SDL based prototyping of ISDN-DECT-PBX switching software

H.-J. Vögel[+], W. Kellerer[+], S. Karg[+1], M. Kober[+2], A. Beckert[*], G. Einfalt[*]
[+] Lehrstuhl für Kommunikationsnetze, Technische Universität München
Arcisstr. 21, D-80290 München, Germany
Tel.: +49 89 289 23503, Fax: +49 89 289 63503, {voegel, kellerer}@ei.tum.de
[*] Bosch Telecom GmbH, München

Abstract

Telecommunication software engineering is a highly demanding task, since these systems often have to deal with a huge variety of features and customer specific requirements. A software engineering project set up between Bosch Telecom and the institute for communication networks at the Munich University of Technology was centered around switching software development for a DECT-enhanced ISDN-PBX system. SDL has been applied as the main tool for specification and implementation of the software, with an emphasis on code generation and rapid prototyping in mind. This paper introduces the application, briefly discusses options for modelling the system in SDL and gives an overview of the implementation's software architecture. It further reports on the practical experience gained with SDL during the project, with a special focus on the benefits of SDL and MSC in prototyping.

Keywords

SDL, industrial application, prototyping, telecommunication switching software, SDT

1 INTRODUCTION

Telecommunication software engineering is a highly demanding task, since these systems have to cope with diverse customer requirements. Particularly PBX systems must provide a huge variety of different customer specific features. Bosch Telecom GmbH is a leading company in the provision of telecom solutions. The development of private branch exchanges (PBX) for ISDN and DECT is one of the company's major activities. A project has been set up between Bosch Telecom GmbH and the institute for communication networks at the Munich University of Technology to improve the software development for these switching systems with a special interest in rapid prototyping for the switching software. The Specification and Description Language SDL [1] has been chosen because of its recommendation for telecommunications systems and its wide spread use in industry.

The remainder of this paper is organized as follows: in section 2, we briefly introduce the system setup used in our laboratory, which is based on a slightly modified production version of a standard PBX system. Section 3 discusses in more detail the SDL modelling approach and gives an overview of the complete switching software implementation. Finally, before concluding, section 4 discusses some performance issues, while section 5 elaborates on the experience gained with our prototyping approach, dealing with the benefits of SDL and the application of MSC techniques in particular.

2 LABORATORY SETUP: SWITCHING SOFTWARE FOR AN ISDN-DECT-PBX

The aim of the project described in this paper is twofold. First, to have a demonstration platform for teaching purposes. And second, to have a versatile hardware platform ready, facilitating projects in the area of software engineering techniques and rapid prototyping particularly for mid-size PBX switching systems.

---

1. *Now with Bosch Telecom GmbH, München*
2. *Now with Matra Communication Cellular Terminals GmbH, Ulm*

The system architecture of Figure 1 shows, how this was realized. We have a basic PBX prototyping system, consisting of a slightly modified production version of a DECT-enhanced Bosch PBX [2] on the one hand and a workstation-based software development environment on the other. Telelogic's SDL tool SDT was chosen as our CASE tool [3].

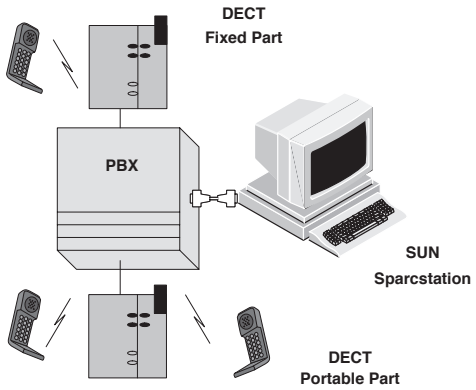

Figure 1: System Architecture

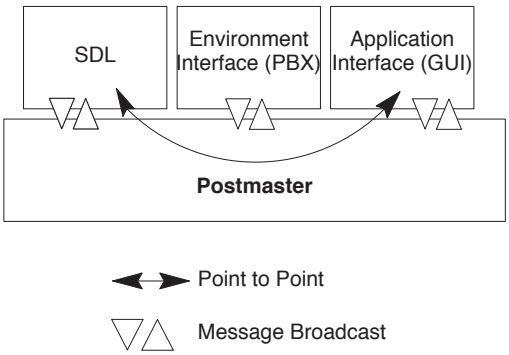

Figure 2: System integration with SDT Postmaster

A standard serial interface (V.24) is used for linking the PBX to the workstation. The PBX has been stripped of all its original switching software, leaving only a minimum of real time operating system and basic control software running on the original platform. The major part of the switching software has been replaced by the prototyping system running on the workstation. The message-based structure of the PBX's real time environment easily facilitates this system architecture. All messages destined for the original switching software tasks are being relayed to the workstation. The protocol and call control processing takes place in the prototype software system running on the workstation.

