Paper submitted to

10th Aachen Symposium on Signal Theory

Name:	Matthias Siebert, Martin Steppler
Affiliation:	Communication Networks Aachen University of Technology
Address:	Kopernikusstr. 16 D–52074 Aachen, Germany
Phone:	+49 241 80 5828
Fax.:	+49 241 88 90382
E-mail:	{mst|steppler}@comnets.rwth-aachen.de
Paper Title:	Software Engineering in the face of 3/4G Mobile Communication Systems
Symposium Topic:	3rd and Next Generation Systems
Characterizing Keywords:	Software Development (Process), Quality Aspects, CASE-Tool, SPEET, Software (Defined) Radio, Genericity of Protocol SW
WWW:	http://www.comnets.rwth-aachen.de/~{mst|steppler}
Number of Pages:	8

Software Engineering in the face of 3/4G Mobile Communication Systems

Matthias Siebert, Martin Steppler

Communication Networks
Aachen University of Technology
Kopernikusstr. 16
D–52074 Aachen, Germany
Phone: +49 241 80 5828
E-mail: {mst|steppler}@comnets.rwth-aachen.de

Abstract

The development of (protocol) software (SW) has to be done from an engineer's point of view. By applying methods of project management, the process of software engineering can be scheduled, controlled and monitored. Since future systems will rely on well proven features, characteristics and methods of existing systems, it is obvious that the same applies to the employed SW. To enable 3G/4G systems to benefit from today's programmer's work, certain 'rules' have to be followed. Congruously, this paper firstly points out general steps within the SW development process, quality aspects and a CASE* tool called SPEET,** and secondly applies these worksteps by presenting a method for Designing Generic and Adaptive Protocol Software (DGAPS).

1 Software Development

Every developing process matches a process of problem solving: A development task is stated and finally at the end of the process one has to supply evidence that the solution of the problem, the software system, matches the specification.

1.1 Software Life Cycle

Several proposals have been made in the context of the software development process: In order to characterize the different steps, an underlying **Phase Concept** is defined. Each phase points out typical activities within the development process. Well known phases within the **Software Life Cycle** are:

- ▷ Specification
- ▷ Analysis
- ▷ Design
- ▷ Modeling/Implementation
- ▷ Testing/Validation
- ▷ Installation
- ▷ Usage/Maintenance

Within this scope, several different languages come into operation as shown in Fig. 1. In the beginning there is a qualitative outline of the problem followed by a *Feasibility Study*. In this phase an informal, nat-

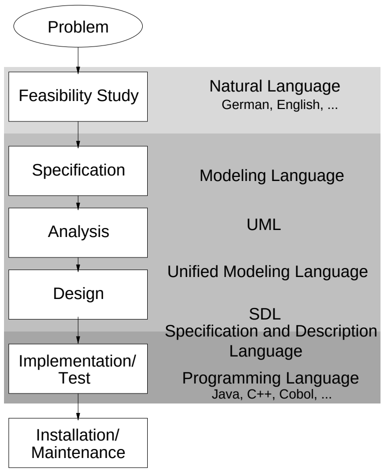

Figure 1: Languages in the process of SW engineering

*Computer Added Software Engineering
**SDL based Performance Evaluation Environment and Tools

ural language (German, English,..) is used. The next steps are definition of requirements (*Specification*), *Analysis* and *Design*. Since natural language includes a high risk of misconstruction, the modeling has to be non-ambiguous. This is why dedicated modeling languages come into operation.

Still at the beginning of the nineties, a large number of (object oriented) modeling languages were in use (Object Modeling Technique OMT, Booch-Method, Object Oriented Software Engineering OOSE,...) which led to acceptance problems. The Unified Modeling Language (UML) which was accepted in November 1997 by the Object Management Group (OMG) as official industry standard managed to incorporate and harmonize the different approaches.

Once the modeling has concluded, the specifications have to be implemented. This can either be done directly in a higher programming language, or by an intermediate step with the help of the Specification and Description Language (SDL). SDL is based on the theory of extended finite state machines. It is an object oriented programming language which was standardized by ITU-T[1].

