Open telecom platform

Seved Torstendahl

The open telecom platform (OTP) is a development system platform for building telecommunications applications, and a control system platform for running them. The platform, whose aim is to reduce time to market, enables designers to build – from standard, commercially available computer platforms – a highly-productive development environment that is based on the programming language Erlang.
The OTP also permits application designers who program in C, C++, Java and other languages to take full advantage of sourced components. Moreover, the OTP allows designers to consider costs when matching computer platforms with requirements for processing power and component availability.
The author outlines the OTP system architecture, its tools and building blocks, while describing the strengths of the open telecom platform as either a development environment or as a target system.

The open telecom platform (OTP) described in this article is primarily intended for new applications that combine a need for reliable, high-performance telecom characteristics with a need for using externally sourced hardware and software components. This includes ATM-products for access networks and data communications, such as Internet protocol-related traffic.

Multipurpose platform

The open telecom platform is a development system platform for building, and a control system platform for running, telecommunications applications. However, the OTP is not a monolithic platform, but is made up of sets of tools and building blocks, which include:
– Erlang – a programming language complete with compiler, debugger, and other development tools (as a rule, the tools and building blocks that are required for Erlang are already implemented in the language, Box D);
– SASL – systems architecture support libraries (SASL) contain functions for building fault-tolerant distributed applications;
– Mnesia – a real-time fault-tolerant distributed database management system (DBMS);
  – the Erlang run-time system;
  – sourced programs;
  – standard, commercially available operating systems;
  – computer hardware.

Different packages of OTP-related components have been defined to suit different user needs. Applications development packages, for example, contain all the tools and building blocks that designers need to develop applications for specific computer systems. Package types that are meant to be integrated into an embedded application system (target system) make up a basic part of that system and contain the building blocks whose functions are needed at run-time. Examples include a database, or a simple network management protocol (SNMP) agent.

OTP development environment
Many of the application programs for the OTP are written in Erlang. Application programmers will find that several tools in the OTP development environment support the tasks of developing and testing applications. For example, the OTP development environment contains:
– tools for translating into stub code the interface specifications given in the form of C header files, simple network management protocol-management information base (SNMP-MIB) definitions, and abstract syntax notation number one (ASN.1) definitions;
– an Erlang-to-Emacs mode that facilitates editing Erlang programs;
– the Erlang compiler and debugger for testing modules;
– a test coverage tool, and a hot-spot finder.

Some programs are written in other languages, including C, C++ and Java. In such cases, the application programmer uses the tools provided by the supplier of the target system, or the tools that are available in the development environment.

Target systems
An OTP that consists of a control system with one or more processors may be configured as a target system. If the system comprises more than one processor, then the processors must be able to communicate with one another through a logical network. Some processors, equipped with secondary storage and various I/O units, supervise and control the general system by means of functions in the systems architecture support libraries (SASL)—the OTP's central component.

A communication mechanism joins each processor to the distributed system. If desired – depending on requirements for reliability – the processors and the

Box A
Abbreviations

API	Application program interface
ASN.1	Abstract syntax notation number one
BOS	Basic operating system
BSD	Berkeley software distribution
DBMS	Database management system
HTTP	Hypertext transfer protocol
I/O	Input/output
IP	Internet protocol
JAM	Joe's abstract machine
MIB	Management information base
MMU	Memory management unit
OTP	Open telecom platform
RCS	Revision control system
RPC	Remote procedure call
SASL	Systems architecture support libraries
SNMP	Simple network management protocol
TCP	Transfer control protocol
UDP	User datagram protocol

communication mechanism may be duplicated. The OTP handles all the basic telecommunications requirements for the control system (real-time, fault-tolerance, live software upgrades, distribution), thereby allowing application designers to fully concentrate on the unique aspects of their own work. Moreover, the OTP can easily be ported to several different commercial operating systems.

System architecture

All application systems that are built on the open telecom platform (OTP) adhere to a basic architectural structure, Figure 1.

Bottom layer
Commercial computer systems make up the bottom layer. The system hardware in this layer may also be designed in-house, if the manufacturing cost and the volume of delivery can be justified. This is merely an architectural view; in real systems, the bottom layer contains many computers which may be of different types.

Middle layer
Support for telecommunications requirements is provided by a robust real-time distributed database management system (DBMS); basic support for handling software and reporting events; an extensible SNMP agent; a Web server; and a library of routines for interworking between applications written in C and Erlang.

