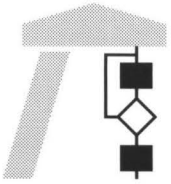
TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

INTERNER BERICHT

C. Webel, I. Fliege

SDL Design Patterns and Components – Watchdog and Heartbeat

Technical Report 335/04



FACHBEREICH
INFORMATIK

SDL Design Patterns and Components – Watchdog and Heartbeat

C. Webel[1], I. Fliege[2]

[1]Computer Science Department, University of Kaiserslautern, Kaiserslautern, Germany
webel@informatik.uni-kl.de

[2]Computer Science Department, University of Kaiserslautern, Kaiserslautern, Germany
fliege@informatik.uni-kl.de

Technical Report 335/04

Computer Science Department
University of Kaiserslautern
Postfach 3049
67653 Kaiserslautern
Germany

SDL Design Patterns and Components Watchdog and Heartbeat

Christian Webel, Ingmar Fliege

Computer Science Department, University of Kaiserslautern
Postfach 3049, D-67653 Kaiserslautern, Germany
{webel,fliege}@informatik.uni-kl.de

Abstract. We present a methodology to augment system safety step-by-step and illustrate the approach by the definition of reusable solutions for the detection of fail-silent nodes – a watchdog and a heartbeat. These solutions can be added to real-time system designs, to protect against certain types of system failures. We use SDL as a system design language for the development of distributed systems, including real-time systems.

1 Introduction

Safety-critical systems are required to exhibit extremely low rates of critical failures. A variety of reliability measures such as hardware redundancy, time redundancy, software diversity, or recovery can be combined. Also, the functionality of a system may be reduced as a final consequence of a failure by guiding the system to a fail-operational (e.g., aircraft) or fail-safe (e.g., train) state. We define SDL design patterns and micro protocols for the detection of fail-silent nodes and to move the controlled system into a *fail-safe* or *fail-operational* state. Therefore new functionality must be added to realize those new requirements.
We have identified two different solutions. The first solution introduces the new functionality by refining the given components using existing design patterns WATCHDOG and HEARTBEAT, the second by adding available micro protocols to the system.
Both approaches support reusability and thus augment both design quality and productivity. One important difference is the architectural impact. SDL micro protocols are self-contained components that extend a system on the architectural level, without modifying the behaviour of context components. SDL design patterns can be applied on architectural level, too, but may also modify the behaviour of existing components, which requires detailed engineering knowledge. When applying patterns, the developer may also adapt functionalities and optimize certain system aspects.
In this report, we will introduce SDL design solutions to augment reliablity found in safety-critical systems in a systematical way, with the purpose of reusing them in the incremental development of safety-critical systems. We define two SDL design patterns and two micro protocols for

the detection of fail-silent nodes.

2 SDL design patterns for reliable systems

2.1. SDL design patterns

Design patterns [4] are a well-known approach for the reuse of design decisions. In [1], another specialization of the design pattern concept for the development of distributed systems and communication protocols, called *SDL design patterns*, has been introduced. SDL design patterns combine the traditional advantages of design patterns – reduced development effort, quality improvements, and orthogonal documentation – with the precision of a formal design language for pattern definition and pattern application.

The SDL design pattern approach [6,8] consists of a *pattern-based design process*, a notation for the description of generic SDL fragments called *PA-SDL* (Pattern Annotated SDL), a *template* and *rules* for the definition of SDL design patterns, and an *SDL design pattern pool*. The approach has been applied successfully to the engineering and reengineering of several distributed applications and communication protocols, including the SILICON case study [8], the Internet Stream Protocol ST2+ [13], and a quality-of-service management and application functionality for CAN (Controller Area Network) [5]. Applications in industry, e.g., in UMTS Radio Network Controller call processing development, are in progress [10].

An *SDL design pattern* [4,6] is a reusable software artifact that represents a generic solution for a recurring design problem with *SDL* [12] as design language. Over a period of more than 25 years, SDL (System Design Language) has matured from a simple graphical notation for describing a set of asynchronously communicating finite state machines to a sophisticated specification technique with graphical syntax, data type constructs, structuring mechanisms, object-oriented features, support for reuse, companion notations, tool environments, and a formal semantics. These language features and the availability of excellent commercial tool environments are the primary reasons why SDL is one of the few FDTs that are widely used in industry.

When SDL patterns are applied, they are selected from a pattern pool, adapted and composed into an embedding context. The pattern pool can be seen as a repository of experience from previous projects that has been analyzed and packaged. The SDL patterns we have identified so far can be classified into five categories:

- *Architecture patterns* capture generic architectures and their refinements.

  Example: CLIENTSERVER [10]. This pattern captures a client/server architecture of a distributed system.

- *Interaction patterns* capture the interaction among peers, e.g., a set of application agents or service users.

  Example: SYNCHRONOUSINQUIRY [9]. This pattern introduces a confirmed interaction between two peers. After a trigger from the embedding context, an agent sends an inquiry and is blocked until receiving a response from the second agent.

- *Control patterns* deal with the detection and handling of errors that may result from loss, delay, or corruption of messages, or from agent failures.

  Example: LOSSCONTROL [9]. This pattern provides a generic solution for the detection and handling of message loss in the case of confirmed interactions, such as synchronous

inquiries. If a response does not arrive before the expiry of a timer, the message is repeated (Positive Acknowledgement with Retransmission).

- *Management patterns* deal with local management issues, such as buffer creation or message addressing[1].

  Example: BUFFERMANAGEMENT [11]. When a signal is passed between two local processes, the signal parameters are stored into a buffer, and a buffer reference is sent. This technique has an impact on implementation efficiency, it reduces memory consumption and copying overhead. The pattern addresses the correct buffer management such that memory leaks, for instance, are avoided.

- *Interfacing patterns* replace the interaction between peers by interaction through a basic service provider. This may include segmentation and reassembly, lower layer connection management, and routing.

<NAME>
Each pattern is identified by a pattern name, which serves as a handle to describe a design problem, its solution and its consequences.

Version: The version of the pattern.

Intent: Provides a short informal description of the design problem and its solution.

Motivation: Example for the patterns usage without relying on the patterns definition.

Structure: Graphical representation of the involved (design) components and their relations (before and after applying the pattern).

Message Scenario: Typical behaviour related to this pattern. Complements the structural aspects.

SDL Fragment: Syntactical part of the design solution in PA-SDL (Pattern Annotated SDL). This part is adapted and composed when the pattern is applied. It defined the context, in which the patter nis applicable, the permitted adaptions, and the embedding into the context specification.

