7   REFERENCIES

[1] CCITT I.130 Method for characterisation of tele-communication services

[2] CCITT Q.65 Detailed description of Stage 2

[3] CCITT I.310 ISDN Network Functional Principles

[I] Experiments with Programming Languages and Techniques for Telecommunications Applications. B Dacker, N Elshiewy, P Hedeland, C W Welin & M Williams. Sixth International Conference on Software Engineering for Telecommunication Switching Systems, April 14-18, 1986, Eindhoven.

[II] Erlang - an experimental telephony programming language, J Armstrong and R Virding, ISS 90, Stockholm May 27 - June 1, 1990

[III] Using Prolog for Rapid Prototyping of Telecommunication Systems. J Armstrong & M Williams. Seventh International Conference on Software Engineering for Telecommunication Switching Systems, July 3-6, 1989, Bournemouth.

[IV] Logic Programming at IBM: from the Lab to the Customer, Bernard Robinet, ICLP Paris 1991.

[V] Implementing a Functional Language for Highly Parallel Real Time Applications, J Armstrong, B Dacker, R Virding, M Williams, Eighth International Conference on Software Engineering for Telecommunication Systems and Services, 30 March - 1 April 1992.

Call Completion on Busy / No Reply

Operator Recall

Intrusion

5 EXPERIENCE

Application Designers. The layering approach in the software architecture combined with the close mapping to the functional structure of the specification makes it possible for one person to perform the entire feature design and implementation. He / she may also be responsible for the function specification work phase. This allows designers to become real "application designers" focusing on customer requirements. Lower layers in the architecture are taken care of by system designers.

Increase in Design Efficiency. The close mapping of the software architecture to the functional structure makes the work model very simple. As a consequence the number of documents to be produced is greatly reduced. Some of the documents in the prototype are automatically generated. Measurements of design efficiency compared to that in present systems indicates an increase in efficiency up to 10 times!

Easy and confident Planning. The number of persons involved in design and verification of a specific feature is reduced to one, which offers the advantages e.g

Annoying waiting times disappears and lead times are greatly reduced.

Planning is much more accurate.

Designing a Feature Module. The main factors that makes it stimulating and easy to design a feature module are:

The written text of the Function Specification corresponds to the code in the corresponding feature module which improves code / feature understanding.

Programs can be made small and surveyable by using language features such as matching, list handling and recursive functions.

The design is incremental and interactive and allows structured growth.

Patching is not needed as programs can be recompiled on the fly.

Repeated verification of parts of a complete feature is "automatic", i.e. merely by test file activation.

Data can be displayed at a high symbolic level

It is not necessary to consider capacity problems during design.

There are much fewer documents to write.

Performance versus CPU Requirements

Erlang was first implemented as an interpreter written in Prolog. This was useful to develop and experiment with Erlang but it had unsatisfactory real time performance. The present implementation of Erlang is based on a virtual machine for which an Emulator has been written in "C". [V]. The compiler which generates code for this machine is itself written in Erlang. Performance of this implementation is very promising and constantly improving. Performance measurements on a prototype implementation indicate that a Motorola 68040 running this implementation of Erlang can handle between 9 000 and 26 000 BHCA.

6 CONCLUSIONS

The architectural work is based on genuine knowledge about the application. This made it possible to define a limited number well defined entities in a layered structure. The main work effort has been to delimiting these entities and to strictly define their functional contents rather than to develop methods for how to document and inherit the objects.

Combining this architecture, which is easy to understand and handle, with a real time declarative language, has greatly reduced the amount of work needed to implement new services and features in telecommunication systems.

This makes it possible to perform "real life" tests of new services and features by implementing advanced prototypes before they are introduced in a large scale into the market.

These technologies, when they have matured sufficiently for use in products, will make it possible to move the volume of our work away from implementation problems and, in the future, allow us to concentrate on customer needs and the development of new services.

Further development of these technologies are therefore of vital interest for the whole telecom business.

2.4 Programming Language

The use of declarative languages, such as Prolog, for programming conventional sequential applications is now well known to result in small programs and greatly increased programmer productivity [IV]. A series of experiments [I], [II], [III] have been performed at Ericsson to determine if such languages can also be used to implement highly parallel real time control applications and, if so, if similar productivity gains can be made. Languages such as ML, Prolog, Parlog, Hope and SASL were used. These experiments showed that productivity gains similar to those for sequential programming could be made, but these languages lacked the necessary concepts to handle parallelism, real time etc. in a manner suitable for our applications. Either there were no concepts to express concurrency (as in Prolog) or the granularity of the parallelism available (for example in Parlog) did not enable us to easily represent one asynchronous telephony process by one process in the language. A result of these experiments was the development our own language with many of the features of the languages investigated but with concurrency and error recovery built into the language. The language developed is called **Erlang**.