Top layer
All applications have access to Mnesia and SASL. The SNMP agent and the Web server may also invoke functions that are provided by the applications in this layer.

Interfaces
Three interfaces are provided: an interface to OTP software; an interface to the operating system; and an interface between applications written in different languages. In terms of logic, the third interface applies to the application level, but is implemented in the OTP and in the operating system.

Application programs
The OTP includes a set of application program interfaces (API), as well as rules and guidelines for writing application programs. It also includes teaching materials, documentation standards, and sample programs that show how application programs are designed.

Most programs are written in Erlang. However, application programs for time-critical parts may be written in C.

Sourced programs
The OTP defines how sourced programs, which include protocol stacks and management applications, may be incorporated into a system and made to interwork with programs that were developed by application programmers.

SASL
The systems architecture support libraries (SASL) contain basic software that supports system start/restart, live system software updates, and process management. The basic operating system (BOS)—a predecesor to SASL – was used for several years in the Mobility Server and other systems.

Mnesia
A real-time fault-tolerant distributed DBMS that supports fast transactions for the telecommunications application, and a query language, called Mnemosyne, for handling complex queries.

SNMP support
SNMP provides run-time support through an extensible agent, and development support by means of agent/sub-agent design principles and an MIB compiler.

Web server
The Web server permits data that refers to Erlang functions to be collected via Web pages.

Erlang run-time system
The basic system that supports the execution of Erlang programs. The Erlang run-time system includes the Erlang abstract machine, which executes intermediate code, the kernel, and standard libraries.

Computer platforms
The operating system and computer hardware consist of standard commercially available systems. Testing environments, for example, use conventional workstations.

OTP component packages are defined to suit various purposes. Devel-

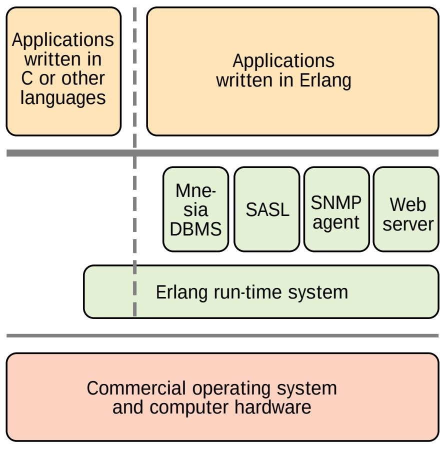

Applications written in C or other languages
Applications written in Erlang
Mne-sia DBMS
SASL
SNMP agent
Web server
Erlang run-time system
Commercial operating system and computer hardware

Figure 1
The OTP system architecture.

Box B
Reducing time to market

Shortening time to market entails much more than simply reducing the time it takes to write program code. The time spent on the other phases of a project must also be reduced.
One way to do this is to have program designers begin their work at the early stages, or at high levels, of design – for example, starting with the functional specification. Moreover, design phases can be broken down into short incremental steps, where each step is fully integrated into and tested at the level where it will be implemented. The initial step, for example, might consist of starting all blocks. Next, critical parts can be added. Finally, management functions and error handling can be added. This way, problems can be detected and corrected early on.

Box C
Getting the most out of Erlang's easy-to-port functionality:

In any part of an application written in C, do not include functions that relate to a specific operating system. Otherwise, gather these parts into a cluster, or else mark them clearly.
Recommendation:
Use the C compiler that is used for the Erlang run-time system.

Table 1
Operating systems and computer platforms on which the OTP may be run

Microprocessor architecture	**Operating system**
Intel Pentium	Linux, Solaris 2, Windows 95, Windows NT
Motorola 680X0	Vx Works
Sun SPARC	Solaris 2

opment packages include software for developing Erlang programs, as well as libraries and applications. The user organisation purchases the actual computer platform, such as a Sun workstation with the Solaris operating system, a PC with a Windows operating system, or some other computer system for which an OTP development environment is available.

Packages for embedded systems include libraries and applications, but do not contain the compiler and debugger. These packages also include references to agreements that have been reached with vendors of computer systems. Under the terms of these agreements, various licensing fees are paid automatically when specific brands of hardware are procured, or when specific versions of software are copied.

Strengths of the open telecom platform

The open telecom platform (OTP) runs on many computer platforms, and caters to applications that have been designed in Erlang as well as in standard programming languages. Moreover, the OTP is designed to support typical telecommunications requirements for robustness, smooth software upgrades, distribution, and real-time functionality.