Syntactical Embedding Rules: The S.E.R. constrain the application of the pattern such that certain desirable properties are added or preserved.

Example Application: Illustration of the patterns application.

Semantic Properties: Results from the correct application of the pattern.

Refinement: Remarks for further redefining an applied pattern in accordance with the patterns instance.

Cooperative Usage: Usage together with other patterns.

Known Uses: Documentation of the use of the pattern.

Checklist: Used during a design review to prevent common errors. This list may be extended based on problems occuring during usage of the pattern.

Figure 1: SDL Design Pattern Template

---

1. It can be argued that these management patterns are rather low-level, as compared to the other examples. However, they have been discovered in an industrial cooperation, and capture realistic design decisions that lead to the generation of more efficient code. Furthermore, application of these patterns signficantly reduces the number of design errors [11].

Example: CODEX [2]. This pattern provides a generic solution for encoding service data units (SDUs) and interface control information into protocol data units, the exchange of PDUs among specific protocol entities, and the decoding and forwarding of SDUs.

The definition of SDL design patterns supports their selection during the protocol design. As the result of the object-oriented analysis of requirements, an analysis model consisting of a UML object diagram and MSC message scenarios are built. Comparing the structure and the message scenarios of SDL design patterns against this analysis model strongly supports the selection of suitable patterns [9]. As the number of patterns in a typical pattern pool (see also [4]) is relatively small (10-30 patterns[2]), and with additional information contained in the pattern pool, for instance, on cooperative usage, this should be sufficient for a proper selection. The complete template for a pattern's description is shown in Figure 1.

2.2. The design pattern *Watchdog*

The *Watchdog* pattern realizes the safety functionality called watchdog and belongs to the category of *Interaction patterns*. It describes a behaviour, that extends a given system.

The MSC in Figure 2 shows an example where the described design problem arises, which is solved by the suggested solution:

In an automatic safety device on trains (dead man's control), an operator has to press a button periodically within a prior well defined time interval. When the operator desists from pressing the button, the automatic safety device assumes the operator is dead and stops the train, which leads the system to a *fail-safe* state in order to prevent a catastrophe, e.g., a crash at an unmanned crossing.

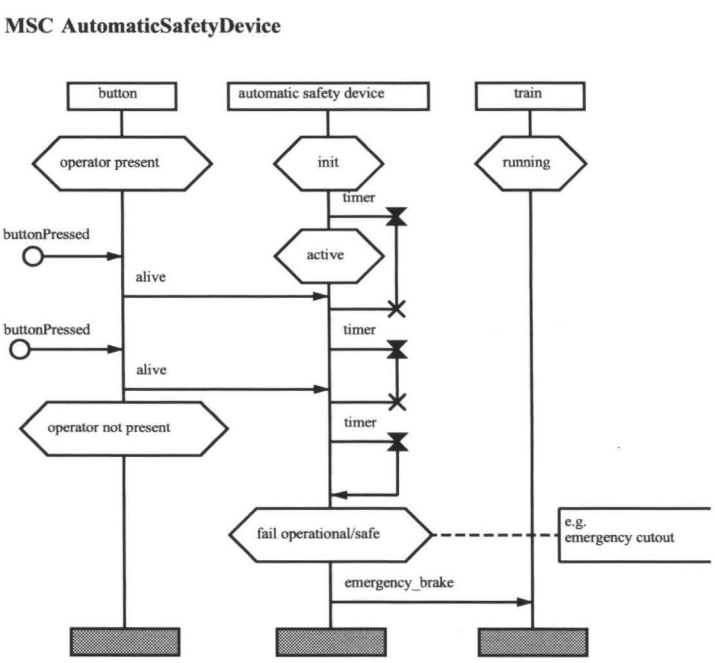


Figure 2: MSC automatic safety device

Figure 3 shows the graphical representation of the structural aspects of the pattern's solution. Note that *Watchdog* either refines a component from the context or is added as a new component to the structure:

- *Trigger* is a component of the context, which provides an alive signal periodically.
- *Controller* is a component where watchdog functionality is to be added. The WATCHDOG pattern can either refine a component of the system or augment the system with a new component.

2. These figures result from practical experience. They differ substantially from the size of typical component repositories with 100s of elements. The relatively small number can be explained by the generic nature of patterns. Also, as the definition of "good" patterns is a substantial investment, only those patterns that are frequently applied should be included in the pattern pool.

- *Controlled System* is the part of the system which is moved into a safe state, when life-sign fails to arrive.

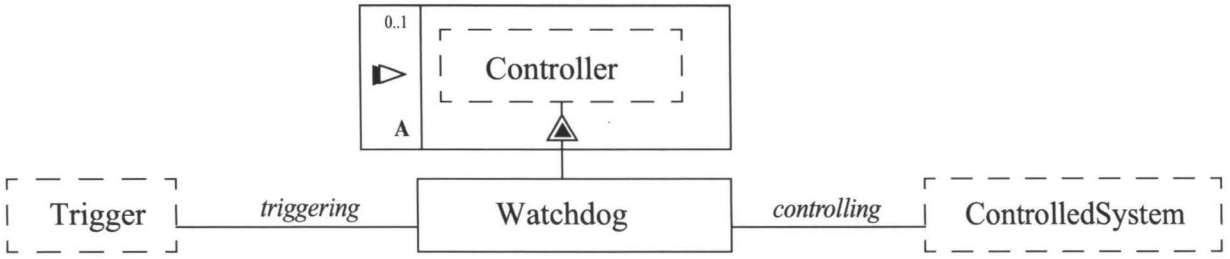


Figure 3: Structure of the design solution - WATCHDOG

The *SDL-Description* of the WATCHDOG pattern is shown in Figure 4. It describes the syntactical part of the suggested design solution, which is adapted and composed when the pattern is applied. The extended finite state machine *Watchdog* that optionally refines *Controller* describes the watchdog functionality. The timer *watchdogT* is set for a duration of *hbiW* when triggered by a certain input from the context and restarted after a trigger (Figure 4 left). The trigger showing that the system is still alive can be one or more inputs or continuous signals. The duration *hbiW* is the timeout interval after which the system changes to a *fail-safe/fail-operational* state. This is done by sending one or more control signals to the controlled system (Figure 4 centre). Disabling the *watchdog* is also possible (Figure 4 right).