The SDT Postmaster – Telelogic's tool for the communication between the SDT components – has been used as the central means for system integration (Figure 2). Unix processes programmed in C provide the necessary interfaces to the serial line (environment interface to the PBX) and to potential system extensions like e.g. a Graphical User Interface (application interface to GUI) for visualization and configuration purposes. The messages received from the PBX are being converted into SDL signals by the environment interface.

The first application of the system setup described above was to reengineer the PBX's switching software. This involved a new software specification and implementation from the scratch, which was done completely in SDL.

The tasks of the switching software are: control of the PBX's hardware resources (tone generators, switching fabric etc.), protocol processing for the interfaces to the DECT subsystem as well as to the user terminal, stimulus processing for the user interface and call processing (implementation of the basic call state model). This involves a number of protocols, which Figure 3 shows in a schematic protocol architecture overview. The most important protocols are the Internal Management Protocol IM, the DECT Network Management Protocol DNM and the Application Layer Stimulus protocol. The IM protocol is used for the control of the PBX's switching hardware. The DNM protocol contains the control of the DECT subsystem and the translation of the DECT lower layer air interface signaling protocols [4] for the PBX, i.e. it has to perform tasks like control and configuration of the DECT fixed parts, authentication, location registration and location updating, paging etc. The stimulus protocol is used for signaling the user actions (basically keypress information) to the switching system and to control the portable part's hardware functions, like e.g. display information or ring tone selection. In addition to that, a modified data link control protocol has been introduced, called the Application Layer Datalink Control ADLC. It is used for protecting the stimulus information elements against loss or corruption on the air interface, e.g. in critical radio conditions or during handover.

3 ARCHITECTURE, SPECIFICATION AND PROTOTYPE

Specification and implementation of the protocol architecture's workstation part was the main development project, which is under discussion in this paper. Part of the software, like serial interface handling and the basic data link protocol IDLC were written in the C programming language, as these protocols have more utility function than they are part of the switching software. This piece of code is hereafter called the environment interface. Inte-

grated into this implementation of the environment interface is the application interface code (Figure 2). The highlighted part of Figure 3 is that part of the protocol architecture, which had to be implemented in SDL. In addition to that the switching software system has to provide means for call setup and connection control (not shown in Figure 3).

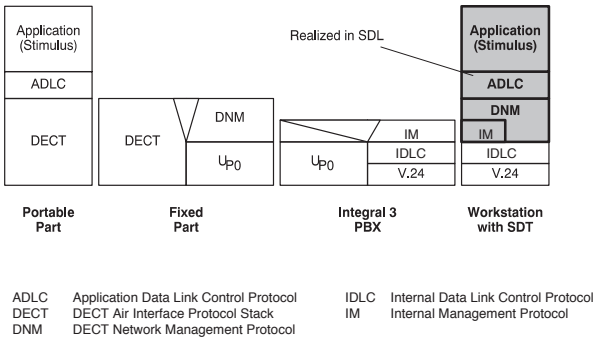

Figure 3: Protocol architecture

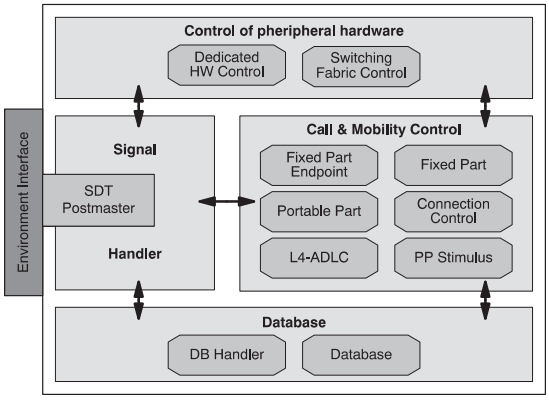

Figure 4: SDL software architecture

The overall software architecture defined in the top down design process is shown in Figure 4. Messages at the serial interface are converted from/into SDL signals by the environment interface and traded with the SDT Postmaster. The SDL-based switching software has a single central block (Signal Handler) for message reception and distribution of the signals to the proper processes. At the same time, the Signal Handler is responsible for receiving commands from and presenting results/data to the graphical user interface, which is connected to the system via the environment interface. The core switching software is divided into three parts, which are rather independent in functional respect: the control software for the peripheral hardware, the database and the mobility/call control software.