The nexus of these phases results in a **Process Model**. The process model thus defines besides an overall structure, the contents and the results of the phases as well as their interconnection. The different paradigm of software development basically rely on a different **process strategy**, see Figure 2. Basically, one distinguishes between two concepts, pointed up by the **Life Cycle** and the **evolutionary Model**.

The Life Cycle again can be split up in the following sub-models with their respective properties:

- Linear Life Cycle
  - ▷ top-down-refinement
  - ▷ sequential processing of different steps
  - ▷ no return to preceding phases
  - ▷ implementation in relatively late process (phase) of development
- Iterative Life Cycle
  - ▷ top-down-refinement
  - ▷ stepwise approach: general specification $\leftrightarrow$ concrete tasks
  - ▷ return to preceding phases possible $\rightarrow$ error correction
  - ▷ implementation in relatively late process (phase) of development
- Prototype-oriented Life Cycle
  - – explorative, experimental, evolutionary
    - ▷ combination of prototyping-concepts with those of SW Life Cycle
    - ▷ reduction of evaluation risks
    - ▷ overlapping of different phases possible

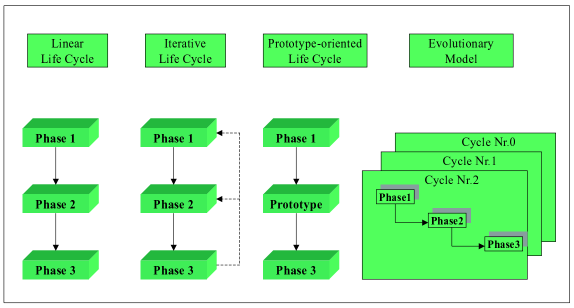

Figure 2: Paradigm within SW development

The evolutionary prototyping contravenes to the principles of the Life-Cycle-Models and leads over to an own concept, presented by the **evolutionary Model**, in which the SW developing process passes through several developing-cycles, each of which consists of the same phases. Although this model is also based on a top-down refinement, the onward level of detail is related to the developing-cycles and not to the phases.

1.2 Quality Aspects

DIN 55350 defines quality as the whole of characteristics and features of a product or of activities for the fulfillment of determined requirements[1]. In such a way, software can be characterized by quality-features, as they are:

- **correctness**: correct solution of the problem/task
- **reliability**: robust within error situation
- **efficiency**: economical handling of resources
- **handling**: easy and intuitive usage
- **maintainability**: easy to adapt and to advance
- **testability**: easy and fully testing of operativeness
- **flexibility**: adjustable to different terms of use
- **reuseability**: (partly) operation in different applications
- **portability**: portable to different hard-/software platform

In some points these targets are orthogonal which means that it is not possible to optimize all features at the same time. Table 1.2 relates the aforementioned (plus some other) features and their influence on the cost of development and time. Thus the developer has to define a main target and boundary conditions

[1]„Qualität ist die Gesamtheit von Eigenschaften und Merkmalen eines Produktes oder von Tätigkeiten zur Erfüllung festgelegter Erfordernisse.“

	Feature	1	2	3	4	5	6	7	8	9	10	A	B	C	D	E	F
1	correctness		+	o	o	+	o	o	o	o	o	-	+	-	+	+	o
2	reliability	o		o	o	+	o	o	o	-	o	-	+	-	+	+	o
3	adequacy	o	o		+	o	o	o	o	+	-	-	o	-	+	+	-
4	learnability	o	o	o		o	o	o	o	-	o	-	o	-	+	o	o
5	robustness	o	+	+	o		o	o	+	-	o	-	+	-	+	+	o
6	readability	+	+	o	o	+		+	+	-	+	+	+	+	o	+	+
7	change-/extensibility	+	+	o	o	+	o		+	-	+	-	+	+	o	+	+
8	testability	+	+	o	o	+	o	+		-	+	+	+	+	o	+	+
9	efficiency	-	-	+	-	-	-	-	-		-	-	+	-	+	+	-
10	portability	o	o	-	o	o	o	+	o	-		-	+	-	-	-	+

+/-/o : positive/negative/no impact A : time of development B : life time C : cost of development
D : cost of operation E : cost of maintenance F : portability cost

Table 1: Interaction of SW quality features with respect to costs and time [2]

subject to this limitation the product has to be optimized.