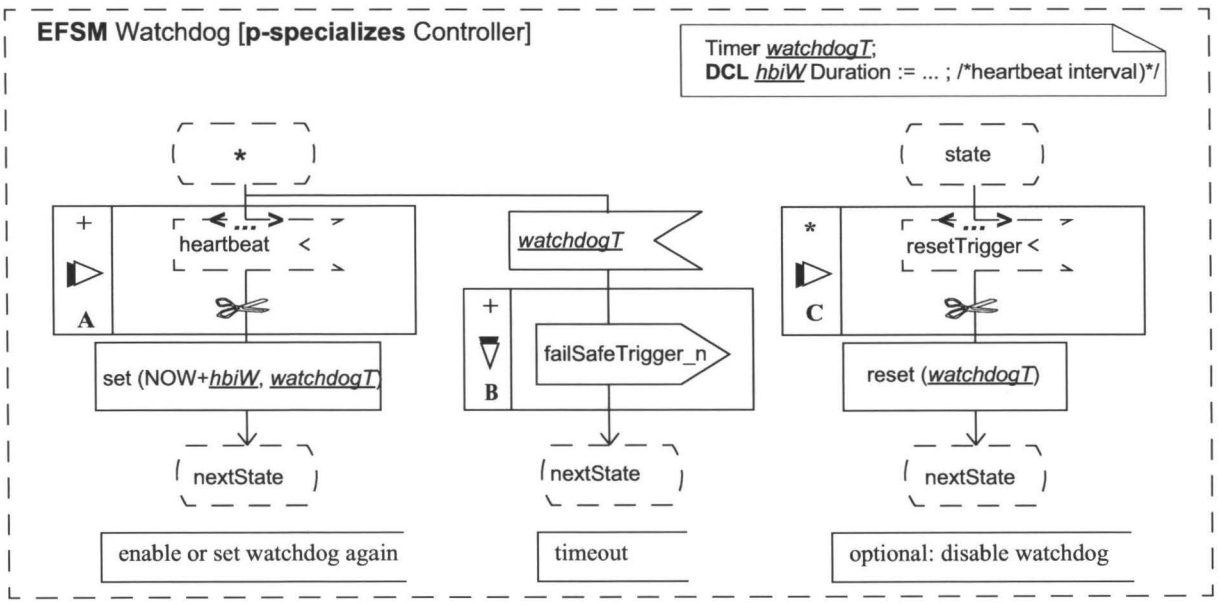


Figure 4: SDL design pattern WATCHDOG: SDL Fragment

2.3. The design pattern *Heartbeat*

The watchdog functionality assumes a periodic trigger in order to prevent the watchdog from sending control signals. If the system does not provide a periodic communication with an adequate interval, the *Heartbeat* pattern can be applied. This augments the system behaviour with the *heartbeat* that is periodically sent.

The following figure shows the graphical representation of the structural aspects of the patterns solution. Note that *Heartbeat* either refines a component from the context or is added as a new component to the structure:

- *Controller* is refined by *Heartbeat* and describes the system that has to be observed.
- *Watchdog* is a component realizing watchdog functionality as described in Section 2.2.

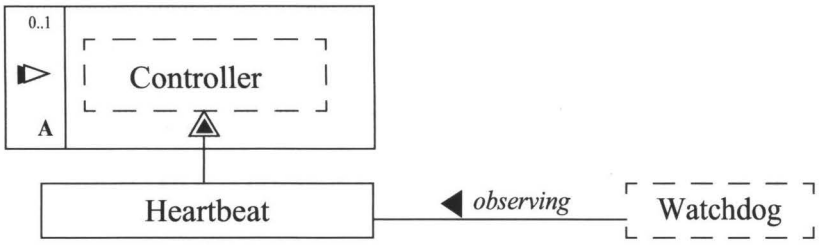


Figure 5: Structure of the design solution - HEARTBEAT

Figure 6 shows the SDL-description of the HEARTBEAT pattern. The EFSM *Heartbeat* describes the heartbeat functionality. After the initialization or an input signal (Figure 6, centre), the timer *heartbeatT* is set to the duration of *hbiA* and after the timeout one or more *heartbeat_n* signal is generated and propagated. Therefore *Controller* is refined by adding transitions to start and stop the heartbeat (optionally) and one to handle the *heartbeatT* by sending the *heartbeat_n*. The *heartbeat_n* has to be consumed by a corresponding component which realizes watchdog functionality. This signal can also be an existing signal in the system that can be used for a heartbeat.