3.1 System design overview and modelling methods

Designing and specifying telecommunication switching software is a complex task, even more so as usually there is an oversailing system design process for the joint development of the hard- and software. There are plenty of options available for structuring the software architecture. For example, individual protocol entities can be implemented as SDL blocks with an SDL process for each of any separable protocol functions. Another means of structuring is to have one SDL process per hardware entity, performing all the protocol and control functions associated with a particular hardware entity. Furthermore, a very common technique in telecommunication systems is the state-transition-table approach, which, based on a context vector, basically implements an extended state machine for an arbitrary number of instantiations of a particular protocol, e.g. Q.931 user-network-interface signaling.

The software engineering approach taken in the project contains several modeling methods, all of which can be seen in Figure 4 and Figure 5. As a structuring guideline, we took both, the entities of the protocol architecture and the physical entities, like e.g. the DECT portable part. Consequently, we have a functional separation of protocol and control tasks into single SDL processes.

On the one hand, parts of the system architecture were modeled in the SDL system by directly associating SDL processes with hardware entities. This was done for the handsets (DECT portable part PP) and the basestations (DECT fixed part FP), where at system startup one SDL process is dynamically generated per PP and FP each. These processes implement the functionality of the DNM protocol, which also includes challenges and responses of the DECT air interface protocols to and from the switching software system. There also are two SDL processes for the core PBX system, implementing the IM protocol for control of the switching functions and peripheral hardware inside the PBX, i.e. specialized resources like e.g. tone generators. For the ADLC data link protocol, a single-process implementation has been chosen. This ADLC implementation employes the state-transition-table approach, where a state vector is associated with each ADLC connection. The call modelling has been carried out in the form of a state-transition-table with a single-process implementation, too. For each call in progress a state vector is being initialized and controlled by the single call control process. In the end, the stimulus protocol is translated into the *PP Stimulus* process, responsible for encoding and decoding stimulus protocol information elements.

In addition to that, some auxiliary processes are necessary to operate the system. This in the main is the *Fixed Part Endpoint* process (Figure 5), which is responsible for reading the system configuration information from the database during startup and for dynamically instantiating the FP- and PP-process entities.

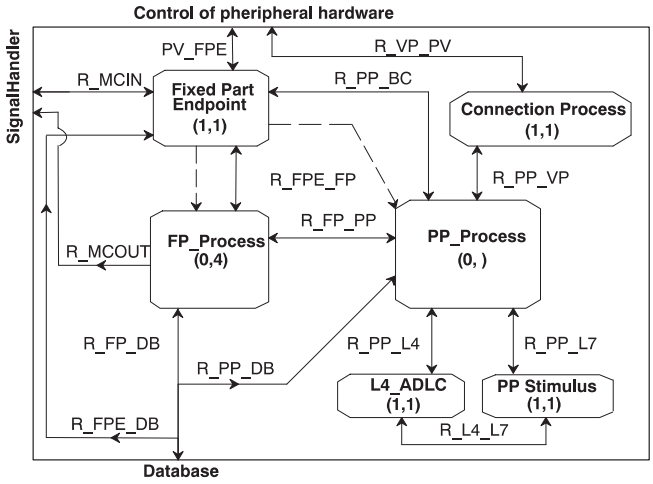

Figure 5: Processes of the Mobility/Call Control block

As already mentioned, an interesting technique for efficient implementation of protocol state machines is the state-transition-table approach, used in this project to implement the *ADLC* and *Stimulus* protocol and the call control. For implementing such an extended finite state machine, we specified an array definition for saving state vector information and a single protocol process, which uses the SDL continuous signal to jump to the process state saved in the state vector and branch into the correct part of the protocol processing (Figure 6). An example state vector for the call control context of a connection is being given in Figure 7.