Time to market

A major objective of the OTP is to provide a highly productive environment for designing telecommunications applications. This objective is supported by:

– the Erlang programming language – a high-level functional language that supports distribution, fault detection, and recovery;
– building blocks
  • robust, real-time distributed DBMS;
  • support libraries for creating applications that can report errors, restart themselves when errors occur, and update themselves with new versions of software when ordered to do so;
  • an extensible SNMP agent and a Web server that is closely integrated with the database.

Multiple computer platforms; easy to port; up-to-date computer technology

To accomodate different microprocessor architectures and operating systems, several versions of the OTP exist, see Table 1.

The effort required to support many different operating systems is manageable, mainly because most support for non-Erlang languages is obtained directly from the vendors of computer systems. Only a minor part of the Erlang run-time support is dependent on the computer platform. Most Erlang run-time support, and all parts of applications that are written in Erlang, are independent of system hardware and related operating systems. Moreover, "Joe's abstract machine" (JAM) code is directly executable on any platform provided that the Erlang code does not explicitly make use of a specific feature of the operating system. Applications written entirely in Erlang are easily ported, and in most cases their load modules (compiled code) are directly executible on other computer systems.

Hardware and software from external suppliers

Sourced hardware and software play a decisive role in reducing time to market. Very often, software vendors can offer special applications at prices far below the cost of developing an in-house solution. Likewise, because computer systems change rapidly, rather than investing considerable resources to develop a computer board, Ericsson can offer a wide range of attractive solutions that are based on existing, commercially available systems. To this end, the Ericsson external technology provisioning process is used to enforce competitive terms for external hardware and software, and to forge ties with reliable vendors.

Software

Software from external vendors falls into three categories:

– Applications – which include the upper layers of protocol stacks.
– Operating system (OS) kernels.
– Software that is directly coupled to hardware or to the OS kernel – for example, device drivers and protocol stacks.

To handle support and maintenance, sourced software should be imported as object code. Because malfunctions can cause a system to crash, operating system software and software that is directly coupled to hardware or to the OS kernel must be thoroughly tested and proven reliable. Some protection may be provided for applications by running them as

Box D
The Erlang programming language

Background
When introducing new technology, real-time systems often lag behind other systems. Indeed, many real-time systems are written in the programming language C. Some languages, however, have been developed specifically for programming concurrent real-time systems. Some of the best known examples are Ada, Modula2 and PLEX.
Today, declarative programming languages such as Prolog or ML are used for a wide range of industrial applications. These languages drastically reduce the total volume of source code in applications, as well as the efforts that are required to design and maintain them. However, declarative languages were not primarily designed to be used in concurrent real-time systems.
Erlang, which is a small but extremely powerful language for programming concurrent real-time, fault-tolerant distributed control applications, combines important attributes of declarative languages with constructs for supporting concurrency, distribution, and error-detection. It is an expressive, high-level functional programming language without pointers – a feature that greatly simplifies design and testing.

Built on experience
Erlang has been in use at Ericsson for more than five years. To date, many hundreds of thousands of lines of Erlang source code have been written, demonstrating the language's suitability for use in large projects. A key to achieving very high productivity, Erlang has been used for some years in several Ericsson products, including Mobility Server, ISO Ethernet, and NETSIM.

Functions, modules and processes
Erlang programs are made up of functions that have been grouped into modules and packaged as software products. Functions spawn processes – the executing elements of an Erlang system – that are very lightweight and that enable fine-grained concurrency.
Processes communicate by sending and receiving messages. Communication with the external non-Erlang world is conducted through ports (which behave like processes). Processes may also be linked to each other in order to detect software errors.

A built-in distribution mechanism enables designers to create a system whose processes may run on different computers. Erlang allows fault detection and recovery in a distributed system, and the OTP software implements customisable schemes for recovery after faults.

Reference:
"Concurrent Programming in Erlang" by Armstrong, Virding, Wikström and Williams.

Very high-level functional/declarative language
Symbolic data representation
Support of massive lightweight concurrency (parallelism)
Support of designing distributed, non-homogeneous systems
Permits tailored-to-fit fault recovery schemes in distributed systems
No pointers, no memory leaks
No fixed sizes or limits
Easy to interface other software and hardware
Permits software to be updated while running
Modular concept for structuring applications
Easy to create reusable libraries

Figure D
Characteristics of Erlang.

separate processes, preferably with memory management unit (MMU) protection. Application software is best managed through its source code, where software management requirements can be adapted.