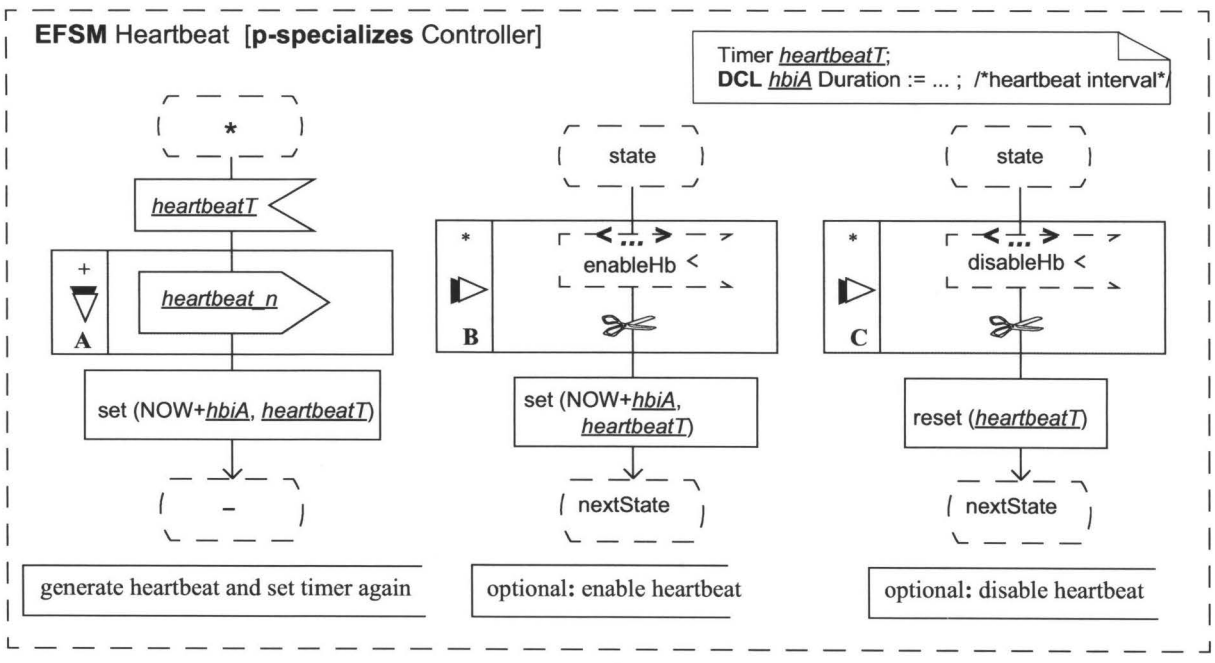


Figure 6: SDL design pattern HEARTBEAT

3 Micro Protocols for reliable systems

3.1. SDL micro protocols

In [7], the structuring unit *micro protocol*, i.e. a communication protocol with a single (distributed) functionality and the required protocol collaboration, has been identified and applied to SDL designs. A functionality (e.g., flow control, loss control, QoS monitoring) is a single aspect of internal system behavior that may be distributed among a set of system agents, with causality relationships between single events. In [3], a fine-grained development process, together with a generic micro protocol framework, is presented. From a reuse viewpoint, micro protocols classify as design components, which are selected from a library and composed. Obviously, there are several ways to represent them in SDL, for instance, by specifying SDL block types, SDL process types, SDL service types, or SDL procedures. Which one to use depends on the composition of micro protocols, which in turn depends on the protocol that is to be configured.

Micro protocol definitions are organized using SDL packages. An SDL package is a collection of type definitions, and is used here to encapsulate SDL types belonging to the same micro protocol. This way, a micro protocol library can be expressed as a set of SDL packages, i.e., ready-to-use components. Also, common parts of a set of micro protocols may be extracted into a package that is imported by each micro protocol definition. Alternatively, several related micro protocols may be grouped into one package.

3.2. The micro protocol *Watchdog*

The micro protocol *Watchdog* is encapsulated in one single process type (Figure 7) and may be specialized to match the requirements of the embedding context. A timer *watchdogT* is used to monitor receipt of an *alive* signal from the context within a well-defined interval. This *safeInterval* is initially defined, but can be modified by redefining the virtual start transition. When *Watchdog* does not receive an *alive* signal within this given period, the timer *watchdogT* triggers a transition to send a signal to the context (controlled system). Again, this signal must be specified by redefining a virtual transition. Optionally, a signal may be send when the watchdog assumes the observed system to be dead and

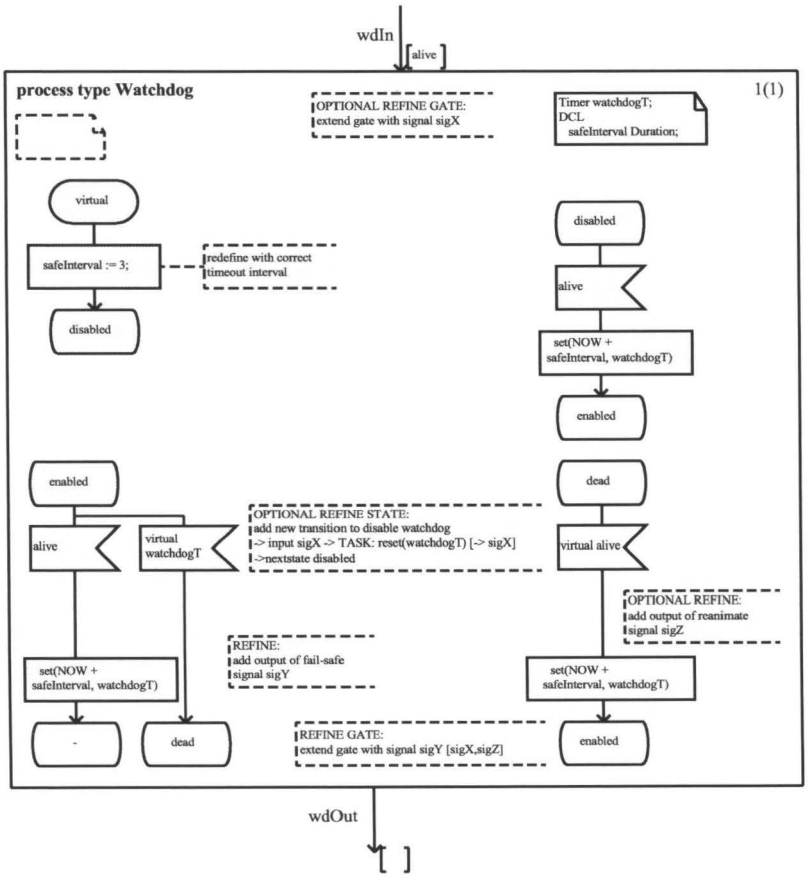


Figure 7: Micro protocol *Watchdog*

an *alive* signal reappears.
In order to provide a periodic *alive* trigger, another micro protocol *Heartbeat* can be used.

3.3. The micro protocol *Heartbeat*

The micro protocol *Heartbeat* is used to provide a system with a periodic sending of an *alive* signal. This signal is used to trigger the micro protocol *Watchdog* showing that the observed system is still alive.

The behaviour is encapsulated in one single process type shown in Figure 8. The predefined *heartbeatInterval* in which signals are sent should be adapted to fit the requirements of the watchdog observing this system. This is done by refining the start transition.