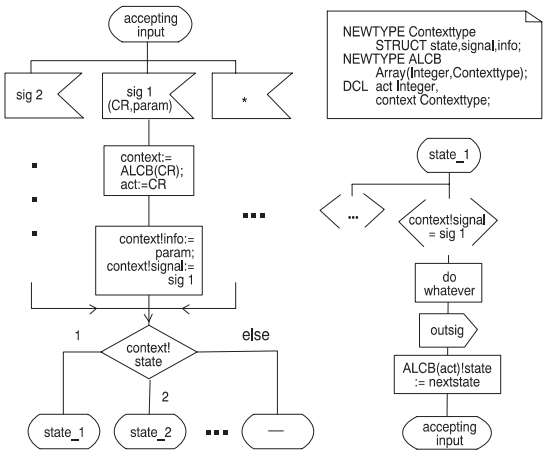

Figure 6: Principle of a state machine implementation in SDL

```
NEWTYPE Hoerton
LITERALS   Kein,Besetzt,Freiton,
           Hinweis,WTintern
ENDNEWTYPE;
NEWTYPE Connecttype
  STRUCT  ProzID  PID;
          Adr      phys_adr_type;
          Rufnr,
          STS,
          ETS,
          BCH,
          PartyCR,
          Nummer   Integer;
          Zustd    Charstring;
          Ton      Hoerton;
ENDNEWTYPE;
NEWTYPE  CCType
   ARRAY (Integer,Connecttype)
ENDNEWTYPE;
```

Figure 7: Call Control connection context definition

3.2 Hardware control and database implementation

To operate a telecommunication switching system, even a small scale system like the PBX at hand, a multitude of administration and control tasks has to be performed. This ranges from the administration of a variety of peripheral hardware (tone generators, tone recognition devices, conferencing equipment, music output and other specialized resources) to the control of the time division multiplex based switching fabric. Therefore, a dedicated SDL process has been implemented to perform either of these tasks (*Dedicated Hardware Control* and *Switching Fabric Control*, cf. Figure 4). At system startup time, as the PBX's real time operating system scans the hardware equip-

ment, the different components are being registered with the switching software. Through special *IM* registration messages, the *Dedicated Hardware Control* process learns about the components available in the PBX. Thus, having a clear picture of the configuration of the PBX at hand, the SDL process makes some preliminary arrangements, e.g. switching specific tone generators (dial tone, alerting tone etc.) onto dedicated time slots of the switching fabrics for easy later availability. Naturally, the peripheral hardware control processes have to interact with other parts of the switching system, especially the call control.

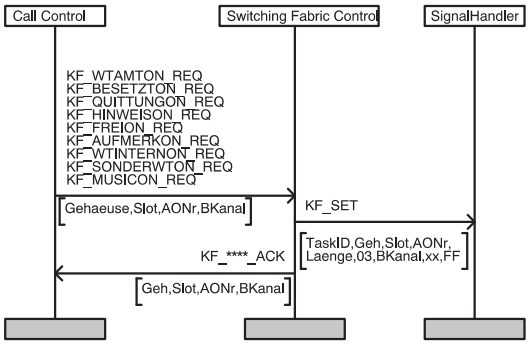

Figure 8: Signal interaction for connecting signaling tone generators to a party's handset

An example case is depicted in the MSC of Figure 8: during call setup, the calling party needs to be connected to different types of tones, in particular the dial tone and the alerting signal or the busy signal. Therefore, the *Call Control* process requests a desired tone to be connected to a particular terminal by the *Switching Fabric Control* process, which in turn instructs the *Signal Handler* process to assemble an *IM* command (KF_SET) and send it to the PBX.

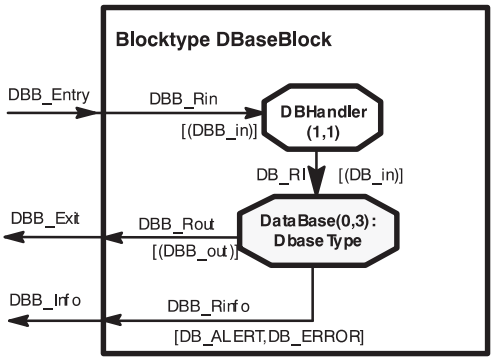

Figure 9: Internal database structure

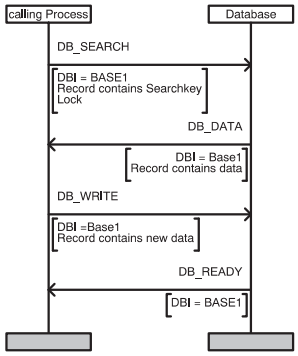

Figure 10: Updating of a database record

A very important central resource of the switching software is the system database. At the moment this database contains three tables, namely the *Fixed Part Database*, the *Portable Part Database* and the *Location Database*. During the initialization phase at system startup, the database is being read from ASCII files. From then on it is maintained as memory data structures, with changes made to the database during runtime immediately being written out to the hard disk. The *FP* table holds information about the system's hardware configuration, while the *PP* database contains identities and keys for the DECT handsets registered with the system (e.g. International Portable User Identity IPUI, Temporary Portable User Identity TPUI, User Authentication Key UAK, phone number associated with the handset). The database is implemented using the SDL-92 object-oriented approach (Figure 9). At the moment, the implementation consists of a single *DBHandler* process and three *DataBase* processes, one for each table. The *DBHandler* receives all requests to the database, handing them over to the correct *DataBase* process. The *DataBase* process in turn holds the corresponding data structure, which is organized as a chained list based on an Abstract Data Type supplied with the SDL tool. These processes perform the core database functions, like searching and retrieving records, updating fields in a record or deleting entries from the database. An example for the signaling at the database block's control interface is given in Figure 10, which shows the updating procedure for a specific record.