Hardware
The OTP is designed to use different microprocessor architectures and computer boards. For example, standard workstations or PCs are used for development, and standard computer boards are normally used for inclusion in embedded (target) systems.

Computer technology is evolving very rapidly. Every nine months, PC vendors design a new generation of boards with faster processors, more and faster memory, and new integrated I/O circuits. Competing at this pace with in-house solutions would be very costly for Ericsson. However, by relying on external processor hardware, Ericsson can transfer their need for staying abreast of developments to computer vendors. Other advantages of using standard computer boards in the OTP are that every new board, from low- to high-end, can be made available, and that development costs can be shared with the rest of the market. Thus, Ericsson draw on the expertise of external vendors, while remaining focused on their own core areas of business.

Erlang and other languages
Applications that use the OTP can be implemented in Erlang or in any other programming language. The choice of language is governed by the characteristics one hopes to derive.

Programming in Erlang shortens development time and provides support for designing robust distributed real-time applications. Support is provided in the form of
– libraries of ready-to-use components;
– guidelines for using the components;
– guidelines for designing the applications that provide desired characteristics.

By means of careful system design, and by applying the “90/10” rule (fine-tune the 10% of Erlang code that occupies 90%

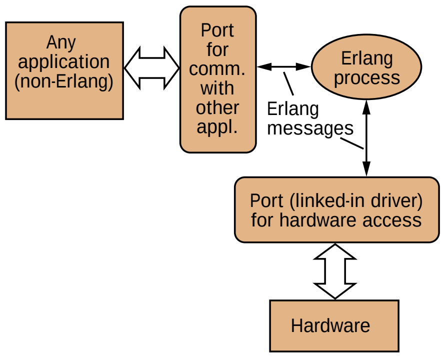

Any application (non-Erlang)
Port for comm. with other appl.
Erlang process
Erlang messages
Port (linked-in driver) for hardware access
Hardware

Figure 2
The port mechanism in Erlang.

of processing time), it is possible to obtain good run-time performance from Erlang.

Development in other programming languages (usually C) is sometimes necessary for reducing execution time. Highly optimised C code is very efficient, and may be required in time-critical parts of applications. The OTP offers different ways of integrating applications written in C into applications written in Erlang or in mixed programming languages.

One reason for using C is that a market already exists for applications or components written in that language, including applications for implementing communication protocols. Obviously, being able to use existing products is a desirable feature. Thanks to a component in the OTP and to an adaptation unit written in C, these parts can be integrated into the software management principles used by the OTP.

There are several ways of creating interfaces between applications written in Erlang and other programming languages:
– through the port mechanism;
– using the socket library;
– through the C/C++ interface generator;
– by enabling C programs to imitate an Erlang node.

The port mechanism
According to the port mechanism, which is the standard mechanism for interfacing applications, Figure 2, a port on the Erlang side of the interface is perceived and behaves as an Erlang process. Here, all communication consists of standard Erlang messages. Software units written in another language perceive these messages as communication lines to another program. Ports may also be used for accessing hardware or low-level features directly.

The socket library
Delivered as part of the standard Erlang distribution, the socket library is a low-level mechanism that enables non-Erlang processes to communicate with Erlang processes by means of the transmission control protocol/Internet protocol (TCP/IP). The Erlang program is responsible for understanding and complying with the protocol that is used on top of the TCP.

The C/C++ interface generator
Thanks to the C/C++ interface generator, functions written in C and Erlang can easily communicate with one another. The interface generator includes conversion routines on both sides of the interface for encoding and decoding transferred data. This enables applications written in Erlang to access and manipulate C data structures. To use the conversion routines, some programming is required on the C side of the interface. All underlying communication is conducted using the port mechanism, or the socket library.

C programs that imitate an Erlang node
Erlang data can be encoded to byte sequences in a format known as the Erlang external format. A library of useful functions has been provided to enable programs written in C to decode byte sequences of this kind. The library has been extended to facilitate communication with a distributed Erlang node using the Erlang distribution protocol. Because they behave like an Erlang node, C programs that decode byte sequences of Erlang code are called C nodes. C nodes, which have very limited functionality, are not visible to Erlang, and are therefore said to be hidden.