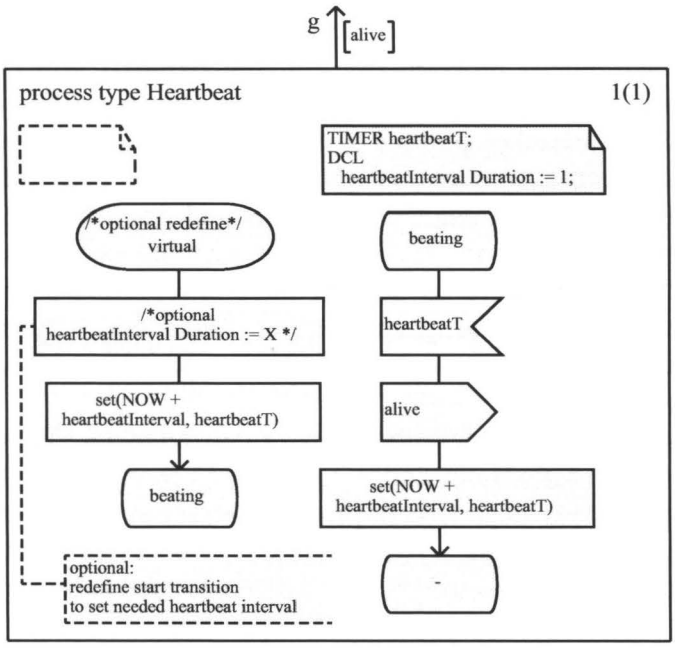


Figure 8: Micro protocol *Heartbeat*

WATCHDOG

Version 1.3

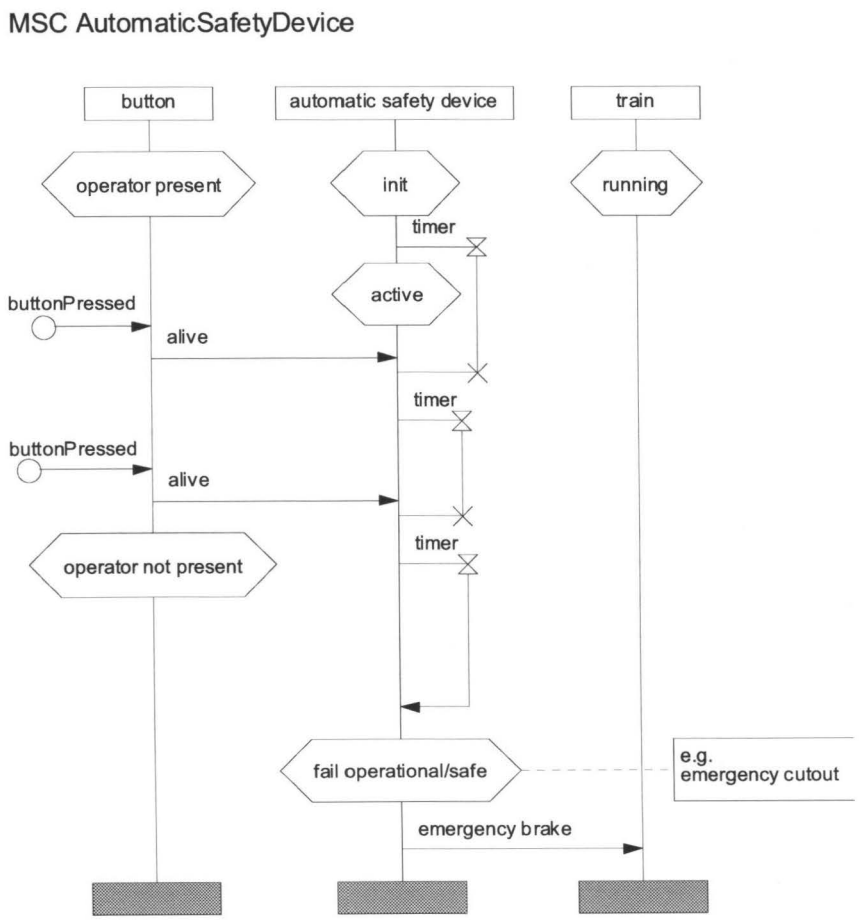
Intent:

The WATCHDOG pattern realizes a safety functionality generally known as *watchdog*. A *watchdog* is a component or functionality monitoring the operation of a system by observing an *alive*-signal. If this signalfails, a fail operational or fail safe state has to be reached.

Motivation:

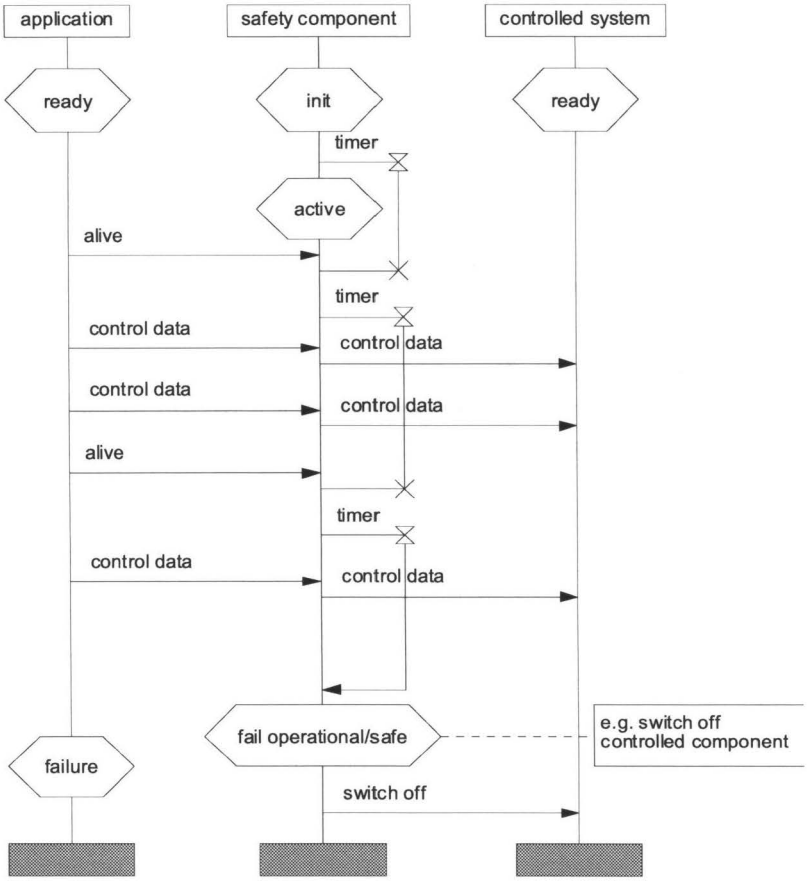
Here are some examples where the described design problem arises, which can be solved by the suggested solution.

- **Automatic safety device (dead man's control):**
  An operator (e.g. a train conducter) has to activate periodically a button or switch in accordance with a prior well- defined time interval.



- **Application with safety aspects:**
  An application periodically sends an alive signal to a safety component to propagate its running state. If this signal fails to appear, the safety component has to switch to a *fail-operational* or *fail-safe* state (e.g. switch off controlled system).