1.3 CASE-Tool: SPEET

A systematic approach to the analysis, design, implementation and maintenance of software usually involves the use of dedicated tools. A key word in this context is **Computer Aided Software Engineering** (CASE). One CASE oriented solution for product development using formal methods is **SPEET** [3, 4].

1.3.1 SPEET Tool components

SPEET defines a SW development environment supplemented by respective tools, as indicated in Fig.3. An automatic code-generator, called SDL2SPEETCL [5] maps system-software, formally specified in SDL on respective C++ classes of a dedicated class library, the SPEET Class Library, SPEETCL [6]. With the help of dedicated interfaces, the *Command Line Interface* (CLI) and the *Graphical User Interface* (GUI)

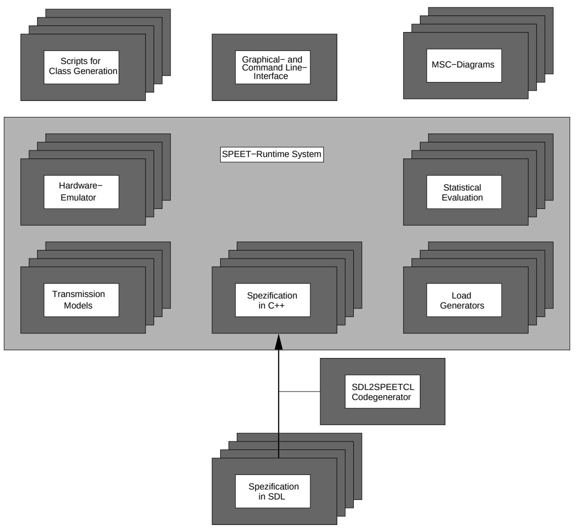

Figure 3: Interaction of SPEET components and tools

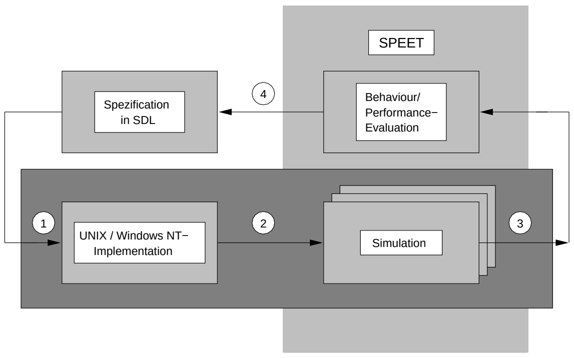
SPEET
Spezification in SDL
4
Behaviour/ Performance– Evaluation
1
UNIX / Windows NT– Implementation
2
Simulation
3

Figure 4: SPEET as used in the development cycle

the user can control the simulation process and visualize statistical data. Additional output for drawing (MSCs) for debugging purposes is also facilitated.

Altogether SPEET provides

- event based simulation of several SDL systems in parallel,
- various traffic load generators representing different services as well as an adjustable mix of them,
- transmission models including en-/decoding, de-/modulation and channel behaviour,
- statistical evaluation of data measured during simulation runs.

"The objective of SPEET is the design and evaluation of complex, formally specified communication systems in an early phase of their development by means of simulation [...]" [3]. For this, a recursive cycle is applied as shown in Fig. 4.

In a first step (step 1) the formal SDL specification is taken to automatically generate C source code which will be compiled under the same operation system SPEET runs on. Prior to code generation the user is supposed to insert probes into the formal specification [7] to gain additional information about the simulation model and its implementation (step 2). These information can be evaluated (step 3) and the respective insights can be used to improve the specification (step 4).

As we can see, SPEET follows the evolutionary model presented in section 1.1.

2 Design of 3G/4G Protocol Software

The following section describes the design of protocol software to be used in present and future mobile communication systems following the aforementioned phases within the life cycle.