3.3 Call and Mobility Control

The call and mobility control processes are the switching software in the very sense of the word. They, at a very high system level point of view, control the processing of basic calls. To perform this complex task, they have to rely on services provided by other parts of the switching system and consequently have an elaborate interface for close cooperation with other processes (hardware control, database). The call/mobility control block analyses signaling information elements coming in from a portable part via an ADLC connection in the *PP Stimulus* process. Depending upon the contents of the stimulus elements, in the main actions are being taken in the FP/PP process for mobility control and in the *Connection* process for call control.

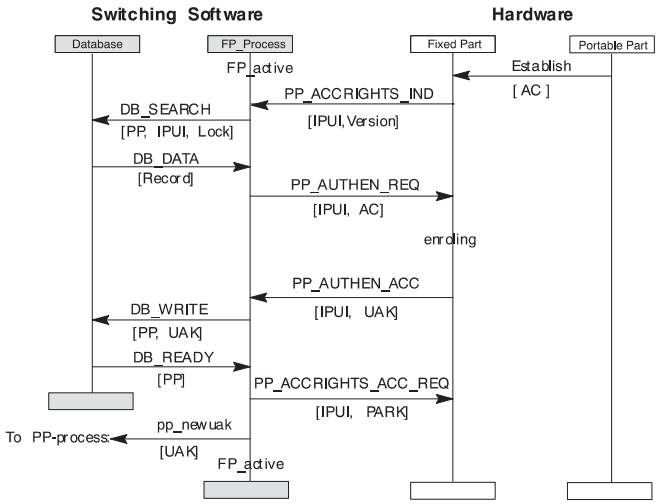

Figure 11: Portable Part Enrolment

Most of the mobility control functions are performed without a call being active. They include functions like authentication and enrolment of portable parts, allocation and assignment of temporary identifiers (DECT TPUI) and location updating. An example signaling transaction is shown in Figure 11 displaying the principle enrolment of new handhelds in the system. For fraud protection reasons, in DECT PBX systems every handheld has to be registered in a local system to be able to place and receive phone calls. This registration process, called enrolment, is normally performed by the DECT fixed part. In our system, it necessitates services from the switching software (FP process and database). To register with the system, the portable part has to know the system's access code AC. Upon a registration attempt, this access code is being retrieved from the PP's (preconfigured) database record and sent to the fixed part, which then is able to perform the enrolment. After successful enrolment, the DECT lower layer protocols generate a new access rights key UAK for that specific PP. This UAK is then stored in the database and the corresponding PP receives a notification about the UAK change.