MSC Application with safety aspects

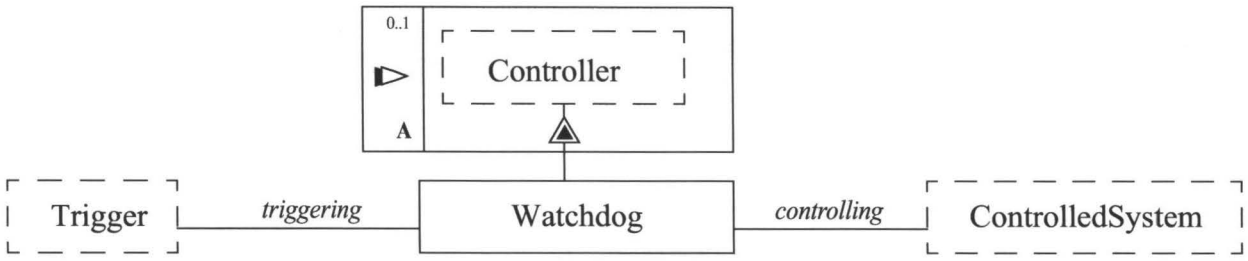


Structure:

The following shows the graphical representation of the strutural aspects of the pattern's solution. Note that *Watchdog* not necessarily has to refine a component from the context. It is also possible to newly add this component to the structure.

Trigger is a component from the context, which provides a peridocal *alive* signal. It can be a button or a switch (environment) or any kind of system/component.

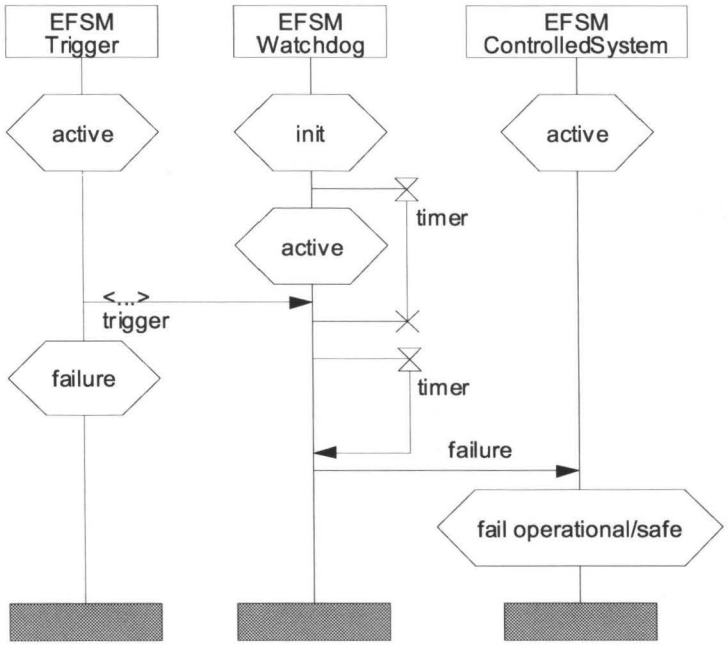
Controller is, where necessary, refined by *Watchdog* and can be, for example, some control system where watchdog functionality should be added.



Message Scenario:

The following shows a typical generic usage scenario between the components described above.

MSC Watchdog



SDL Fragment:

After being triggered, *Watchdog* resets its timer *watchdogT* to a given period and continues working. After receiving a timeout, *Watchdog* has to ensure, that *ControlledSystem* reaches a safe state (e.g. by sending one ore more control signals). If *heartbeat* becomes active again (after a failure), *Watchdog* has to set its timer to a given period in order to resume controlling.

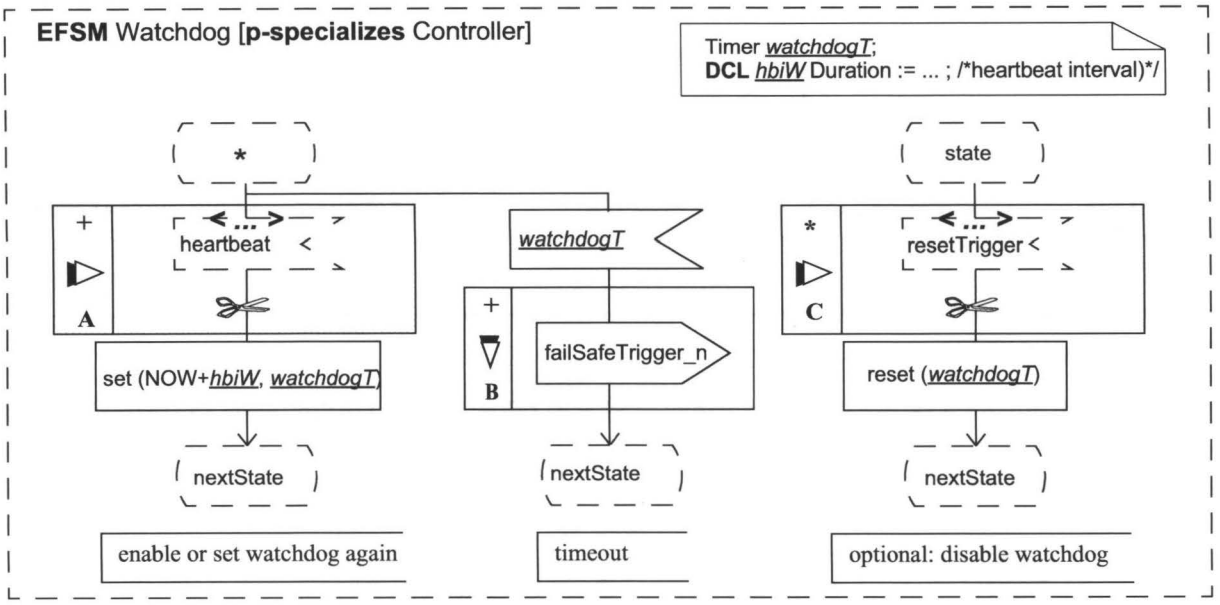


Figure 1: SDL design pattern WATCHDOG: SDL Fragment

Syntactical Embedding Rules:

In the following, we describe how to instantiate the considered SDL fragments.

- **Watchdog:**
  - Variants:
    - Resolve the trigger symbols - this could result in an input symbol, conditioned input, continuous signal, or an input symbol followed by a condition symbol.
    - If no component *Controller* is refined (thus *Watchdog* is added to the structure) all symbols and states from the context have to be solved and added.
  - Renaming:
    - If refinement is provided, the state (set) *state* and the signals/variables *resetTrigger*, *failSafeTrigger_n* and *heartbeat* are set to the states/signals/variables of the embedding context.
    - If no refinement is provided, the state (set) *state* is to set to a new state, the signal/variable *heartbeat* is set to a signal sent by the component *Trigger* and the signal/variable *resetTrigger* is set to a new signal disabling the watchdog.
    - The signals *failSafeTrigger_n* are to set to suitable names of the embedding context (*ControlledSystem*) to reach a fail-safe/operational state.