OTP support for high reliability
An Erlang system that is run as a task by a host operating system is called an Erlang node. In the OTP, processes in different nodes communicate as easily with one another as processes within the same node. Erlang processes may also be linked together. Should one of the processes die, due to an error, then a sig-

Box E
Reliability

A key requirement of the OTP is to provide a robust environment for implementing fault-tolerant telecommunications applications. By robust environment is meant a system that provides specified services despite the presence of hardware faults or software errors. The minimum requirements of a robust environment are error detection, damage assessment, and error correction. Each of these aspects is necessary for meaningful error-recovery at run time.
Other requirements of the control system are as follows:
– The “outage” time (that is, the time during which services are unavailable) must be short.
– While error recovery operations are under way, only a minimum of services may be disrupted. This implies that some kind of "hot" or "warm" standby solution must be used.

Hardware considerations
Hardware is made fault-tolerant by duplicating resources; for instance, by means of redundant CPUs and dual communication paths. The provision of a redundant CPU structure is more important than synchronous duplication of CPUs, since identical CPUs executing in parallel do nothing to guard against software errors – that is, given the same error, both machines would fail at the same time.

Software considerations
Reliable software systems are built by carefully structuring the system so that an error in one part of the system may not affect its other parts.

nal is sent to each process that is linked to it. Since error signals can be trapped by supervisory processes, highly reliable layered systems can be designed.

The Erlang programming technique of dividing computations into "supervisory" and "worker" roles can be employed to build a robust system architecture. SASL provide patterns that facilitate design according to these rules.

The system structure consists of a critical "safe kernel" that must always be correct, and an application area where the requirement for correctness is somewhat less stringent. The safe kernel is provided by SASL.

Erlang has a real-time garbage collector, very few operations with side effects, and no pointers. Thus, when Erlang is used, a large class of problems commonly associated with programming real-time systems is eliminated.

Updating sourced software
Ordinarily, operating system software and device drivers can only be replaced by rebooting the processor. Thus, where continuous operation is a requirement, system software may only be updated in systems that make use of multiple processors. In a multiple-processor system, the system software is replaced on one processor while the other processor continues to execute as usual.

By contrast, even in single-processor systems, individual applications can usually be replaced while the system is running; however, the applications themselves must typically be terminated, updated, and then restarted. Again, if this method is not acceptable, then a multiple-processor system may be necessary.

Tools in the OTP

In this context, tools are programs that are used to develop software. A brief description of the tools included in the OTP follows below.

Application monitoring.
The appmon program has two main parts: the node window, which shows an overview of the applications on all known nodes; and the application window, which shows the process tree of each application.

Because both windows run on one node (called the server node), graphics need only be installed on one node. The monitoring programs then use small information agents on each monitored node.

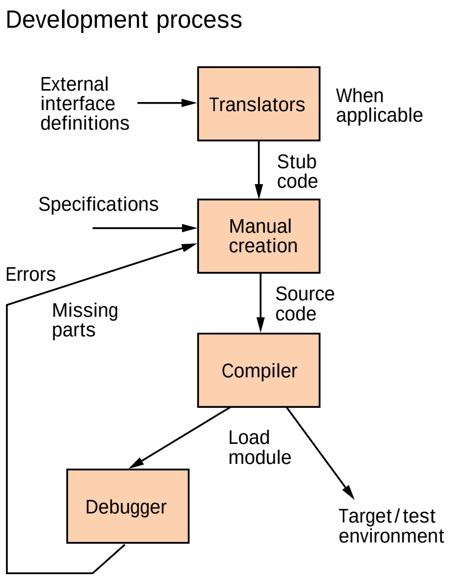


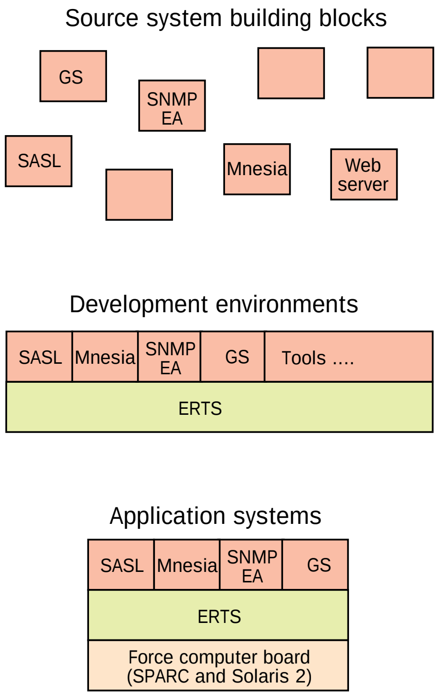
Figure 3
Applications may be developed from interface definitions (C header files, ASN.1 specifications, CORBA/IDL interface definitions). After the source code has been written, compiled and tested, it is reiterated until satisfactory results are reached.