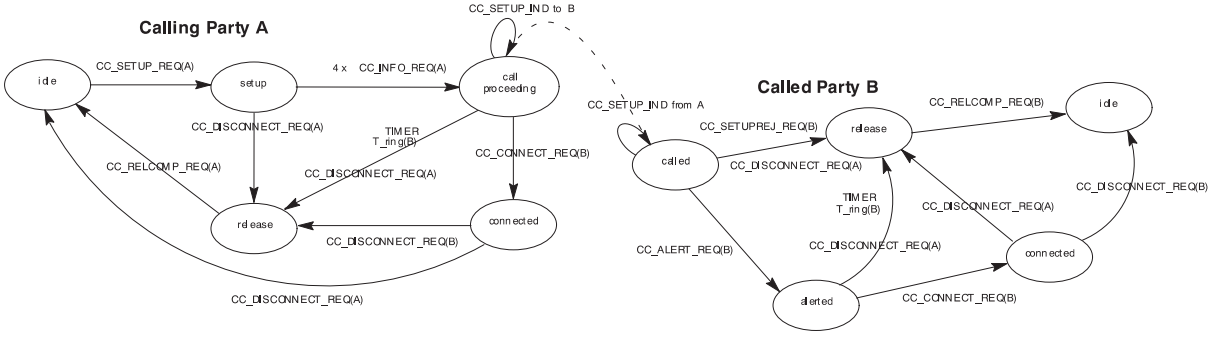

Figure 12: Call Control state transition diagram

The *Connection* process performing the call control functions implements the basic call state model of Figure 12. It is designed as an extended finite state machine using the state transition table approach. There are always two different call segments, the calling party segment and the called party segment. Consequently, for each complete call, there are at least two state vectors (cf. Figure 7) being generated. To perform its task, the *Connection* process closely interacts with several processes. It receives requests (setup request, target number info request etc.) from the SDL processes corresponding directly to the handsets (*PP Process*). It sends commands to the processes controlling the peripheral hardware and the switching fabric, e.g. to connect signaling tones or make adjustments to the switching fabric.

3.4 Further Applications

The specification and implementation of the basic switching architecture was the first application of the prototyping setup. Several projects have been carried out afterwards using the same environment and also the main parts of the switching architecture that is described in this paper. In the following we will report on some related research projects, where the prototyping setup is being used in.

A graphical user interface was developed to support the administration and visualization of the switching software [7]. This GUI allows to start and stop all parts of the software system conveniently and realizes remote control functions for the SDT system. It is possible to record signaling sequences during the run of the system and to analyze them afterwards in an off-line mode. Visualization of the signaling between the PBX and the switching software is another feature of the GUI. We have realized a detailed protocol analyzer, a database observer and an MSC visualizer. In addition to that the GUI has been designed as a specialized front end for the SDT simulator with predefined and editable messages. The GUI is realized with the scripting language Tcl/Tk [8] and connected via a socket interface to the environment/application interface of the system.

To enhance the functionality of the switching system with new services we applied the concept of Intelligent Networks (IN) [9] to the basic switching software. The main idea of the IN is the separation of basic switching functionalities and the control of supplementary services e.g. call forwarding or abbreviated dialing. This reduces complexity when integrating new features into the system. To validate this approach two different prototypes were implemented.

In both cases the supplementary service control (associated with a Service Control Point SCP) was strictly separated from the basic switching system (Service Switching Point SSP) according to the principles of the IN. For this reason the call control had to be redesigned to support the IN call model and service switching functionalities.

One prototype was to realize the SCP in another SDL system connected via the postmaster to the basic switching system [10]. The other prototype used Java for the implementation of the service control [11]. For the connection to the Java-SCP an additional “application interface” had to be built. Figure 13 shows an overview of both approaches.

Beneath the projects described above, further research projects are planned using the prototyping setup. It is intended to enhance the switching software with more mobility functions e.g. handover between neighboring fixed parts. For teaching purposes the prototype platform is going to be prepared to serve not only as a demonstration platform but can also be used for prototyping by student themselves in short exercises to show the benefits of SDL usage in system engineering. This leads to an extension of the laboratory course “system engineering using SDL” which is held at the institute of communication networks [12].

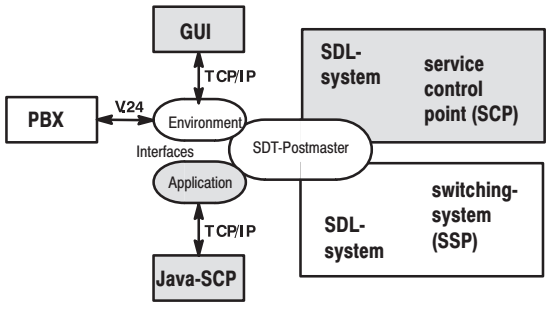

Figure 13: Related research projects: GUI and separated service control (SCP) done in SDL and in Java

4 PERFORMANCE ISSUES

The arrangement of the prototyping system has several advantages in the software development process. Most of all, no cross-compilation is necessary for runtime testing of the software system specified with SDL. The whole system can be operated with the switching software running as a UNIX task on the workstation. Telephone calls for testing purposes can be made with the SDL system running in a simulator environment, with the possibility to display MSC trace information while the PBX is being operated. This integrated approach to specification, implementation and testing greatly speeds up software development and shortens test cycles. At the same time, no great requirements are imposed on the development system: we used a SUN sparcstation 4 from SUN microsystems with 64 Mbyte of memory and a Sparc processor running at 110 MHz clocking frequency.

The performance and real-time requirements imposed by the switching hardware, like e.g. hardware timers in the telephone handsets for the ADLC, could be met for single call scenarios. Problems like lost signaling messages came up when two or more handsets tried to log on at the same time – the transaction rate provided was too low, due to the system running in simulation mode.