Example Application:

see: C. Webel: *Development and Integration of QoS Micro Protocols for Controlling an Airship via WLAN,* Master Thesis, Computer Networks Group, University of Kaiserslautern, Kaiserslautern, Germany, June 2004 (in german)

Semantic Properties:

Under the **Assumption** that ...

(A-1) The state (set) *state* of *Watchdog* will always eventually be reached.
Sufficient condition: *state* is the only state of *Watchdog.*

... the following **Commitment** holds:

(C-1) Every time *resetTrigger* arises, the watchdog is disabled.

(C-2) Every time a timeout arises, the timer *watchdogT* is not reset and suitable signals are sent to *ControlledSystem*.

Refinement:

No Refinement needed.

Cooperative Usage:

HEARTBEAT-Pattern

Known Uses:

C. Webel: *Development and Integration of QoS Micro Protocols for Controlling an Airship via WLAN,* Master Thesis, Computer Networks Group, University of Kaiserslautern, Kaiserslautern, Germany, June 2004 (in german)

Checklist

- **Watchdog:**
  - After receiving a trigger, *watchdogT* is set.
  - After receiving a timeout, *watchdogT* is not set.
  - For every possible *heartbeat* a transition has to be added.
  - For every possible *resetTrigger* a transition has to be added.
  - *watchdogT* is set to a suitable value.
  - *failSafeTrigger_n* is always sent after a timeout.
  - the signals *failSafeTrigger_n* lead to a fail-safe/operational state of the controlled component.

HEARTBEAT

Version 1.0

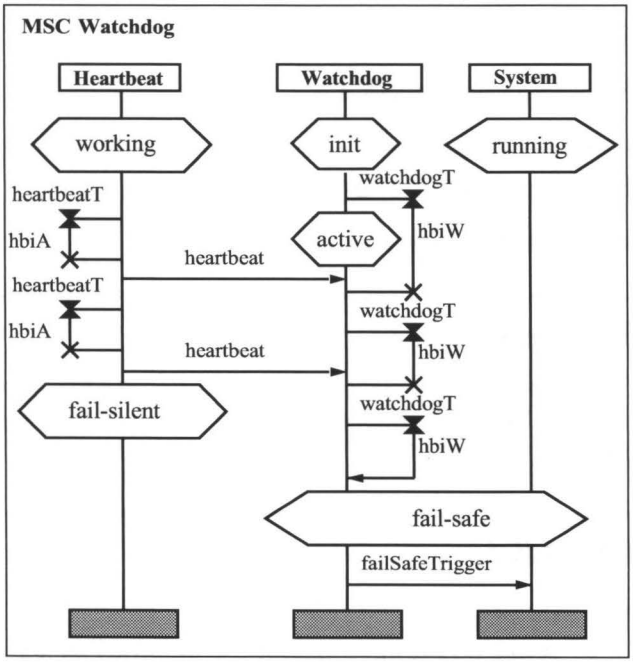
Intent:

The HEARTBEAT pattern realizes a functionality generally known as *heartbeat*.

Motivation:

Here are some examples where the described design problem arises, which can be solved by the suggested solution.

- **Watchdog:**
  A *heartbeat* is needed in order to send a periodic *heartbeat* signal to provide the watchdog and the system to switch to fail-safe/fail-operational state.

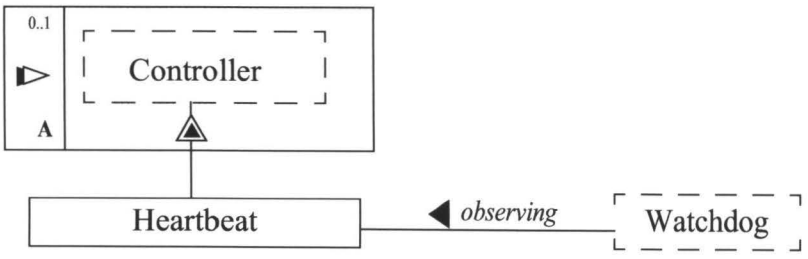


Structure:

The following shows the graphical representation of the strutural aspects of the pattern's solution. Note that *Heartbeat* not necessarily has to refine a component from the context. It is also possible to newly add this component to the structure.

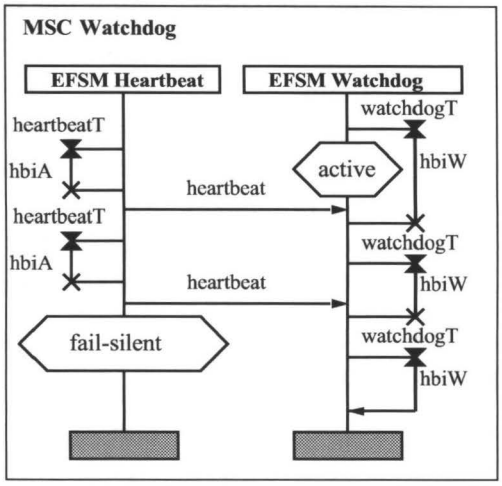
Controller is a component from the context, which has to be monitored. Therefore this component has to sent a periodic signal.

