a usage scenario about how this structure supports the SDL-pattern approach is given in Section 3.2.

3.1 Logical structure of the SFB-EB

The SFB-EB acts as a repository for experience. Therefore, a logical structure was defined that organizes the Experience Base and supports the search and retrieval of experience elements. Fig. 2 shows the parts of the logical structure that are relevant for the SDL-pattern approach.

As shown, the SFB-EB is subdivided into two *sections*, called *experiment-specific section* and *organization-wide section.* These sections are similar to the project databases and organization-wide database described by Basili et. al. [2]. In the experiment-specific section all information concerning single projects, like case studies and controlled experiments, are stored according to predefined templates [9]. These templates are based on the steps of the QIP and are completed while the project is conducted, i.e., planned, executed, and analyzed.

The organization-wide section stores experience relevant to several projects (such as the SDL pattern pool). It consists of different *areas*. Areas are the main building blocks of the SFB-EB. They can be seen as modules that are added to the existing SFB-EB instantiation depending on what should be supported. Therefore, all areas are disjunct. An example for an area is the glossaries area which provides definitions for terms commonly used in projects and experience elements.

Areas can be further refined (indicated by the dashed lines in Fig. 2). For example, the glossaries area is split up into a GQM glossary defining terms concerning GQM-based [3] measurement activities, an SDL glossary with