ASN.1 compiler
The ASN.1 standard, which is used for describing data representations in communication, defines unambiguously the interpretation of transferred data units. An ASN.1 compiler is provided for translating interface definitions in ASN.1 notation into stub code together with calls to translation routines for packing and unpacking transferred data.

C interface generator
This tool facilitates communication between programs written in C and Erlang. An interface specification received in the form of a header file is translated into stub code. The stub code may then be used for calls between Erlang and C to convert data representations.

Compiler
The Erlang compiler translates Erlang source code (text files) into object code that is independent of the CPU in use. This means that the resulting code may be loaded onto any computer platform

Figure 4
After building blocks of the OTP are compiled, a selection of generated load modules are combined to form a platform – for example, to form a development environment to be used on a Sun SPARC with Solaris 2, or to form an application system that includes the computer platform. Likewise, a project may define its own OTP application system.



that supports Erlang without the code having to be changed or recompiled.

Coverage tester
The coverage tester is used to ensure that all code in a module has been executed during a test.

Cross-reference tool
The exref tool is an incremental cross-reference server that builds a cross-reference graph for every module that is loaded into it. A great deal of information, including graphs and module dependencies, may be derived from the cross-reference graph. The call graph is represented as a directed graph in text.

Debugger
The Erlang debugger provides mechanisms for visualising what happens when code is executed in specified modules, or when processes crash. Because it allows breakpoints to be set, and because it performs sophisticated tracing, the debugger interferes with the behaviour of the system being debugged.

The debugger is mainly used to locate errors in code (bugs). However, designers may also use it to learn about, or to better understand, applications that were written by other designers.

Because Erlang is a distributed concurrent language, conventional debugging techniques do not apply. Thus, the Erlang debugger provides mechanisms for attaching to, and interacting with, several processes simultaneously, including local processes and processes located in other Erlang systems in a network of Erlang systems. Every process that runs code in debugged modules is monitored, and continuous information on the status of any given process may be displayed.

Erlang mode for Emacs
Emacs is a widely-used text editor whose behaviour can be customised. A definition file is distributed with the OTP that allows the Emacs editor to recognise Erlang syntax, and that helps designers to write well-formatted, syntactically correct code. The Emacs editor is not included, but may be obtained free of charge from many sources on the Internet.

Graphics interface for Erlang
The graphics system is a general graphics interface for Erlang. The interface is easy to learn, and may be ported to many different platforms. If future applications in Erlang are written to the same graphics API, then it will be possible to run each supported platform without having to change a single line of the application code.

Profiler
The profiler is used to determine which parts of code, "hot spots", occupy the most CPU time. This tool is useful for optimising programs.

SASL
The systems architecture support libraries (SASL), which are used as building blocks by applications that are running, provide usage rules for designing robust applications that may be started, stopped, and restarted, and that can report errors or events.

SNMP MIB compiler and instrumentation
The OTP provides support for using the simple network management protocol (SNMP) for operating and maintaining applications on the platform. SNMP support consists of
– run-time support – in the form of an extensible agent;
– development support – in the form of a management information base (MIB) compiler and a programming model for implementing MIBs.

The extensible agent uses two mechanisms – dynamic MIB loading and subagent handling – that provide an environment where MIB modules can be loaded / unloaded in an efficient plug-and-play fashion. The sub-agent concept also supports distributed applications; that is, different MIBs can be implemented at different nodes. If necessary, the transaction mechanism for SNMP set requests may be customised.

The MIB compiler may be used to generate the prototype instrumentation of MIBs automatically. The results may then be incrementally refined and tuned. This feature enables designers to start developing manager applications at the early stages of a project.

Support is also provided for using the Mnesia DBMS together with the SNMP tool kit. This means that Mnesia tables can be read and manipulated with SNMP, and that an SNMP table can be implemented as a Mnesia table. The

SNMP tool kit provides an API for implementing application-specific MIB modules, including a mechanism for sending traps.

