Systematic Evolution of Reusable SDL Patterns [1]

Birgit Geppert
Computer Networks Group
University of Kaiserslautern
Postfach 3049
67653 Kaiserslautern, Germany
b.geppert@computer.org

Frank Rößler
Computer Networks Group
University of Kaiserslautern
Postfach 3049
67653 Kaiserslautern, Germany
f.roessler@computer.org

Raimund L. Feldmann
Software Engineering Group
University of Kaiserslautern
Postfach 3049
67653 Kaiserslautern, Germany
r.feldmann@computer.org

ABSTRACT

The formal description technique SDL is in widespread use in industry as a design language for reactive, distributed systems. Recently, SDL patterns have been introduced as a new concept to increase reusability in SDL based system design. However, developing SDL patterns is a very time consuming task and an ongoing process as one always expects the latest state of the art.

This paper outlines an approach that supports systematic evolution of SDL patterns. It controls the effort by developing the artifacts incrementally and only on demand of application engineers. In accordance with the SDL pattern based reuse process, an improvement cycle is established, which is based on applying the patterns in practice. Gained experience flows directly into the improvement activity, eventually resulting in a new pattern release.

1 INTRODUCTION

For developing distributed systems, formal methods are often applied to increase productivity and ensure high quality of resulting products. A well-tried combination is UML (Unified Modeling Language [2]) together with MSC (Message Sequence Charts [9]) for object-oriented system analysis and SDL (Specification and Description Language [8]) for detailed design.

Analysis of current SDL-based reuse technology yields that component based reuse (in form of SDL procedures and classes) is well supported. High-level reuse as in the case of frameworks [14] or patterns, however, has just recently become a topic of interest in the SDL community. In [7] we present the SDL pattern approach that extends pattern-based reuse [5] for formal SDL design. It comprises a product model for reusable SDL patterns together with an iterative, use-case driven design process.

Experiences with our approach made apparent that reusing SDL patterns is a promising way for transferring existing design knowledge into new projects. However, as a matter of fact, developing such artifacts is quite expensive and time consuming. Additionally, always the latest state of the art is expected, as reuse benefits strongly depend on the quality of reuse artifacts. This all calls for a systematic and efficient method for developing and continuously improving SDL patterns.

In this paper, we discuss an approach for systematically detecting and capturing improvement potential of SDL patterns. In accordance with the pattern-based reuse process, an improvement cycle is established, which is triggered on demand of application engineers and directly exploits practical experience with the reuse artifacts. Thereby, reuse knowledge is systematically and directly captured through a measurement program, which is executed in parallel to the development projects. The approach is organized by a reuse repository, which stores the reuse artifacts and interrelates them with further experience elements that are essential to the improvement cycle.

2 THE SDL PATTERN APPROACH

The Specification and Description Language SDL

SDL is a graphical, object-oriented description technique with a formal syntax and semantics. It is under standardization by the International Telecommunication Union (ITU). An SDL system specification is hierarchically structured into subsystems (SDL blocks) and active objects with data attributes (SDL processes) resulting in a tree-like architecture with the system as root, blocks as intermediate nodes, and processes as leaves. SDL processes are defined in terms of extended finite state machines. They run concurrently and communicate asynchronously by signal exchange.

SDL adopts the object-oriented paradigm. Classes of objects are specified by so-called SDL types which can be specialized by certain inheritance mechanisms. Associations that can be modeled with SDL are block aggregation and object communication. However, multiple inheritance and dynamic binding are not supported.

An advantage of designing a distributed system in SDL is that an SDL design specification is already executable and can be used for simulation and validation. Even code can be automatically derived from an SDL design. SDL system development is well-supported by existing methodologies [10, 13] and tool environments [12].

SDL Patterns

Scattered parts of a given SDL specification may collaborate and thereby offer a certain functionality. By analysis, abstraction, and documentation, such a design solution can be reused whenever the design problem arises again. This is the main idea of software patterns in general and of SDL patterns in particular. Roughly speaking, an SDL pattern is defined as a reusable software artifact representing a generic solution for a recurring design problem with SDL as applied design language. SDL patterns serve three main purposes:

- First, patterns are descriptive in nature. That is, they describe design expertise and experiences gained in prior projects and allow to pass this knowledge on to other developers. A collection of patterns can therefore be seen as some kind of textbook.
- SDL patterns generate a design solution for a given problem. The solution is described by an SDL fragment that consists of several syntactical elements (e.g., transition fragments) which may be scattered over the context specification. Additionally, application rules describe when the pattern could be applied and how to adapt the SDL fragment and compose it with the given context specification. The possibility of precisly specifying application rules, results from the formal basis provided by SDL and is a main advantage compared to conventional design patterns. It is also a prerequisite for tool supported pattern application [3].
- Furthermore, patterns help to document a given design. The information about embedded pattern instances can be

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

inserted during pattern-based design, i.e., when applying a pattern (top-down documentation). However, pattern information can also be inserted into a given SDL specification by pattern-based reverse engineering (bottom-up documentation). Therefore the SDL specification is scanned for pattern instances and enriched with suitable pattern comments. Pattern documentation strongly profits from our focus on a specific target language, because extended tool support is possible [3].

Design reuse as in the case of patterns is more flexible than code reuse, but the learning curve required before a pattern can be reused could be very high. It is therefore important to keep the specification of SDL patterns precise but intelligible. For this purpose, a standard description template is defined (Table 1).

Reuse Driven SDL Design

An SDL system development process [10, 13] generally encompasses the following development phases: object-oriented analysis, SDL-based design, formal validation, and finally code generation. It is suggested to provide a possibility for reusing artifacts and other material from previous designs that are stored in a reuse repository [10].

In [7] we present an iterative, use-case driven design process tailored to SDL patterns. First, system requirements are decomposed. This includes developing an outline system architecture, finding and exploring use cases and breaking them down into single collaborations between the design elements. Detailed design is done incrementally by incorporating the discovered collaborations step by step. In each design step we analyze, if for the current design problem existing SDL patterns could be applied or if an ad hoc solution must be developed. If a pattern has been selected, it must be adapted and is finally composed with the embedding context specification according to its application rules. The result of each design step is an executable SDL design specification, which is validated before entering the next design step. A comprehensive case study is described in [11].

3 CONTINUOUSLY IMPROVING SDL PATTERNS

At least some informal improvement process is necessarily common practice for each kind of reuse artifact. This is due to the fact that artifacts should fit many application contexts and be reused by different developers. Usually, we do not capture all possible achievements of a reuse artifact at the first go. Thus, ideas for a high-quality artifact come in portions while experience accumulates.

In traditional development and improvement processes, the first application of a reuse artifact requires a relatively high initial investment before a payoff can be expected. Furthermore, they lack to explicitly relate the development activity of a reuse artifact with the project contexts in which experiences with the artifact are actually gained. Thus, valuable information might be lost and also multiple superfluous development activities might be initiated. Additionally, quality of the artifacts is only assessed by the developers of the artifacts themselves. That is, we have no objective evidence for the necessity or adequacy of an artifact

These observations also hold for our former development practice of SDL patterns. We therefore suggest an improvement cycle, which is embedded into the normal software development process, so that reusable knowledge can be systematically collected and continuously packaged into reusable SDL patterns. In order to guide the engineer's improvement activities, a measurement program is developed which is executed in parallel to the development projects. The measurement program is based on the description model of the artifacts and also takes their maturity levels (see next paragraph) into account. *By the means of measurement we objectively evaluate reuse benefits and systematically identify potential of improvement.*

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: a short description of one or more examples 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 UML object model. This defines the involved design elements and their relations.
Message scenario: example scenarios illustrating typical interactions between the objects involved in the design solution are specified by using *MSC diagrams.*
SDL fragment: the mere syntactical part of the design solution is defined by a generic SDL fragment representing the context invariant parts of the solution. If more than one SDL version of the design solution is possible (e.g., interaction by message passing or shared variables), fragments for the most frequent versions are included. The fragments must be adapted and textually embedded into the context specification when applying the pattern. For each fragment, this is prescribed in terms of *syntactical embedding rules*, which, e.g., guide renaming of generic identifiers or specialization of embedding design elements. Note that an SDL fragment is generally not a syntactically complete SDL specification. From a semantic point of view, SDL fragments therefore only become formally meaningful when they are part of a complete SDL specification.
Semantic properties: the formal basis provided by SDL allows to specify semantic properties of a pattern expressed in an assumption/commitment style. That means, if the *assumptions* on the embedding context hold, applying the pattern results in the specified *commitments*. Normally, for verifying the assumptions, a tool supported verification technique based on state space exploration such as model checking is needed. In order to simplify verification, an assumption can be complemented with one or more *sufficient conditions* which imply the considered assumption and which can be analyzed statically. For instance, instead of claiming that a signal should never be consumed by another transition of the same process, it is sufficient to ensure that the applied signal is unique within that process. This can be verified statically when applying the pattern.
Refinement: an embedded pattern instance can be further refined, e.g., by the embedding of another pattern instance in subsequent development steps. Care has to be taken not to destroy a pattern's intent by such refinement steps. Therefore, refinements compatible with the pattern's intent are specified.
Cooperative usage: possible usage with other patterns is described. This links the pattern with other patterns of the repository and distinguishes our SDL pattern repository from a mere pattern catalogue where the patterns are unrelated or only loosely related.