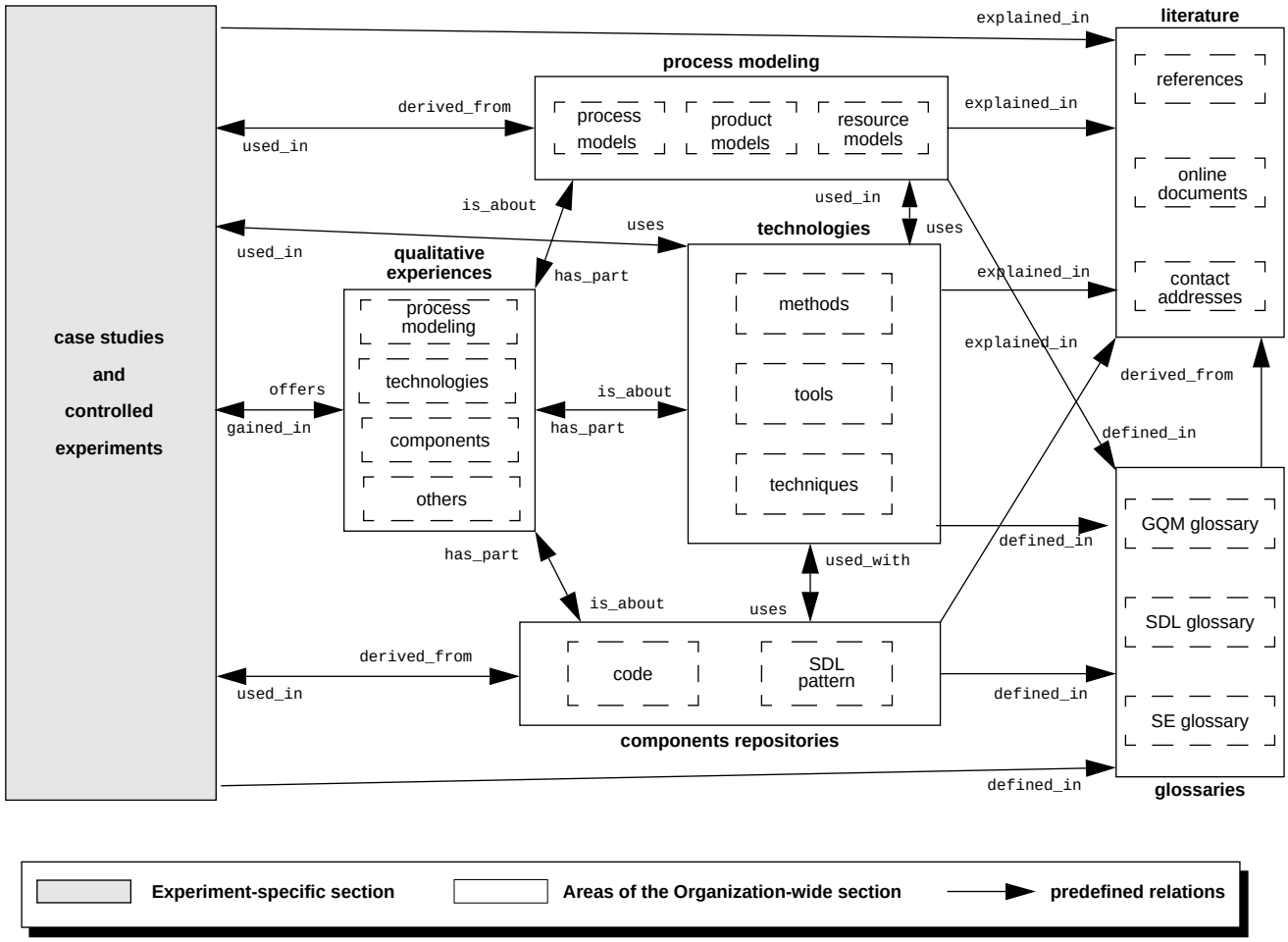

Figure 2. Logical Structure of the SFB-EB regarding SDL pattern oriented development of communication protocols

SDL-related definitions, and a general Software Engineering glossary providing definitions for terms like process, product, etc..

Besides the glossaries area, the following areas are instantiated:

The component repositories area. This area contains components that can be reused in different projects. For example, SDL patterns such as the DynamicEntitySet pattern (Section 2.2), or C++ code for checksum algorithms are stored here.

The process modeling area. Process, product, and resource models, describing how to conduct a project or apply a technique, are offered in this area. For example, the configuration process for the application of the construction set of protocol building blocks (Section 2.3), the SOMT process model [25], or the Bræk and Haugen [6] model for

developing real-time systems with SDL are provided. Most of these models are represented using the process modeling language MVP-L [7].

The technologies area. For different techniques, methods, and tools, so-called *technology packages* are stored in this area. These packages contain basic information about the technologies and help to select the appropriate techniques when setting up a new project. The SDT (SDL Design Tool) package, e.g., helps newcomers to get into the SDL development environment.

The qualitative experiences area. All lessons learned while conducting projects are represented in this area. They are categorized according to the topics they are dealing with. Currently we deposit experiences about adequate decomposition of communication requirements in this area (Section 2.3).

The literature area. Background knowledge in the form of (external) references, on-line documents, and contact addresses is provided within this area. For instance, a reference to the SDL forum society web page[2], or relevant papers dealing with communication protocols can be found here.

Between the areas of the organization-wide section listed above and the experiment-specific section, different *relations* are defined. Some typical relations are `experience element X `uses` experience element Y´ or `experience element X is `derived_from` experience element Z´ where X,Y and Z belong to different areas/sections. These relations help support the search for experience elements in a given project context. For instance, from an experience element that describes a concrete process model of a project (and therefore is stored in the experiment-specific section), one can follow the `uses` relations to the experience elements that describe the technologies addressed within the process model (Note that for a certain SDL pattern project different description techniques for protocol analysis or different code generators for simulation could be applied). From the same process model, the `derived_from` relation helps finding the experience element describing the general process model from which the actual one was derived and which is stored in the process modeling area.

The areas and the defined relations together form a framework that represents the logical structure of the SFB-EB. Note that the SFB-EB is also developed and used for contexts other than the SDL pattern approach. A detailed discussion of the complete logical structure and the technical realization using HTML-pages that are accessible via the SFB 501 intra-net can be found in [13].

2 http://www.sdl-forum.org/

3.2 A usage scenario

This section discusses a specific usage scenario on how the SFB-EB supports the SDL pattern approach. The given scenario only describes steps that are already supported by the SFB-EB. Usage scenarios on how the SFB-EB structure supports systematic reuse in projects according to the steps of the QIP are given in [10] and [11].

Planning a project:

When setting up a new project, it must first be characterized. The answers to questions like "*What is the goal of the project?*", "*In which environment is the project to be conducted?*", and "*Are there time restrictions for the project?*" give first hints for planning the project. With this information one can search the process modeling area of the SFB-EB for similar project plans and descriptions of technologies suitable for the new project. At the end of this first step:

- A new entry for the project is created in the experiment-specific section of the SFB-EB. In accordance with the predefined template all documents produced for the project will be stored in this entry.
- The characterization of the new project is stored in the new entry of the experiment-specific section.
- A project plan and/or technology description from the organization-wide section that can be adapted to the new project is selected from the SFB-EB.