Some characteristics of Erlang are:

Small and simple language with functional / declarative semantics

Symbolic

Choice by pattern matching

Light weight processes with asynchronous message passing

Support for error detection and error correction

3 PROTOTYPING ENVIRONMENT

General. Standard work stations running under UNIX are used. Together with standard commercial or freeware products these provide a user interface including:

X - Windows

Archives with menu based access

Version management

Any editor running under UNIX

Document Preparation with Frame Maker

Communication with electronic mail

A support system for prototyping is being developed. This contains tools for the specification phase as well as for the feature design and verification phase. It contains tools which are common for these work phases such as:

Browsers

Selected views

Hypertext links

Traceability within document, between documents in the same work phase, traceability between specification and code

Specification Environment. For the specification work phase the support system provides graphics tools, tools for static and dynamic modelling and templates etc.

Feature Design and Verification. For the design and verification work phase the most important tool today is the Erlang System which supports:

Execution of a feature module either using simulated or real hardware.

Tracing of individual functions.

Spying of all communications to a particular process.

Examination process structure, i.e. see which processes are suspended, how processes are linked for error recovery purposes etc.

Examination of process global variables.

Recompilation of code "on the fly" and loading new or changed software modules into running a system.

The support system also provide interactive graphics, a database and other tools.

4 IMPLEMENTED FEATURES

Several features were chosen as test objects for evaluating the prototyping technology. These features were implemented according to the function specifications for Ericsson's MD110 PBX. These features are:

Basic Call

Basic Network Call

Basic Cordless Call

Calling Line Identification

Three Party Services

Call Forwarding

Operator Extending

Generic functions for device allocation/deallocation.

Generic functions for drivers.

Error recovery

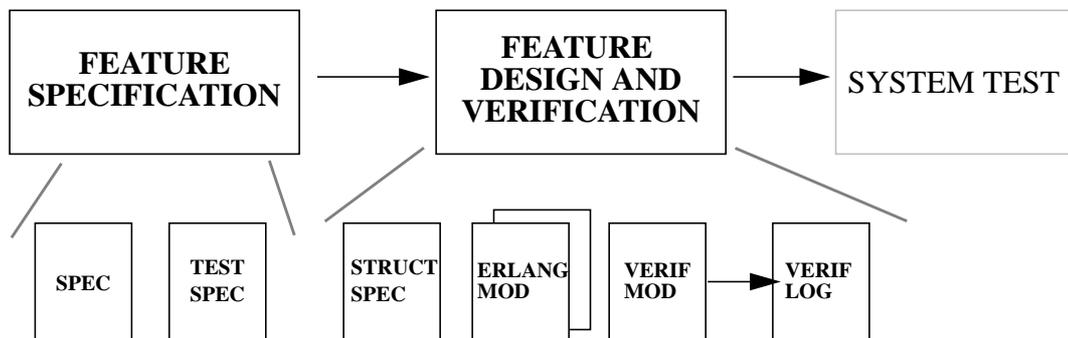Hiding S/W distribution on CPUs and the system configuration

This raises the abstraction level of programming and contributes to fault tolerance. BOS maintains information about the resources which have been allocated to application processes and which processes are linked together into a transaction. If a fault occurs in a transaction, either due to a programming error or because a hardware failure occurs, BOS kills processes involved in the transaction and frees the resources allocated. This enables per call error recovery and results in a very robust test environment. Hardware failure and software faults thus only affect the transactions in which they occur and their effects can be tidied up in an orderly manner.

2.3 Work Methodology

Prototyping is a very important phase in the overall methodology and focuses on the user. The methodology involves the user from the start. User studies are made for new applications, simulations of user interfaces with real users and test protocols are performed, demonstration of prototypes in real service are made to users.