Since the software is supposed to be used in current but also 3G/4G mobile communication systems, the main target that has to be achieved is **reusability**, cp. Section 1.2. The boundary conditions given here are change-/extensibility and efficiency. Due to Table 1.2 these quality-aspects do have a negative impact on each other. However, the here presented method of *Designing Generic and Adaptive Protocol Software* (DGAPS) features by minimizing this impact.

2.1 Problem Description

Today's different existing air-interfaces together with the different services they support still require the use of multiple physical handsets. Similar situations apply if the mobile is to be used abroad. First solutions to this problem offer multi-band, multi-mode portables that mainly represent a number of dedicated devices integrated into a single cover. This way of solution means that a certain overhead has to be accepted, due to the multiple protocol software having basically the same structure and functionality. Furthermore, updating of software in a terminal is difficult or even impossible. The introduction of new services like GPRS, HSCSD or EDGE reveals the disadvantage of the aforementioned solution: Though only a part of the air-interface related software has to be changed, newly developed devices will be necessary.

2.2 Feasibility Study

Software (Defined) Radios (SDRs) [8][9] maybe a more efficient solution. An SDR is based on processors and wide band front-ends and all kind of functionality is achieved by means of dedicated control software. Thus it is possible to design a multi-mode radio capable of supporting multiple air-interfaces and protocol stacks.

2.2.1 1. Requirements on SDRs

SDRs will be able to meet various demands. The support of several air-interface standards provides the ability to roam across access networks with one single handset, called network mobility. Since all modules of an SDR are under software control, a change of the functionality is easier to realize. Dedicated interfaces will allow to add new services without the need of hardware modifications.

Generation bridging will be eased, an important aspect since current and new standards like e.g. GSM and UMTS will have to coexist for a transitional period.

Another goal is reconfigurability: By modifying a radio's configuration software, its use with different access networks and the adaptation to different air-interfaces can be achieved. Since the protocols running at the base station (BS) and mobile station (MS) are basically symmetrically, reconfigurability is applicable to both network elements. Thereby an operator can built up an infrastructure that easily can be reconfigured to support a new standard in addition when necessary. This is interesting for both, developed countries that are in the migration from 2G to 3G mobile radio networks and for developing countries that are planning to introduce mobile technology but do not want to decide for a standard now due to the unpredictable acceptance of the systems under construction.

Reusability of software is a big concern. New air-interface standards typically rely on well-understood protocol stacks of predecessor systems. Cost intensive re-engineering of software can be avoided and software can be re-used if designed in a suitable way. Modular software design entails several advantages. On the one hand, portability of dedicated functionality is supported. Using well defined interfaces, the same module can operate within different systems. On the other hand software-upgrades are easily facilitated, as the respective modules can be changed individually.

There are a number of research issues that need to be addressed, like

- the definition of elementary commonalities of the various mobile communication systems,
- resource-sharing within telecommunication software,
- modular software design and interfaces,
- reusability of software,
- interworking of different systems.
- structure of a generic protocol stack,
- nature of software extensions to an existent system, or
- composition of an adaptive protocol stack architecture.

2.3 Specification

Since the configuration software and an implemented protocol stack represent one important part of a device, their structural composition, implementation and realization is of fundamental research interest. Having the ISO/OSI reference model in mind a high degree of similarity can be found for different air-interface standards. Concerning the control software many features can be implemented as *shared resources*. Applying DGAPS results in a *generic protocol stack*, that provides a common basis for a number of different systems. Specialization by introducing standard-specific functions to the generic stack stepwise results in a specific realization towards a specific protocol stack.