Table 1: SDL pattern description template

Maturity Levels of SDL Patterns

In a first step, we introduce different maturity levels for SDL patterns that shall allow the engineers to participate in the improvement cycle, while reusing premature artifacts. It can generally be stated that the effort in reusing an artifact decreases with higher maturity levels. As long as the quality is low, the engineer is mainly forced to develop a solution from scratch, but she may add value to the reuse artifact by collecting experience and providing suggestions for improvement as well as by justifying the necessity and adequacy of the pattern to some extent, as it has just turned out to be valuable

For defining the maturity levles of an SDL pattern, we take validation status and completeness of pattern description

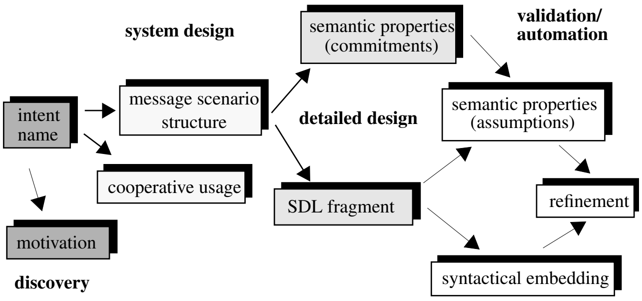

Figure 1: Development steps of an SDL pattern

into account. In Figure 1, we classify the description items of the SDL pattern description template (Table 1) according to the development phases they support. Furthermore, we illustrate the partial order between them which describes dependences on completing/developing an SDL pattern specification. An arrow indicates that a description item is completed before its corresponding successor(s). This results in four maturity levels, which are enriched by further validation criteria in the following definitions:

- <u>Informal</u>: the items *name*, *intent*, and *motivation* are completed. The pattern event (some developer has witnessed the application of the potential pattern) has occurred at least three times.
- <u>Architectural</u>: the items *structure*, *message scenario*, and *cooperative usage* are completed. The relation (new, specialization, cooperation, change, ...) of the pattern to other members of the pattern pool has been determined, so that the necessity of the pattern is confirmed. It has been validated that the pattern architecture meets the pattern's intent. The architecture conforms to the pattern events occurred so far.
- <u>Behavioral</u>: the items *SDL fragment* and *semantic properties (commitments)* are completed. The correspondence between UML diagram and SDL specification is checked. It has been validated that the specified pattern behavior implements the message scenarios and the pattern's intent. The behavior conforms to the pattern events occurred so far.
- <u>Automatic</u>: the items *syntactical embedding*, *semantic properties (assumptions)*, and *refinement* are completed. They are incorporated into the SDL pattern editor [3] and tested for the pattern events occurred so far.

Note that the maturity levels do not prescribe a strict development order of the different items. Rather, the order of completion is expressed. As the witnesses of a pattern event must develop a solution for the corresponding design problem anyway, it is advantageous to partially elaborate the items beyond the current maturity level (Figure 1). Thus experience from former projects is already provided according to the description model of SDL patterns and packaging is done easier when the maturity level will rise later.