Let us assume that the new project were to develop a communication protocol, in the environment of the SFB 501, by the end of the year. A process description for developing communication software with SDL patterns was found. The `uses` *relation of the process description provides a link to the SDL technology package in the technologies area. Unfortunately, there are no* `has_part` *relations from the SDL technology package to some lessons learned in the quantitative experiences area that would support us with more information about the usage of SDL in former projects. But with the help of the* `used_in` *relation(s) stored with the process description, information about former projects that used the process before can be viewed.*

In the next planning step a project plan has to be defined for the new project. The plan has to include information about resources (people and tools) that will be used. For this purpose, information is needed about appropriate technologies that have been selected for the project. This is supported by the technology packages stored in the technology area. In addition to this, the goal(s) of the project, regarding the validation of some hypotheses, will be defined in a quantitative manner using the GQM paradigm [3]. At the end of this planning step:

- A concrete project plan is added to the project entry in the experiment-specific section.
- A list of needed resources (people and technologies to be used) exists.
- The quantitatively defined goal(s) of the project is stored in the project entry in the experiment-specific section together with the GQM plan(s), measurement plan, and questionnaires that will be used to control the project in the execution phase.

- A `derived_from / used_in` relation between the project plan of the project and an experience element in the process modeling area might be added to the SFB-EB.
- `Uses / used_in` relations between technologies used in the project and technology packages in the technologies area might also be added to the SFB-EB.

Let us assume that the new project uses the configuration process described in Section 2.3. Therefore, SDL patterns will be used to develop the communication protocol. A project plan is developed and researchers from the University of Kaiserslautern are chosen to develop the communication protocol.

Finally, it must be checked if all needed resources are ready to be used in the project. This includes the setup and installation of the needed tools and maybe the preparation of training of the people regarding the technologies and development process that will be used within the project. The installation of the tools may be supported by the description given in the technology packages and/or the links given by the `explained_in` relations from the technology packages of the tool to the literature area. The same relations may also support the setup of the training material or the technologies or the used process description from the process modeling area. Reading the lessons learned regarding technologies can help avoid problems in the new projects. At the end of this planning step:

- All tools needed for the project are installed and ready to use.
- Training materials for the people that will conduct the project are prepared.

Let us assume that the tools needed for the project already have been implemented, but the chosen researchers, that should conduct the project, do not have sufficient knowledge about communication protocols using SDL and design patterns. Therefore they are asked to read the technology packages stored in the technologies area and the SDL glossary in the glossary area. Furthermore, they can use the `explained_in` *link to the web page of the SDL forum society to gain further knowledge about SDL and the* `defined_in` *relations to look up the definitions of commonly used terms in the glossaries.*

Executing the project:

The project is executed according to the project plan. If possible, components from the component repository area will be reused for the new project. Furthermore, measurement data is collected according to the GQM-based measurement program to control the project and gain new knowledge that can be used to test the hypotheses that have been formulated for the project. At the end of the execution step:

- Measurement data has been collected and stored in the project entry in the experiment-specific section.
- `Used_in` and `derived_from` relations between objects from the component repositories and the new project might be added to the SFB-EB.

Let us assume that the project has been developed according to the process described in Section 2.3. The developers used SDL patterns from the component repositories area to develop the communication protocol. `Defined_in` *and* `explained_in` *relations between the objects of the component repositories and the technology packages have been used to look up some definitions when problems occur. Furthermore, some experiences with the SDT tool and the SDL patterns have been made that have not been expected. Therefore, these experiences have been reported, i.e., written down and stored in the project entry of the experiment-specific section. Note that the* `gained_in` *and* `offers` *relations to the qualitative experiences area are not added during the execution of the project. This is done later when the project experiences are analyzed and fixed. Last, but not least, the new communication protocol was developed.*

Fixing the project experiences:

The data that has been collected during the execution of the project is now processed and analyzed. With its help the questions concerning the measurement program goals are answered, i.e., the hypotheses that were formulated at the beginning of the project are tested to see if they have been validated or must be rejected.

To find out if additional knowledge that was not captured by the measurement data, was gained while executing the project, the people from the development team are interviewed. From these interviews and the experience that has been reported and stored in the project entry of the experiment-specific section, lessons learned are formulated and stored in the qualitative experiences area, if they are of interest for other projects. Both the analyzed measurement data and the captured lessons learned can then be used to improve the technologies (e.g., tools), process models (e.g., the SDL-pattern process), and components (e.g., the SDL- patterns) that were used in the project. At the end of the analysis step:

- The hypotheses that were to be tested can be validated or must be rejected.
- New lessons learned might be introduced into the qualitative experiences area and the `is_about` / `has_part` relations from the qualitative experiences area to the project entry in the experiment-specific section are introduced.
- Technologies, process models, and components might have been validated and can be more trusted when being selected for use in upcoming projects.

After we have described all relevant actions that can be supported by the current structure of the SFB-EB in this fictive scenario the next section discusses experiences gained in a concrete research case study that was conducted within the SFB 501.