To get an overview of the system's size and complexity some figures about number of SDL processes and SDL procedures are given in Table 1.

Block	**No. of processes**	**No. of procedures**
Peripheral control	2	2
Database	4	13
Call & mobility control	6	16
Signal handler	1	0

Table 1: Software metrics: SDL processes and procedures

The prototype has not been designed for efficient code generation for the implementation on certain microcontroller environments of the PBX hardware platform. Case studies at Bosch Telecom resulted that the generated C-code using SDT Cbaisc code generator is too large to fit for PBX's processors. In fact Bosch Telecom is using SDL for the implementation of protocols on their platforms. The SDT Cmicro code generator is used for this purpose. This means that the use of SDL is restricted in the way that not all SDL techniques and symbols can be used. In the case of our laboratory system, several modifications in the SDL specification would be necessary in order to generate target code for the step from prototype to product. The next chapter will give a more detailed discussion of SDL and prototyping.

5 PROTOTYPING USING SDL

Prototyping within software engineering is part of the software development life cycle and describes the rapid implementation of certain parts of a system or the whole system itself on a platform which is not necessarily identical with the final platform. Prototyping is a means for the evaluation of system requirements or the evaluation of decisions in system design. The industrial application described above focused on the latter case: functional evaluation of software for a complex switching system. In that way the prototype has not been built for performance testing reasons but for the validation of the software architecture realizing the switching system and its functionality.

5.1 Benefits of SDL for prototyping

The use of SDL for specification and the use of valuable tools for SDL-based prototyping for the industrial application has been motivated due to two main advantages. SDL reduces time of development for prototyping since specification, design and coding are done in one language. This is a decisive point for rapid prototyping confirmed by the experiences gained during the project. Beneath that, CASE tools for SDL-based development provide features not only for specification, design and simulation but also for system implementation. Thus, it is possible to use the SDL-prototype as the basis for the implementation of the system on the final hardware platform (evolutionary prototyping). Depending on the target system the generated code has to be modified to meet the performance requirements.

Time of development for the whole prototype SDL switching system was about only 16 man months, including training on SDT and the PBX system. This is very short compared to conventional development of systems of the same complexity. The use of SDL in combination with CASE tools provides one notation for the phases of development. This supports and speeds up development cycles since specification, design and coding are always kept consistent. In conventional prototyping different notations or programming languages are used in each step of development. This makes it difficult to implement new design ideas or advanced functionality in a prototype when even before the functionality could be validated the use of certain languages forces a redesign due to implementation issues. In this way SDL supports top-down design in an excellent way. Figure 14 illustrates the comparison of SDL based prototyping with conventional prototyping (see also [13]).

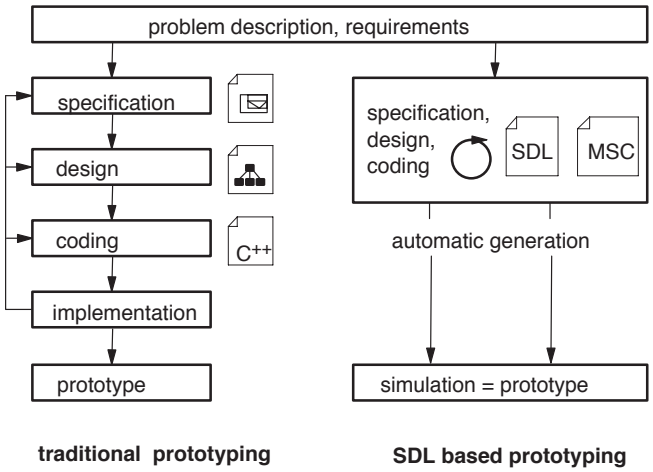

Figure 14: Prototyping for functional evaluation using SDL

Another advantage of SDL lies in its graphical representation, which made it easy for the development team to share specifications and work in parallel on the system. Due to formalized interface descriptions in SDL the requirements stated by the switching system's protocol stack were easily realized in the prototype. SDL is the best choice for the specification of even large protocols. That was especially true for the complex call control that had to be designed for the switching system.

A fact that also should be mentioned here is the aspect of teaching. The graphical specification and structuring methods of SDL greatly helps the students to understand, how telecommunication switching systems work. Even without the graphical user interface that was built in a related project the prototype served as an excellent demonstration system to show the complexity of the system and all message passing that lies behind just calling another telephone handset.