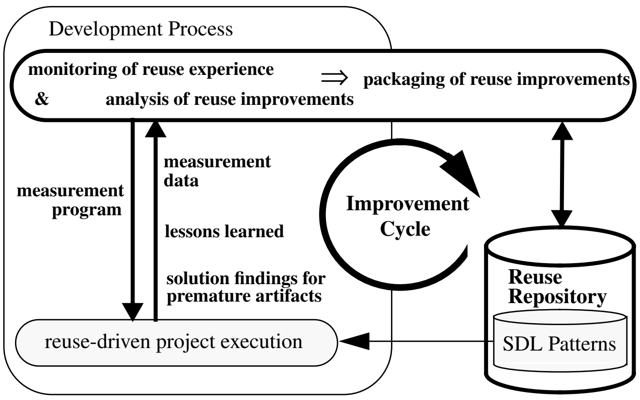

Figure 2: Interaction of reuse process and improvement cycle

Measurement Program

By measurement we aim at evaluating the current reuse practice as well as systematically identifying potential of improvements. One measurement approach that allows to realize these goals is the GQM paradigm [1]. We applied this approach to define several evaluation and improvement goals and subsequently refine them into more detailed questions and metrics. For instance, we developed measurement plans for monitoring design errors and reuse effort [4] that enable the evaluation of benefits when compared to historical data. Evaluation plans are directly related to the classical SE goals of reducing development costs and increasing product quality. However, these direct measures give only few hints for improving a specific SDL pattern. With the assumption that the SDL-pattern approach works on principle, we can derive measurement plans from the description model, that better help to continuously improve the approach. For instance, if the pattern maturity level is automatic and the static analyser [3] could not verify some assumption on the pattern application, the developer is asked for a (statically checkable) sufficient condition that might be used. As the developer must validate the pattern application anyway, this does not cause any extra effort.

Figure 2 illustrates the improvement cycle (bold lines) and its interaction with the reuse process (thin lines). The suggested improvements need to be related to experiences from other application projects. This is done in a subsequent packaging step, so that the improved artifact can finally be stored in the reuse repository. The reuse repository plays a central role in the improvement cycle, because it enables the transfer of reuse experience between different project contexts, which is decisive for a successful packaging activity.

Reuse Repository

The described improvement cycle for SDL based system development not only suggests systematic improvement and finding of SDL patterns, but also a comprehensive reuse of other artifacts involved in the cycle (Table 2). Since all artifacts represent experience, we call them "*experience elements*".

project phase		experience elements
reuse driven	planning	process models, measurement goals & plans, questionnaires, lessons learned
	execution	SDL patterns, (SDL frameworks, SDL library components,) lessons learned, process models
improvement driven	analysis	all experience elements used in the planning and execution phase of the project
	packaging	templates to store newly gained experience, all experience elements that have been used in the project and need to be updated

Table 2: Experience elements of different project phases

To allow an effective storage, search, and retrieval of the experience elements in the repository, a unique representation of similar experience is regarded. Predefined templates and/or style guides have to be offered when adding new experience into the repository. For example, all SDL patterns are represented according to a predefined structure (Table 1). To further support an efficient search and retrieval, we clearly structure the repository into so called "*areas*" that are utterly disjunct. Each area can be seen as a class that holds instances of the same type of experience elements. All the experience elements listed in Table 2 can be stored exactly within one of the areas and are interrelated by bidirectional relations, such as *used_in/uses*. With the help of this simple structure a person wanting to add or retrieve a certain kind of experience element get's a first guidance where to search or place it in the reuse repository.

But this is not the only possible support for the improvement cycle. Furthermore, it would be useful to additionally capture the characteristics of a project, and support easy access to its whole documentation in the analysis and packaging phases. Therefore, our reuse repository is extended by a project database, in which pattern related information concerning single projects is stored. This is organized by a template, whose structure is defined in accordance with our improvement and reuse cycle. For instance, the measurement plan, the applied SDL process model, a list of reused SDL patterns as well as the collected measurement data and newly gained lessons learned are included. For each project a copy of the template is created and filled out while the project is conducted. With the help of this project documentation, we can exactly tell what worked out well and should be packaged for reuse in other projects.

In our intra-net based implementation of the repository, we represent experience elements with the help of so called "*context vectors*". A context vector consists of three parts. The first part holds references (HTML-Links) to the different representation forms of the experience element. In the second part, a list of existing relations to other experience elements is given (again, realized by HTML-Links). The last part consists of a number of attribute-value-pairs (e.g., <application domain, real-time systems>, <design language, SDL>). With the help of these attribute-value-pairs it is possible to describe and characterize an experience element, independently from the format in which it is stored in the repository. All context vectors together form a complex network that is the basis for accessing and browsing the repository by simply following the offered links.

4 CURRENT EXPERIENCE

We have applied the improvement cycle to several SDL projects and summarize some representative results subsequently. All projects were conducted by different teams of about 2-4 members, which consisted of students and researchers of the Computer Science Department at the University of Kaiserslautern.