4 FIRST EXPERIENCES: A CASE STUDY

To check some of our hypotheses, a first case study has been set up. In general, several goals of the case study would be interesting in order to evaluate our SDL pattern approach. These include, e.g., the analysis of our construction set of protocol building blocks with respect to reduction of development effort or the readability / intelligibility of our pattern descriptions. For the initial case study we decided to analyze the influence of pattern usage on the number and distribution of errors and faults (For a definition of these terms see Fig. 3). This serves as a baselining for future experiments. The other goals are not part of the initial case study, but were postponed to later projects. This is due to the fact that our case study is performed as students' project work and, therefore, has a time limitation of four months. As subject of development we have chosen the Real Time Transport Protocol (RTP) of the Internet protocol suite. Our decision was driven by the fact that we already had experiences with the reengineering of Internet-based protocols and that RTP is well suited to complete the set of protocols we already developed (e.g., IPv6 or ST2+ [23]).

During the planning of the project, suitable technologies had to be chosen. As the subject of analysis is our construction set of protocol building blocks, we (re)used the corresponding SDL-pattern process (Section 2.3) as it is stored in the process modeling area of the Experience Base.

In the next step, a project plan was defined based on the process description. Students that attended a lecture about communication protocols at the University of Kaiserslautern were chosen to conduct the project as part of their education. It was decided to use the SDT-tool from Telelogic to support the design, FrameMaker as a word processor, and the SFB-EB with the stored information about the SDL patterns as the only tools for the project.

The chosen experimental goal of the case study was formalized according to the GQM paradigm. The white oval in Fig. 3 shows the formalized goal describing the object, purpose, quality focus, viewpoint, and context of the study.

After the goal was fixed, we formulated questions that refine the quality focus of our study, with additional questions that capture the variation factors which have influence on the results of the observed quality factors. The quality focus we identified is characterized by the following items:

- The distribution of faults to fault types such as deadlock, implicit consumption of signals, incomplete or incorrectly implemented requirements. The hypothesis we proposed is that 50% of the faults normally correspond to the last fault type.
- The error type that caused a fault, with the types being named according to the activity in which they occurred. Examples are design error, selection error, decomposition error, or analysis error. We expect that 70% of faults

GQM plan errors and faults

This GQM plan is designed for a case study regarding the construction set of protocol building blocks. A quality model based on error and fault occurrences is defined. Therefore, the following definitions are used:

- **Def. failure:** Departure of observed behavior from expected behavior
- **Def. fault:** Inconsistency in product that causes failure(s);
- **Def. error:** Human action resulting in software with fault(s);

Analyze the construction set of protocol building blocks
for the purpose of characterization
with respect to errors and faults
from the viewpoint of the project leader
in context of the project RTP.

[…]

Question and metric form the Model Definition:

Q: Which type of faults occurred and how are they distributed to the different types?
M: type of fault
[deadlock / implicit consumption / requirement not completely covered / requirement not correctly implemented / other]

[…]

A surveyed variation factor regarding the listed question above is described by the following question:

Figure 3. Excerpts from the GQM plan “errors and faults”

are due to errors made during design.

- The distribution of the design steps in which an error occurred to the different development steps. Development steps are characterized by requirements that are to be realized. Examples are protocol operations or basic service-related tasks. As a hypothesis we state that 60% of the errors occur in development steps in which protocol operations are to be realized. Therefore our pattern pool should preferably support these kinds of design problems.
- The distribution of errors to adhoc design steps and design steps supported by SDL patterns. We expect that 90% of errors occur in adhoc design steps.

Actually, the quality focus is influenced by some variation factors. An example is the knowledge of the developer about other communication protocols. The more familiar the developer is with the communications area, the less faults of the type deadlock, implicit consumptions, or requirements not correctly implemented will occur. In Fig. 3 an example for a question of the quality focus with a corresponding question that describes a variation factor and how it influences the quality focus is given. (The complete GQM plan can be found in appendix A)

Finally, we prepared some training materials about our construction set of protocol building blocks and communication protocols in general. For this, knowledge stored in the literature and glossaries area of the experience base was used. Since no tools needed to be prepared, the planning phase was completed and we started the execution of the project according to the project plan supervised by the measurement program. The needed data were collected with the help of questionnaires that had to be filled out at the beginning or end of predefined steps of our process model.

Right now we are in the phase of analyzing the collected data. As a first result we can state that decomposition plays a more important role than expected. Good decomposition is a prerequisite for successful design with a low number of errors. What we have to improve is that enough time should be allowed for the decomposition activity and that it should also be better supported by the experience base. Therefore we will add our experiences concerning decomposition as certain rules of thumb. The usage of the experience base has been found useful throughout the case study. Nevertheless, we found that it is necessary to extend the training with a more detailed section about how to use the information of the experience base.