Trace tool
The debugger-tracer has a graphical front-end. When used in a running system, it causes little disturbance to the system. Trace tool supports distributed debugging, which means it may be used to debug Erlang processes running on the same system as well as processes that are running on several remote systems. The remote systems may each run on different operating systems and CPUs. Also included is a simple line-oriented interface to the trace functions.

xerl
A complete integrated Erlang environment, xerl is an X Window-based interface to the standard Erlang shell. The interface may be used to compile, edit, debug and run Erlang. Moreover, because xerl is sensitive to Erlang syntax, it checks and formats code as it is written.

yecc parser generator
A parser generator for Erlang that is similar to yacc. From a grammar definition (input), the generator produces Erlang code for a parser.

Other tools
At present, tools for managing different versions of software are not delivered as a part of the OTP. Nonetheless, a management tool, such as the revision control system (RCS) or ClearCase, is highly recommended.

Building blocks in the OTP

When the OTP is used to create an embedded system, some of its components are retained by the delivered system. OTP run-time applications are also available as building blocks in the development environment (Figures 3 and 4).

Erlang virtual machine
The Erlang virtual machine runs on top of a host operating system. The Erlang run-time system frequently runs as a single process in the host operating system. The Erlang virtual machine provides the following support for Erlang programs:
– Consistent operating system interface on all platforms.
– Memory allocation and real-time garbage collection, which effectively eliminates memory leaks.
– Lightweight concurrency and support of thousands of simultaneous tasks.
– Transparent cooperation between all computers in the system.
– Location and encapsulation of run-time errors.
– Supervision of run-time code as it loads or is replaced, and while it is linked.

The structure of the Erlang virtual machine allows it to be easily incorporated into new operating systems. The Erlang virtual machine is the only building block that was not written in Erlang.

Kernel
The kernel is always the first application to be started. In a minimal OTP system, the kernel is one of only two applications. The kernel application contains the following services:
– application_controller
– auth
– code
– error_logger
– file
– global_name_server
– net_kernel
– rpc
– user

Box F
Components of supporting design of robust applications

Component	**Description**
Standard libraries	The standard libraries contain Erlang modules, such as lists, “ordsets”, and “dict”. An important property of these libraries is that the code they contain is free of side effects. Thus, calls to functions in these modules work the same in every environment.
Behaviour	In this context, behaviour is a formalised pattern of design that can be reused to build new applications. Behaviour is implemented in the form of call-back modules: a call-back module must export a specific set of functions that are called by the system to evoke a particular pattern of behaviour. Examples of behaviour are: - gen_server – a behaviour that implements a generic client-server machine; - gen_event – a behaviour that implements a generic event-handling machine; - supervisor – a behaviour that implements a specific style of supervisor-worker programming which allows supervision trees to be constructed dynamically.
Applications	Applications are named collections of objects that are used for encapsulating system components. The basic properties of an application are that it can be named, loaded and unloaded, as well as started and stopped. The aim of an application is to provide a mechanism that enables users to divorce themselves from the internal details of the application, and to think instead about the relationships between applications.

The kernel application also includes standard *error_logger* event handlers.

Mnemosyne – the query language

The Mnesia DBMS is organised as a basic layer that takes first-order predicate logic with Erlang data types as values. Recursion, negation, and a simple constraint system are also supported. The evaluation is set-oriented. In practice, users have a choice of several query shells in which to type their queries and receive their answers.

Views of data are displayed according to the rules defined in the modules. Each module corresponds to a file and is compiled and stored in the database itself. The modules and rules are declared in a schema. In some instances, even when the table is used in the module, the schema may be changed by simply recompiling the module.

Mnesia real-time DBMS

In most applications some data must (a) be stored safely and (b) remain easy to access. Requirements vary a great deal; for example, the amount of data may range from a lot to very little. On the one hand, the loss of data from a system crash may be acceptable, whereas in other instances, data must survive practically any kind of system failure. Similarly, some applications require near-instantaneous access time – perhaps only fractions of a millisecond – whereas in other instances a longer access time may be acceptable. Moreover, some applications require a valid prediction of the access time – real-time access.

In short, despite a broad spectrum of requirements, it must be easy to update data without introducing inconsistencies, especially when several corresponding changes are made. Also, for operations and maintenance purposes, operators must sometimes be able to access data by means of complicated and non-standard queries.