(Re-)engineering of SDL Patterns

An SDL pattern, called *BlockingRequestReply* [6], was originally discovered before we applied the improvement cycle. According to our taxonomy it had to be classified as „architectural". After applying the pattern in two further projects the improvement cycle identified three syntactical variants of the SDL fragment and discovered two specializations of the whole pattern. As the pattern could finally be viewed as stabilized, it has reached level „behavioral" and is now being implemented with our SDL pattern editor.

Application of the improvement cycle to another SDL pattern, called *DynamicEntitySet* [6], even revealed that the pattern had only reached level „informal" before. Through practical experience some generalizations turned out that made it a very powerful pattern to expect (it is now at level „architectural").

Evaluation of the SDL Pattern Approach

It turned out that an adequate knowledge of pattern-based design in general is of great importance for the success of a project. This made us reconsider the quality of our training material. Furthermore, experience has shown that decomposition and analysis (these are specific activities in the accompanying reuse process of SDL patterns) 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 reason.

Regarding the number and distribution of design errors, one case study [4] was performed. Thereby it turned out that design errors are reduced by the application of SDL patterns, because the only errors uncovered resulted from ad hoc design steps that did not apply any patterns.

5 CONCLUSION

We have presented an approach for the systematic evolution of SDL patterns. The approach is based on maturity levels for the reuse artifacts and directly exploits practical reuse experience and applies a measurement program for systematic identification of possible improvements. To transfer reuse experience between development projects a specific reuse repository is defined that stores and interrelates experience elements that go far beyond reusable SDL patterns. The approach offers several advantages compared to ad hoc learning cycles:

- Incremental investment in reuse that is triggered by software developers on request.
- Quality increases, as good artifacts depend on practical and long-term experience and several engineers contribute in quality improvement over time.
- Adaptation of the reuse repository to an application domain, i.e., domain specific artifacts are developed from universal artifacts that are contained in a basic pool.
- An evaluation of reuse benefits can easily be integrated into the measurement program.

We did not describe implementation details of our reuse repository so far. However, we have developed an Internet based implementation for distributed, collaborative usage. It is planned to provide part of the SDL pattern pool on the Internet in order to get broader feedback for our improvement cycle from the SDL community [12].

REFERENCES

1. V. R. Basili, G. Caldiera, H. D. Rombach, *Goal Question Metric Paradigm*, in J.J. Marciniak (ed), Encyclopedia of Software Engineering, Volume 1, John Wiley & Sons, 1994
2. G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language, Version 1.1.*, Rational Software Corporation, 1997
3. D. Cisowski, B. Geppert, F. Rößler, M. Schwaiger, *Tool Support for SDL Patterns*, in Proc. of the 1st Workshop of the SDL Forum Society on SDL and MSC (SAM'98), Germany, 1998
4. R. L. Feldmann, B. Geppert, F. Rößler, *Towards an Experimental Evaluation of SDL-Pattern based Protocol Design*, SFB 501 Report 04/98, Computer Science Department, University of Kaiserslautern, Germany, 1998
5. E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995
6. B. Geppert, F. Rößler, *Pattern-based Configuring of a Customized Resource Reservation Protocol with SDL*, SFB 501 Report 19/96, Computer Science Department, University of Kaiserslautern, Germany, 1996
7. B. Geppert, F. Rößler, *Generic Engineering of Communication Protocols - Current Experience and Future Issues*, Proc. of the 1st IEEE Int. Conference on Formal Engineering Methods (ICFEM'97), Hiroshima, Japan, 1997
8. ITU-T Recommendation Z.100 (03/93) *CCITT Specification and Description Language (SDL)*, International Telecommunication Union (ITU)
9. ITU-T Recommendation Z.120 (10/96) *Message Sequence Chart (MSC)*, International Telecommunication Union (ITU)
10. R. Reed, *Methodology for Real Time Systems*, Computer Networks and ISDN Systems 28 (1996)
11. F. Rößler, B. Geppert, P. Schaible, *Re-Engineering of the Internet Stream Protocol ST2+ with Formalized Design Patterns*, in Proc. of the 5th IEEE Int. Conference on Software Reuse (ICSR5), Victoria, British Columbia, Canada, 1998
12. SDL Forum Society, <http://www.sdl-forum.org/>
13. Telelogic, *SDT 3.3 Methodology Guidelines – Part1: The SOMT Method*, Telelogic, Sweden, 1998
14. *TIMe : The Integrated Method*, <http://manon.informatics.sintef.no/time/>.