5 CONCLUSION AND OUTLOOK

We have introduced the SDL pattern approach which integrates the well-known design patterns concept with the formal design language SDL. As a major advantage, SDL patterns allow to precisely specify knowledge about pattern application and its impact on the embedding context. We consider formalization to be a prerequisite for tool support and validation of pattern application. To show the feasibility of the approach, several test projects have been conducted [16] [19] [23]. However, with the current status of the approach we wish to have a more systematic method to investigate certain details concerning SDL patterns. Additionally, it is essential to have an infrastructure available that helps to continuously improve the concepts, as good patterns mainly arise from practical and well-founded experiences. For this purpose we combined the SDL pattern approach with the experience base concept. The SFB-EB serves as a central reuse repository for all kinds of SDL pattern specific experiences and allows us to effectively set up new projects with a corresponding measurement program. Knowledge is systematically transferred into new projects and experiments, so that the SDL pattern approach can be continuously improved. A

usage scenario for the SFB-EB was discussed in detail, in which the main activities are planning, executing, and analyzing the project. Also, a research case study on the experiment-based evaluation of the SDL pattern approach was conducted. As a first result it turned out that an adequate knowledge of pattern-based design in general is of great importance for the success of a project, which made us reconsider the quality of our training material in the SFB-EB. Furthermore, experience has shown that decomposition and analysis have greater impact on pattern-based development than expected. With the current pool of SDL patterns, a bad decomposition or analysis model may result in poor coverage of SDL patterns in the resulting specification. However, further experiments will have to clarify the exact reasons.

From our first results with the SFB-EB we infer that it is a valuable means for supporting the evaluation and continuous improvement of the SDL pattern approach. Thus we plan to conduct more SDL-pattern based projects within this context. This necessitates some extensions of the SFB-EB. For instance, we have to add a measurement area to the organization-wide section of the SFB-EB to store basic GQM plans (e.g., to rate the quality of newly-defined patterns). In the long run, it may also be necessary to fully realize the QIP steps during a project, and, therefore, complete our Experience Base to an Experience Factory [2].

ACKNOWLEDGMENTS

This work is supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Sonderforschungsbereich 501 ‘Development of Large Systems with Generic Methods’ (SFB 501) at the University of Kaiserslautern. Thanks go to our project leaders Prof. Dr. H. Dieter Rombach and Prof. Dr. Reinhard Gotzhein. We would also like to thank our colleagues Stefan Vorwieger and Ralf Reussner from the SFB 501 for their contributions that helped improve this paper. Last but not least, the assistance provided by Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering (IESE) in reviewing the first versions of this paper was much appreciated.

Appendix

A GQM PLAN ERRORS AND FAULTS

This GQM plan is designed for a case study in the subproject B4 of the SFB 501. The goal of the case study is to give a first characterization of the process using the construction set of protocol building blocks developed within B4. Therefore a quality model based on error and fault occurrences is defined.

Analyse: the construction set of protocol building blocks [object of study]
for the purpose of: characterization [purpose of study]
with respect to: errors and faults [quality focus of study]
from the Viewpoint of: the project leader [viewpoint of study]
in the Context of: the project RTP. [context of study]

Process Definition

Domain Conformance

D_1: How great is the experience of developer with SDL?

Hypothesis: Less experience results in more faults of the type deadlocks, implicit consumption, and requirements not correctly implemented. (see Q_1)

D_1.1: Has the developer known SDL before the current project?

MD_1.1.1: knowledge about SDL
[yes / no]

D_1.2: If question D_1.1 was answered with yes: Where was the knowledge about SDL gained?

MD_1.2.1: origin of knowledge about SDL
[courses … / labs … / seminars … / literature … / WWW … / others …]

D_1.3: If question D_1.1 was answered with yes: How often and how long was SDL used for developing software designs before?

MD_1.3.1: usage of SDL in former projects
[list of projects with project duration in months]

D_2: Of what kind is the knowledge of the developer about communication protocols?

Hypothesis: The smaller the knowledge, the more faults of the type deadlocks, implicit consumption, and requirements not correctly implemented will occur. (see Q_1)

D_2.1: Has the developer knowledge of communication protocols?

MD_2.1.1: knowledge of communication protocols
[yes / no]

D_2.2: If question D_2.1 was answered with yes: Which communication protocols does the developer know and where was the knowledge gained?

MD_2.2.1: origin of knowledge about communication protocols
[list of communication protocols with source of knowledge] *(source of knowledge: e.g., courses / labs / seminars / HiWi jobs)*

D_2.3: If question D_2.1 was answered with yes: Which communication protocols have been developed before and in which context?