*W*atchdog is a watchdog component



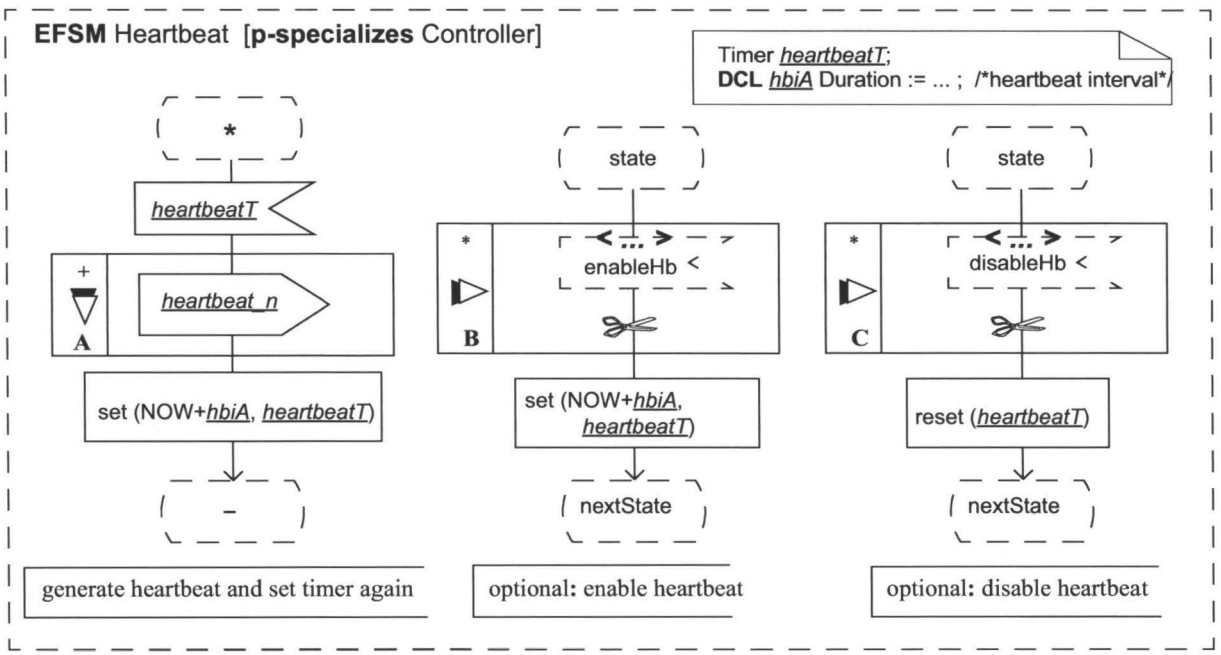
Message Scenario:

The following shows a typical generic usage scenario between the components described above.



SDL Fragment:

After being triggered by signal *enableHb*, *Heartbeat* sets its timer to a given period. After receiving a timeout, one or more signals *heartbeat_n* are sent. The generic trigger *disableHb* disables the timer *heartbeatT* and thus the component *heartbeat*.



Syntactical Embedding Rules:

In the following, we describe how to instantiate the considered SDL fragments.

- **Heartbeat:**
  - Variants:

  - Resolve the trigger symbols - this could result in an input symbol, conditioned input, continuous signal, or an input symbol followed by a condition symbol.
  - If no component *Controller* is refined (thus *Heartbeat* is added to the structure) all symbols and states from the context have to be solved and added.
- Renaming:
  - If refinement is provided, the state (set) *state* and the signals/variables *enableHb* and *disbleHb* are set to the states/signals/variables of the embedding context.
  - If no refinement is provided, the state (set) *state* is to set to a new state, the signal/variables *enableHb* and *disbableHb* are set to new signals send by the component *Controller* and the signal/variable *heartebat* is set to a new signal.

Example Application:

–

Semantic Properties:

Under the **Assumption** that ...

(A-1) The state (set) *state* of *Heartebat* will always eventually be reached.
Sufficient condition: *state* is the only state of *Heartbeat.*

... the following **Commitment** holds:

(C-1) Every time *disableHb* arises, *heartbeatT* is reset.

(C-2) Every time *enableHb* arises, *heartbeatT* is set to the default heartbeat interval.

(C-3) Every time a timeout arises, the timer *heartbeatT* is reset and a new heartebat signal is sent to *Watchdog*.

Refinement:

No Refinement needed.

Cooperative Usage:

WATCHDOG-Pattern

Known Uses:

–

Checklist

- **Heartbeat:**
  - After receiving a timeout, *heartbeatT* is set and one or more heartbeat signals are generated.
  - For every possible *enableHb* a transition has to be added.

- For every possible *disableHb* a transition has to be added.
- *heartbeatT* is set to a suitable value.

References

[1] B. Geppert, R. Gotzhein, F. Rößler: *Configuring Communication Protocols Using SDL Patterns,* In Cavalli, A., Sar,a, Q., eds.: SDL'97 - Time For Testing, Proceedings of the 8th SDL Forum, Amsterdam, Elsevier (1997) pp. 523-538

[2] Computer Networks Group: *The SDL Pattern Pool*, Online document, University of Kaiserslautern, Kaiserslautern, Germany, 2002 (available on request)

[3] I. Fliege, A. Geraldy, R. Gotzhein, P. Schaible: *A Flexible Micro Protocol Framework*, in: D. Amyot, A. W. Williams (Eds.), SAM 2004: SDL and MSC, Fourth International Workshop, Ottawa, Canada, June 2-4, 2004, Revised Papers, LNCS 3319, 2004, pp. 231-244

[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, Massachusetts, 1995

[5] B. Geppert, A. Kühlmeyer, F. Rößler, M. Schneider: *SDL-Pattern based Development of a Communication Subsystem for CAN*, in : S. Budkowski, A. Cavalli, E. Najm (eds.), Formal Description Techniques and Protocol Specification, Testing, adn Verification, Proceedings of FORTE/PSTV'99, Kluwer Academic Publishers, Boston, 1998, pp. 197-212

[6] B. Geppert: *The SDL-Pattern Approach - A Reuse-Driven SDL Methodology for Designing Communication Software Systems*, Ph.D. Thesis, University of Kaiserslautern, Germany, 2000

[7] R. Gotzhein, F. Khendek, P. Schaible: *Micro Protocol Design: The SNMP Case Study,* in: E. Sherratt (Ed.), Telecommunications and beyond: The Broader Applicability of SDL and MSC, LNCS 2599, Springer, 2003, pp. 61-73

[8] R. Gotzhein, C. Peper, P. Schaible, J. Thees: *SILICON - System Development for an Interactive LIght CONtrol*, URL: http://vs.informatik.uni-kl.de/activities/silicon/, 2001

[9] R. Gotzhein: *Consolidating and Applying the SDL-Pattern Approach: A Detailed Case Study*, Information and Software Technology, Elsevier Sciences (in print)

[10] R. Grammes, R. Gotzhein, C. Mahr, P. Schaible, H. Schleiffer: *Industrial Application of the SDL-Pattern Approach in UMTS Call Processing Development - Experience and Quantitative Assessment*, 11th SDL Forum (SDL'2003), Stuttgart, Germany, July 1-4, 2003 (accepted for publication)

[11] R. Grammes: *Evaluation and Application of the SDL Pattern Approach*, Master Thesis, Computer Networks Group, University of Kaiserslautern, Kaiserslautern, Germany, February 2003

[12] ITU-T Recommendation Z.100 (11/99) - *Specification and Description Language (SDL)*, International Telecommunication Union (ITU), 2000

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