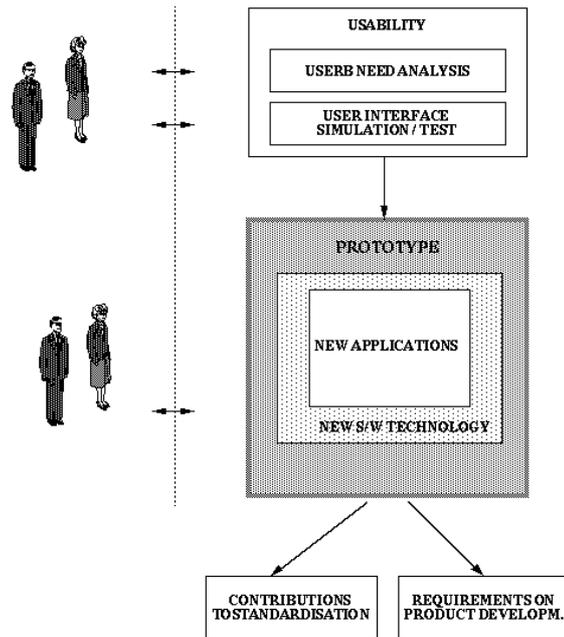A prototype of a new applications or features is designed and installed and taken into service at some friendly user site. The actual methodology for making the prototype is reduced to a few steps. This is possible since features are contained in feature modules. The methodology and the corresponding documents produced in the specification and design phases is shown below.
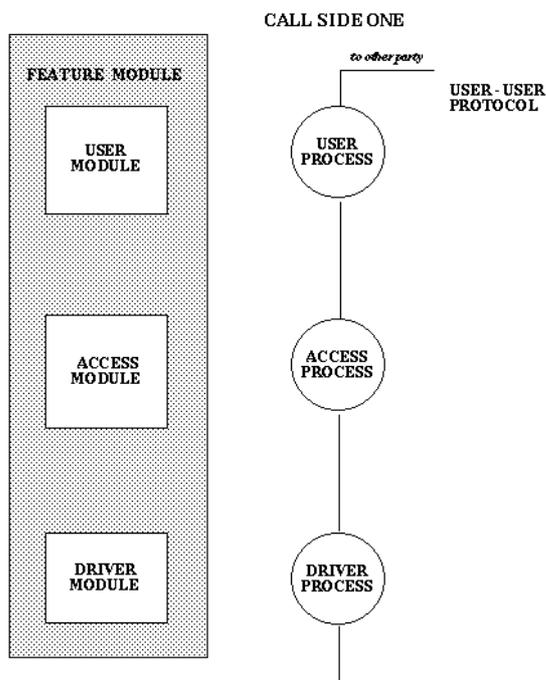
The layers of the software architecture are

> the application layer
>
> the application operating system layer
>
> the basic operating system layer

Application Layer

The application layer code is delimited to have as good a match as possible between the application structure and the run time structure. See figure below.

The Application Layer has a number of independent **Feature Modules**.These reflect the sales objects defined in the specification work phase. The Feature Modules are further subdivided into

> **User module.** The User module handles access independent parts of the feature i.e the traffic control part of the feature
>
> **Access modules.** The Access module handles terminal characteristics and origination / termination of call sessions
>
> **Driver modules.** The Driver module handles encoding of logical signals to bit streams to the hardware and decoding of bit streams and logical signals from the hardware.

There are three types of Feature Modules. These describe the complete feature either for

> the telephony task including signalling protocols
>
> the management task
>
> the interaction with between features.

Basic Call is a mandatory feature.

A Library of reusable components provides a powerful tool to raise the level of abstraction of application design. It contains functions which are frequently used when designing features. These functions are identified in the specification phase, which is run through for every new application.

Application Operating System Layer, AOS

The AOS layer provides support functions for the application layer to avoid duplicating code in several different features and to raise application programming to as high level of abstraction as possible by hiding implementation details from the application designer.

The AOS layer has two main groups of functions, a Tool Kit and Generic Functions.

The Tool Kit provides general purpose functions to the Application Layer, e.g

> Inter Party Communication
>
> Switching
>
> Queue
>
> Timing
>
> Call History
>
> Number Analysis
>
> Configuration Management

The Generic Functions in AOS provides the mechanism for the execution of the user and access programs in the feature modules.

Basic Operating System, BOS

BOS provides facilities similar to those available on standard time sharing systems, with the addition of primitives and functions needed for telecommunication applications e.g

> System start up and restart
>
> Data base storage and retrieval

2 PROTOTYPING TECHNOLOGY

2.1 Specification Structure and Methodology

The initial task in making a prototype is to write the specification. Specification structures and specification methods have been studied. A large number of specifications are available from different standard organizations in various forms similar to the CCITT so called 3 - stage method [1], [2], [3]. These specifications will soon be available on computer readable media. It is a large and "unnecessary" job to rewrite these specifications. We therefore decided that specifications received in a form similar to the so called 3-stage method are to be used in their original form. For entirely new applications the specifications must be written. This will be done by a reduced 3 - stage method. A template with automatic links from the overview to the document and from table of contents of the document to paragraph in the document gives support to writer of the specification.

2.2 Software Design

2.2.1 Call Models

Process Definition. A standard process concept has been chosen to model the run time structure, i.e. the Call Model of the application. Processes execute in a pseudo concurrent manner. A process becomes active due to an external stimuli / signal. After execution a process has a state corresponding to where in its code the process has suspended. This matches very well to the characteristics of our application, i.e. we have many parallel calls and each call triggers several sequence(s) of operations.