MD_2.3.1: usage of communication protocols in former projects
[list of projects with used protocols]

D_3: Knowledge and understanding of the construction set of protocol building blocks?

Hypothesis: The worse the knowledge and understanding of the construction set of protocol building blocks, the more faults of the type deadlocks, implicit consumption, and requirements not correctly implemented will occur. (see Q_1)

Hypothesis: In design parts with support of SDL patterns the number of errors will be lower. (see Q_4)

D_3.1: Does the developer have knowledge about pattern based design in general?

MD_3.1.1: knowledge about general pattern based design
[yes / no]

D_3.2: If question D_3.1 was answered with yes: Where was the knowledge gained?

MD_3.2.1: origin of knowledge about general pattern based design
[list of sources]

D_3.3: If question D_3.1 was answered with yes: How often and how long was general pattern based design used for developing software designs before?

MD_3.3.1: usage of general pattern based design in former projects
[list of projects with project duration in month]

D_3.4: Does the developer have knowledge about the construction set of protocol building blocks?

MD_3.4.1: knowledge about the construction set of protocol building blocks
[yes / no]

D_3.5: If question D_3.4 was answered with yes: Where was the knowledge gained?

MD_3.5.1: origin of knowledge about the construction set of protocol building blocks
[list of sources]

D_3.6: If question D_3.4 was answered with yes: How often and how long was the construction set of protocol building blocks used for developing software designs before?

MD_3.6.1: usage of the construction set of protocol building blocks in former projects
[list of projects with project duration in month]

D_3.7: Did the developer apply SDL patterns whenever possible?

MD_3.7.1: SDL patterns applicable for the design step
[list of SDL pattern names (e.g. TimerControlledRepeat, BlockingRequestReply, DynamicEntitySet, Codex, …)]

MD_3.7.2: selected SDL pattern
[SDL pattern name]
Developer selects SDL patterns but still has the chance of applying the SDL pattern or deciding against the SDL pattern

MD_3.7.3: number of applied SDL patterns in development step
[integer]
Developer selects SDL pattern in design step and applies the offered SDL pattern.

D_4: How is the quality of the construction set of protocol building blocks?
(The quality includes the offered experience regarding SDL patterns!)

Hypothesis: Higher quality reduces the number of faults. (see Q_1)

Hypothesis: Greater experience regarding SDL patterns results in less selection, decomposition, and analysis errors. (see Q_2)

Hypothesis: The better the coverage of a development step, the less errors occur. (see Q_3)

Hypothesis: The higher the quality the less the number of errors in the SDL pattern. (see Q_4)

D_4.1: How often was the description of a selected and applied SDL pattern incorrect?

MD_4.1.1: quality of SDL pattern description
[correct / inconsistent / incomplete / has SDL error / others …]

D_4.2: Why was no SDL pattern applied in the design step?

MD_4.2.1: design step without applied SDL pattern
[name of design step]

MD_4.2.2: reasons for not applying SDL pattern
[no pattern was offered / selected pattern was not understandable / others …]

D_4.3: Are there types of development steps not mentioned in the offered “experiences”?

MD_4.3.1: development step types not mentioned in “experiences”
[list of new development step types with short characterization]

D_5: How is the complexity of the communication requirements?

Hypothesis: The more complex the communication requirements, the more faults occur. (see Q_1)

Hypothesis: The more complex the communication requirements, the more selection and decomposition errors occur. (see Q_2)

Hypothesis: The more complex the communication requirements, the more errors will occur in AdHoc design steps. (see Q_4)

D_5.1: Which concurrent protocol functions could be identified?

MD_5.1.1: concurrent protocol functions
[list of function names]
usually protocol (i.e. project) specific names

D_5.2: Are protocol functions partitioned into layers?

MD_5.2.1: requirements partitioned into layers
[list of layers with name + characterization]

D_5.3: What protocol phases are included?

MD_5.3.1: name of included protocol phases
[connection set-up, connection tear down, data transfer, others …]

D_5.4: What type of service shall be realized?

MD_5.4.1: type of service
[reliable / unreliable, connection-oriented / connection-less, window-based flow control / rate-based flow control, non-functional (like delay, jitter, max. throughput), others …]
Note: more then one type can be selected!

Qualities

Model :errors and faults

Def. Failure: Departure of observed behaviour from expected behaviour
Def. Fault: Inconsistency in product that causes failure(s);
Def. Error: Human action resulting in software with fault(s);

Model Definition

Q_1: Which type of faults occurred and how are they distributed to the different types?

MQ_1.1: type of fault
[deadlock / implicit consumption / requirement not completely covered / requirement not correctly implemented / other …]

Q_2: For each detected fault: What type of errors caused the fault?