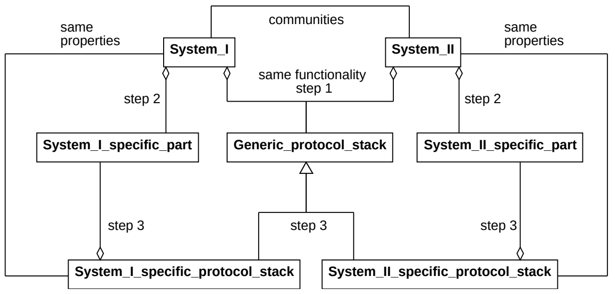

Figure 6: Interaction of components for an SDR protocol stack

2.4 Analysis

Identification of commonalities

In a first step (step 1) different systems, say System I and System II, are analyzed layer by layer to identify their commonalities. A more detailed description of the analysis process (exemplarily for DECT and GSM) together with a reference implementation is described in [10]. The number of different systems to be considered may be two or larger. The result will be an SDL specification of a common subset of the access protocol stacks for the systems, see Figure 5. Since this stack provides the common characteristics of the considered air-interface standards it is called a *generic protocol stack*.

Development of standard-specific supplements

The next step (step 2), see Figure 6, is to develop SDL specifications specific to given air-interface standards, say for System I or System II. These include functions that are specific to respective standards and thus represent the individual behaviour of a system. Different approaches can be taken to achieve that goal.

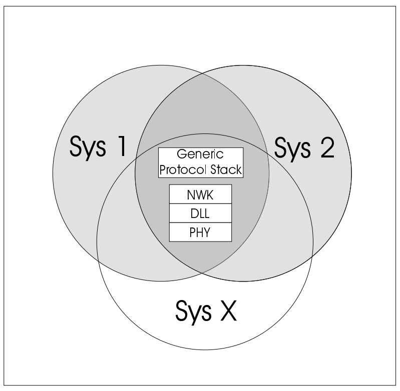

Figure 5: Generic protocol stack for various system types

In order to make use of the object-oriented properties of SDL together with inheritance, it is suggested to implement these parts as subclasses derived from base classes implemented within the generic stack. This is of special advantage, if more than two systems are considered; procedures that are common to most but not necessarily to all standards still will be implemented within the generic stack. The standard-specific supplements than will have to redefine the respective procedures and the behaviour required is achieved then.

Integration of a dedicated air-interface standard

To end in a dedicated air-interface standard, the generic protocol stack and the standard-specific supplement, have to be merged (step 3). This is done by means of inheritance. Figure 6 shows the correlations and dependencies of the aforementioned parts in the notation of UML. In order to distinguish between a specific protocol stack that is designed either conformant with the above presented approach or not, the notation *System_X* (non-conformant) and *System_X_specific_protocol_stack* (conformant) is used.

2.5 Design/Implementation

All the software specifications are being done formally with the help of SDL[11]. SDL provides language constructs for object orientated software design that support features like inheritance and information hiding. Because of its graphical (besides an equivalent phrase-) representation and an easy to understand finite state machine basis, SDL has been accepted worldwide for the specification of communication protocols.

To obtain the portable source code from an SDL specification, an automatic translator, called *SDL2SPEETCL* [12], is used. It takes the phrase representation (SDL/PR), generated by the SDT-Analyzer [13] (thus syntactical and semantical errors are excluded), as input and maps the system behaviour to classes of SPEETCL (SDL Performance Evaluation Tool Class Library) [6]. SPEETCL is a C++ class library developed for the integration of protocols specified in SDL into an event-driven simulation environment.

2.6 Optimizations/Validation

Optimization of the SDL-Specifications

To result in a run-time efficient code after translation it is necessary to undertake some code optimization. Investigations have shown that the following rules will result in a substantial speed-up:

- use of pointers in SDL instead of parameter lists
- decrease the number of process switching
- reduce the number of events and timers
- replace SDL data types by C-code constructs

Further Optimizations

In a final step those parts of the code are to identify that are most frequently used during runtime and the respective functions have to be considered to be implemented in hardware, say in FPGAs etc.

In fact, these aforementioned improvements mostly are not workable right from the beginning. Instead the identification process is part of the recursive cycle of the evolutionary model followed by SPEET as explained in section 1.3.1.

The optimized SDL specification can be guaranteed to be conformant to other specifications, since errors would appear when running against each other. Conformance thus will be reached after removal of these errors in the informally implemented code.