Process Application. Our ambition has been to divide the Call Model in the same way as the Functional Model of the specification is divided. The Functional Model has different entities for Access and for User (call handling). There is also a need to model line terminal devices. The this has led to the assignment of one process for:

each hardware device (*driver process*)

each type of line (*access process*)

each party in a call (*user process*)

Call Sides. Networking features and stand alone PBX features should be well matched. The different sides of the call should be represented as different entities already in the functional model used in specification work. It was therefore found suitable to choose a split view of

the call control. This means splitting a call into two sides each with its own set of processes. Another advantage of split call control is that the total number of states is significantly reduced.

Protocol Principles between Call Sides. In many cases, split call control requires negotiation / communication between the call sides before decisions are taken. The communication between the sides is supported by a high level protocol which hides message passing between processes.

2.2.2 Software Architecture
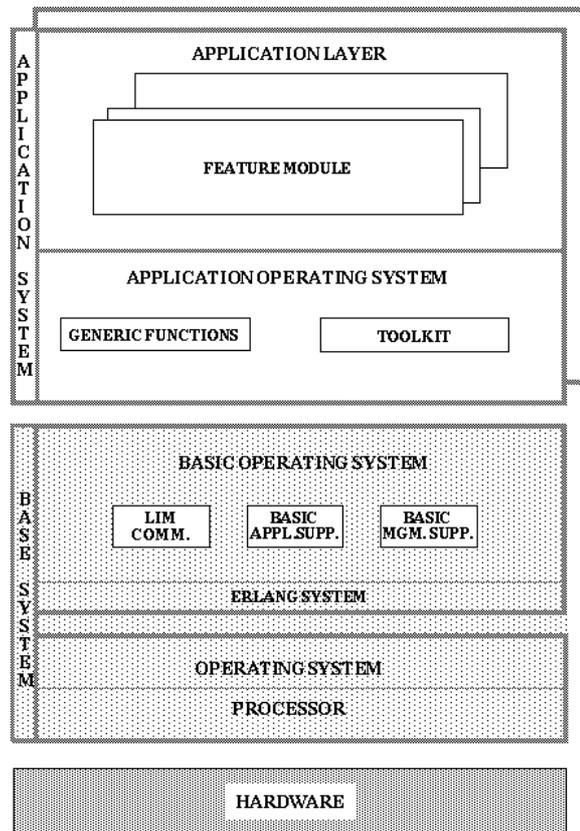
The system software architecture is divided into layers which has following the advantages:

The application becomes independent of the operating system selected.

The application becomes independent of both the CPU hardware and telecom hardware selected.

Distribution of processors is hidden from the application

A SWITCHING SOFTWARE ARCHITECTURE PROTOTYPE USING REAL TIME DECLARTIVE LANGUAGE

M Persson, K Ödling, D Eriksson

Ericsson Business Communication AB
135 83 Tyresö Sweden
Tel. 46 8 6824000, Fax. 46 8 7123925

ABSTRACT

In this paper we present our way of developing new PBX applications by making S/W prototypes. The prototype presented in this paper was designed using the Erlang programming language which has been developed in the Ericsson group. Erlang is a symbolic declarative language with built in real time support. The prototype uses a new software architecture. Well proven commercial operating system products and processors and well tested Ericssons PBX hardware provides a very robust test bed for new application S/W. About 10% of the functionality of an Ericsson PBX MD 110 has been implemented in the prototype. Designing the prototype has provided valuable experience and has indicated an increase in design productivity of more than 10 times compared to current mainstream S/W technology. The prototype demonstrates that it is possible to change S/W technology in a similar way as changing H/W technology. The prototyping tools are now being used for testing new solutions and new features.

1   INTRODUCTION

A large effort has been put into creating a new prototyping technology with the objective of developing and evaluating a state-of-the-art software architecture and a new programming language for telecommunication switching systems. This work has been done by the design of a real prototype and the conclusions of this work are based on demonstrations and measurements. The result of the work is a useful prototyping basis for the development of new applications. This initial work has the following characteristics:

State-of-the-art. The work is technology driven, i.e. the results of modern software research has been applied to telecommunication applications. The programming paradigm chosen is declarative programming. There is mounting evidence that this will be main stream programming technology in the 1990's. The operating system supports telecommunications applications, for example it supports per call fault recovery etc.

Market driven. Application experts have been involved in the design of the software architecture to simplify handling of sales objects throughout the entire development process. Features in the specification are mapped onto feature modules in the software architecture. This, in fact, introduces a market driven approach to the development process.

Prototyping. Prototyping is a method of work for solving very complex problems and to verify results by practical experiments. Prototyping is done in cycles each having its own defined objectives. A cycle contains a limited number of features which are fully implemented. Each cycle introduces new features.

Software Technology Base for Making Prototypes. The software technology for prototyping which evolved as part of the initial work is a new and stable base for building further prototypes. These prototypes represent realistic systems, although limited in their functionality. They have real time characteristics and S/W quality comparable to real products.