MQ_2.1: type of error
[design error / selection error / decomposition error / analysis error]

MQ_2.2: errors that caused the fault
[list of errors]

Q_3: In which design steps did an error occur and how are the errors distributed over the development steps?
List of all applied development steps can be calculated from all names mentioned in MQ_3.2.

MQ_3.1: design step in which the design error occurred
[name of design step]

MQ_3.2: development step of design step
[protocol operations / basic service / mode / multiplicity / other …]

MQ_3.3: development step of selection error, decomposition error or analysis error
[protocol operations / basic service / mode / multiplicity / other …]

Q_4: How are (design) errors distributed to AdHoc design steps and SDL pattern supported design steps?

MQ_4.1: design step in which the design error occurred
[name of design step]
Note: identical with MQ_3.1!

MQ_4.2: type of design step
[pattern design step / AdHoc design step]

B AN EXAMPLE QUESTIONNAIRE FROM THE RTP CASE STUDY

Questionnaire FB 1

To be filled in by: Protocol developer before decomposition
(once at the beginning of the case study)

Date: __.__.____ **Bearbeiter:_____**

Q 1: Do you have knowledge about SDL? ❒ yes ❒ no

If answered with „yes“:

Q 1.1: Where was your knowledge gained?
(KTPS course, Rechnernetze course, SFB lab, Rechnernetze seminar, literature, WWW, Hiwi job, ...)

_____
_____
_____
_____
_____

Q 1.2: Have you already applied SDL in former projects? ❒ yes ❒ no

If answered with „yes“:

Q 1.2.1: Which projects and how long did they last (in month)?

_____
_____
_____
_____
_____

Validated by: A1 **Person: _____** **Date: __.__.____**

RTP Project of B4 **Page: 1/4**

Questionnaire FB 1

To be filled in by: Protocol developer before decomposition
(once at the beginning of the case study)

Date: __.__.____ **Bearbeiter:_____**

Q 2: Do you have knowledge about communication protocols? ❒ yes ❒ no

If answered with „yes“:

Q 2.1: Please list protocols you are familiar with?
(e.g. IP, TCP, OSI-TP, Alternating-Bit, XTP)
_____
_____

Q 2.2: Where was your knowledge gained?
(KTPS course, Rechnernetze course, SFB lab, Rechnernetze seminar, literature, WWW, Hiwi job, ...)
_____
_____

Q 2.3: Have you already developed some communication protocols in former projects? ❒ yes ❒ no

If answered with „yes“:

Q 2.3.1: Which projects and how long did they last (in month)?
_____
_____

Validated by: A1 **Person: _____** **Date: __.__.____**

RTP Project of B4 **Page: 2/4**

Questionnaire FB 1

To be filled in by: Protocol developer before decomposition
(once at the beginning of the case study)

Date: __.__.____ **Bearbeiter:_____**

Q 3: Do you have knowledge about pattern-based design in general? ❒ yes ❒ no

If answered with „yes“:

Q 3.1: Where was your knowledge gained?
(course, lab, seminar, literature, WWW, Hiwi job, ...)
_____
_____
_____

Q 3.2: Have you already applied pattern-based design in former projects?
❒ yes ❒ no

If answered with „yes“:

Q 3.2.1: Which projects and how long did they last (in month)?
_____
_____
_____
_____
_____

Validated by: A1 **Person: _____** **Date: __.__.____**

RTP Project of B4 **Page: 3/4**

Questionnaire FB 1

To be filled in by: Protocol developer before decomposition
(once at the beginning of the case study)

Date: __.__.____ **Bearbeiter:_____**

Q 4: Do you have knowledge about the construction set of protocol building blocks? ❒ yes ❒ no

If answered with „yes“:

Q 4.1: Where was your knowledge gained?
(SFB lab, literature, Hiwi job,...)
_____
_____
_____
_____

Q 4.2: Have you already applied the construction set of protocol building blocks in former projects? ❒ yes ❒ no

If answered with „yes“:

Q 4.2.1: Which projects and how long did they last (in month)?
_____
_____
_____
_____

Validated by: A1 **Person: _____** **Date: __.__.____**

REFERENCES

[1] B. Algayres, Y. Lejeune, and F. Hugonnet. GOAL: Observing SDL behaviors with Object-GEODE. In *Proceedings of the 7th SDL Forum*, Oslo, Norway, 1995.

[2] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Experience Factory. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 469–476. John Wiley & Sons, 1994.

[3] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Goal Question Metric Paradigm. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 528–532. John Wiley & Sons, 1994.