5.2 Application of and experiences with MSC

At the Bosch development site in Munich, MSC are used in all stages of the development process. When a new project based on an SDL system is set up, MSC are used for specification purposes. After a crude design of the system in SDL at block level, the information interchange between the components of the system is developed using MSC. As a result the communication links between the processes are determined and specified as well as the signal interface to the SDL environment and the system's internal protocols.

In the next step a verification of the design (design review) takes place. Now the system design is verified by checking the specification MSCs against the system requirements. An additional exemplary check of error situations is done to determine if the proposed system is able to cope with erroneous situations or if a redesign for parts of the system is necessary. Thus it is ensured that the system design has no severe flaws and that all the necessary functions are implementable.

During the implementation stage of the system, the behavior of the system or a set of system components is checked by interactive system testing. As a result MSCs of the simulations are written in real time, both on the host and in the target system. Those are checked against the specification MSCs to ensure the implementation complies to the specified behavior.

In the final stage MSC of all the testcases are produced and stored for documentation purposes as required by the Bosch guidelines for quality management. The MSC-diagrams give evidence to the fact that the system meets the requirements.

6 CONCLUSION

We have presented our laboratory setup for rapid prototyping based on a slightly modified production version of the Bosch Telecom DECT-enhanced PBX system. A brief discussion of the software modelling approach and of the system architecture gave an overview of the switching software implementation. SDL and MSC techniques have proven highly beneficial in this kind of application and development environment.

ACKNOWLDEGEMENTS

The authors would like to thank the company Bosch Telecom for the support during the project as well as for preparing excellent interfaces for the prototyping system to the PBX hardware. We thank all our colleagues for many lively discussions and for providing valuable assistance.

REFERENCES

[1] ITU-T Recommendation Z.100: “Specification and Description Language (SDL)”, ITU-T, Geneve, 1993.

[2] Bosch Telecom: “I3 Technisches Handbuch”, Bosch Telecom, 1996.

[3] Telelogic: “SDT 3.02 Reference Manual”, Telelogic AB, Malmö, Sweden, 1996.

[4] ETS 300 175: “Radio Equipment and Systems, Digital European Cordless Telecommunications (DECT) Common Interface”, ETSI, 1992.

[5] S. Karg: “Entwicklung von Systemsoftware und SDL-basierter Ruf- und Mobilitätssteuerung für eine ISDN-PABX mit DECT Erweiterung”, Diplomarbeit am Lehrstuhl für Kommunikationsnetze, Technische Universität München, 1997.

[6] M. Kober: “SDL-basierte Entwicklung von Ressourcensteuerung, Stimulusprotokoll und Rufsteuerung für eine ISDN-PABX mit DECT Erweiterung”, Diplomarbeit am Lehrstuhl für Kommunikationsnetze, Technische Universität München, 1997.

[7] W. Brummer: ”Entwicklung eines Bedienkonzeptes und Implementierung einer graphischen Benutzeroberfläche für eine DECT-Nebenstellenanlage”, Diplomarbeit am Lehrstuhl für Kommunikationsnetze, Technische Universität München, 1997.

[8] J. Ousterhoud: “TCL and the TK toolkit”, Addison-Wesley, 1994.

[9] ITU-T Recommendations Q.12xx: “Intelligent Network Series”, ITU-T, Geneve, 1993.

[10] P. Kopetzky: “Entwicklung einer IN-basierten Ruf- und Leistungsmerkmalsteuerung für eine ISDN-DECT-PBX und Realisierung eines SCP in SDL”, Diplomarbeit am Lehrstuhl für Kommunikationsnetze, Technische Universität München, 1998.

[11] W. Weiß: “Entwicklung einer IN-basierten Ruf- und Leistungsmerkmalsteuerung für eine ISDN-DECT-PBX und Implementierung einer Dienststeuerung mit Java”, Diplomarbeit am Lehrstuhl für Kommunikationsnetze, Technische Universität München, 1998.

[12] W. Kellerer; A. Autenrieth; A. Iselt: “SDL based protocol engineering and visualization for education: ISDN Q.931 case study”, submitted to FORTE/PSTV’98.

[13] W. Kellerer; A. Iselt; R. Riek: “Using SDL for the specification, simulation and implementation of an advanced OSI data-link protocol on an embedded microcontroller system”, in Gotzhein R., Bredereke J. (Edts.): Formal Description Techniques IX: Theory, application and tools (Proceedings of FORTE/PSTV’96), Chapman&Hall, 1996, pp. 419-434.

[14] ITU-T Recommendation Z.120: “Message Sequence Chart (MSC)”, ITU-T , Geneve, 1993.