3 Conclusion

The focus of this paper is to point out a method for (protocol) SW design. Thereby it is important, that the software features by a high degree of genericity which allows employment as a basis for current and future mobile communication systems.

The first part of this paper therefore explains the process model as well as respective paradigm to be followed to design SW. Additionally the (partly orthogonal) quality aspects under which the development has to take place are presented. The use of CASE tools thereby is inevitable. One of these tools, SPEET, together with its components subsequently simplifies the different phases by providing an environment accompanying the development process.

The second part describes a method of designing generic and adaptive protocol SW (DGAPS) for current and future 3G/4G systems. The respective phases within the life cycle are taken up and connected to concrete development tasks.

Following these ideas avoids cost-intensive re-engineering since the achieved SW will distinguish oneself by a long life time, for use in XG mobile communication systems.

References

[1] ITU-T, "Specification and description language (sdl)." ITU-T Recommendation Z.100, Nov. 1999. 1.1

[2] D. Klöditz, "Software-Entwicklungstechnologie." http://www.inf.hs-anhalt.de/~Kloeditz/Informatik/17Softwaretechnologie/index.htm. 1

[3] M. Steppler and M. Lott, "Speet – sdl performance evaluation tool," in Cavalli and Sarma [15], pp. 53–67. Dieses Dokument ist unter http://steppler.de frei verfügbar. 1.3, 1.3.1

[4] M. Steppler, "Performance analysis of communication systems formally specified in sdl," in *Tagungsband des „First International Workshop on Simulation and Performance '98"* (C. U. Smith, P. Clements, and M. Woodside, eds.), (Sante Fe, Neu-Mexiko, USA), pp. 49–62, acm, 12.-16. Oktober 1998. Dieses Dokument ist unter http://steppler.de frei verfügbar. 1.3

[5] M. Steppler, W. Olzem, and C. Lampe, *SDL2SPEETCL – SDL-PR to C++ Code Generation Using the SPEETCL.* comnets, sep 1998. 1.3.1

[6] M. Steppler, *SPEETCL — SDL Performance Evaluation Tool Class Library, Rel. 3.2.0.* AixCom GmbH (www.aixcom.com), Dec. 2000. 1.3.1, 2.5

[7] International Telecommunication Union Q.6/10, *Performance Measurement of SDL Specifications – The New SDL Probe Symbol*, (Erlangen, Germany), 17th–19th February 1998. 1.3.1

[8] "http://www.sdrforum.org." Homepage SDR Forum Web Site. 2.2

[9] J. Mitola, "Technical challenges in the globalization of software radio," in *IEEE Communications Magazine, February 1999*, pp. 84–89, February 1999. 2.2

[10] M. Siebert, "Design of a Generic Protocol Stack for an Adaptive Terminal," (Proc. of the 1st Karlsruhe Workshop on Software Radios, Institut für Nachrichtentechnik Karlsruhe, Germany), pp. 31–34, March 2000. 2.4

[11] A. Olsen, O. Færgemand, Møller-Pedersen, R. Reed, and J. Smith, *Systems Engineering Using SDL-92.* ELSEVIER SCIENCE B.V., 1994. 2.5

[12] M. Steppler, *SDL2SPEETCL — An SDL to C++ code generator, Rel. 4.1.0.* AixCom GmbH (www.aixcom.com), Dec. 2000. 2.5

[13] Telelogic, Malmö, Sweden, *SDL Design Tool (SDT) 4.1 Reference Manual*, 2000. 2.5

[14] B. Walke, *Mobile Radio Networks.* Chichester, UK: Wiley, 2nd ed., 2001.

[15] A. Cavalli and A. Sarma, eds., *Tagungsband des 8. SDL-Forums: SDL '97 – Time for Testing – SDL, MSC and Trends*, (Evry, Frankreich), 23.-26. September 1997. Dieser Tagungsband kann unter http://www.elsevier.nl bestellt werden. 3

[16] Hering, Gutekunst, and Dyllong, *Informatik für Ingenieure.* No. ISBN 3-18-400944-0, VDI Verlag, 1995.