[4] Victor R. Basili and H. Dieter Rombach. The TAME Project: Towards improvement–oriented software environments. *IEEE Transactions on Software Engineering*, SE-14(6):758–773, June 1988.

[5] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language, Version 1.0*. Rational Software Corporation, 1997.

[6] R. Bræk and O. Haugen. *Engineering Real-time Systems: An Object-oriented language Methology using SDL*. Prentice Hall, London, 1993.

[7] Alfred Bröckers, Christopher M. Lott, H. Dieter Rombach, and Martin Verlage. MVP–L language report version 2. Technical Report 265/95, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, 1995.

[8] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture – A System of Patterns*. John Wiley and Sons, 1996.

[9] Marion Fechtig. Fixing the case studies' structure for the access and storage system of the experiment-specific section in the SFB 501 Experience Base (in German). Projektarbeit, Dept. of Computer Science, University of Kaiserslautern, Germany, 67653 Kaiserslautern, Germany, January 1998.

[10] Raimund L. Feldmann, Jürgen Münch, and Stefan Vorwieger. Experiences with systematic reuse: Applying the EF/QIP approach. In *Proceedings of the European Reuse Workshop*, Brussels, Belgium, November 1997.

[11] Raimund L. Feldmann, Jürgen Münch, and Stefan Vorwieger. Towards Goal-Oriented Organizational Learning: Representing and Maintaining Knowledge in an Experience Base. In *Proceedings of the Tenth International Conference on Software Engineering and Knowledge Engineering (SEKE'98)*, San Francisco, USA, June 1998.

[12] Raimund L. Feldmann and Stefan Vorwieger. Providing an Experience Base in a research Context via the Internet. In *Proceedings of the ICSE 98 Workshop on "Software Engineering over the Internet"*, Kyoto, Japan, April 1998.

[13] Raimund L. Feldmann and Stefan Vorwieger. The web-based Interface to the SFB 501 Experience Base. Technical Report 1, Sonderforschungsbereich 501, Dept. of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, 1998.

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

[15] B. Geppert, R. Gotzhein, and F. Rößler. Configuring Communication Protocols Using SDL Patterns. In A. Cavalli and A. Sarma, editors, *SDL'97 - Time for Testing, Proceedings of the 8th SDL-Forum*, Evry, France, September 1997. Elsevier Science Publishers.

[16] B. Geppert and F. Rößler. Pattern-based Configuring of a Customized Resource Reservation Protocol with SDL. Technical Report 19, Sonderforschungsbereich 501, Dept. of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, 1996.

[17] B. Geppert and F. Rößler. Combining SDL and Pattern-Based Design for the Customization of Communication Subsystems. In A. Wolisz, I. Schieferdecker, and A. Rennoch, editors, *Formale Beschreibungstechniken für verteilte Systeme, GI/ITG-Fachgespräch*, pages 201–210, Berlin, Germany, June 1997. (GMD-Studien Nr. 315) ISBN 3-88457-315-2.

[18] B. Geppert and F. Rößler. Generic Engineering of Communication Protocols - Current Experience and Future Issues. In *Proceedings of the 1st IEEE International Conference on Formal Engineering Methods (ICFEM'97)*, Hiroshima, Japan, 1997.

[19] B. Geppert, F. Rößler, and M. Schneider. Using SDL Patterns for the Design of a CAN-based Communication Subsystem. In *8. GI/ITG-Fachgespräch "Formale Beschreibungstechniken für verteilte Systeme"*, Cottbus, Germany, June 1998.

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

[21] ITU–T. *Recommendation Z.120 (10/96) – Message Sequence Chart (MSC)*, 1996.

[22] Barbara Ann Kitchenham. Evaluating software engineering methods and tools, part 1: The evaluation context and evaluation methods. *ACM SIGSOFT Software Engineering Notes*, 21(1):11–15, January 1996.

[23] F. Rößler, B. Geppert, and P. Schaible. Re-Engineering of the Internet Stream Protocol ST2+ with Formalized Design Patterns. In *Proceedings of the 5th International Conference on Software Reuse (ICSR5)*, Victoria, British Columbia, Canada, June 1998.

[24] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[25] Telelogic. *SDT 3.2 Methodology Guidelines – Part1: The SOMT Method*. Telelogic, Sweden, 1997.

[26] Telelogic. *TAU 3.2 User's Manual*. Telelogic, Sweden, 1997.

[27] M. v. Löwis and R. Schröder. Simulation of Telecommunication Systems in SDL-92 using SITE. In Y.M. Teo, W.C.Wong, T.I.Oren, and R.Rimane, editors, *World Congress on Systems Simulation*, Singapur, 1997.