In response to these requirements, the Mnesia DBMS was developed with the following basic properties:

– Data is accessed in two basic ways:
  - through an API for programs;
  - using a query language, called Mnemosyne, for humans
– Read and write access is protected by means of transactions. This method gives each user (program or person) the impression that he is alone in the system. It also bars inconsistencies from being introduced into the data (status control) when several corresponding changes are made.
– Data may be distributed transparently over several Erlang nodes. Important data is copied to several nodes; less important data may reside at only one location. The distribution of data over nodes cuts access time drastically.
– Data is declared to reside in RAM, on secondary storage, or both.
– If a hardware or software node crashes, the data that it contained may be reconstructed from redundant copies of the data stored at other nodes, or from logged updates. Assuming that the data at a particular node is replicated at other live nodes, then the addition or removal of that node will not be observed by users in the system.
– Data is organised in tables. This provides a sound theoretical basis, and a broad range of well-known, proven methods for modelling the data.
– The tables, their organisation, location, storage type, and other aspects are declared in a schema. Views of data are supported from the API and from the query language.
– An optimising query-language compiler and evaluator is available for use by operators and in the API.

OS monitoring

The OS monitoring application defines the following services:

– disksup – which checks the available disk space and sends an event if a stated threshold has been passed;
– memsup – which checks the available primary memory and sends an event if a stated threshold has been passed.

Read, write and search from the API

From the basic API, a program – with or without a transaction – can read and write table entries or merely search a table. Multiple, related accesses that involve writing without a transaction are strongly discouraged, since they can severely damage data. Nonetheless, this option has been provided to give immediate access to experienced programmers who need to perform concurrency control themselves. Each time the data is read without a transaction a snapshot is taken.

A query language has been provided to facilitate complicated queries, such as when data from many tables needs to be combined. The query language, which is mainly based on first-order logic, is accessed through list comprehension. A list comprehension is a functional language construction that is well-suited to the Erlang programming language.

SASL
The systems architecture support libraries (SASL) are designed for building embedded real-time Erlang applications, and include a set of standard design templates that can be used to solve common programming problems on application start, restart and supervision.

SNMP extensible agent
The OTP facilitates use of the SNMP for operating and maintaining applications on the OTP. SNMP support consists of run-time support in the form of an extensible agent, and development support in the form of an MIB compiler and a programming model for implementing MIBs.

The extensible agent uses two mechanisms that provide an environment where MIB modules can be loaded / unloaded in an efficient plug-and-play fashion. These mechanisms include dynamic MIB loading and sub-agent handling.

Sockets
Sockets are the Berkeley software distribution (BSD) UNIX interface to communication protocols. Various protocols may be accessed through sockets. The socket module provides an interface to the BSD UNIX sockets. The udp module supports user datagram protocol / Internet protocol (UDP / IP) sockets.

Standard modules and libraries
The Erlang programming environment contains several standard reusable software modules. The functional program paradigm on which Erlang is based greatly facilitates reusing software.

Many standard modules are specially adapted to the needs of concurrent distributed systems. The remote procedure call (RPC) module, for example, allows designers to program a remote procedure call in one line of source code. Examples of standard modules are lists, “ordsets”, “gen_server” and “gen_event”.

Web server
The Web server is a hypertext transfer protocol (HTTP) daemon implemented in Erlang. Access to the server looks up a requested Web page and sends it to the browser. The page may fetch data from a database table or from a function call. Integration is very efficient.

Conclusion

The open telecom platform (OTP) enables designers to build and run telecommunications applications on a broad range of standard, commercially available hardware and software platforms. The OTP also allows designers who program in C, C++, Java and other languages to integrate sourced components – protocol stacks, APIs, I / O units and drivers – into their applications.

The OTP comes with an exhaustive collection of tools and building blocks, such as the programming language Erlang, systems architecture support libraries (SASL), a real-time database management system (DBMS), an extensible SNMP agent, and a Web server.

As a design environment, the main strengths of the OTP are: time to market; compatibility with many different computer platforms and sourced components; up-to-date hardware and software technology; and reliability. What is more, the OTP permits designers to consider costs when they match computer platforms with requirements for processing power and component availability.

As a target enviroment, the OTP meets all basic telecommunications requirements. It has a real-time distributed control system that is fault-tolerant, and that can handle software upgrades while it is running. In addition, the OTP can easily be ported to several different commercial operating systems.

Box G
Sockets

A socket is a duplex communications channel between two UNIX processes, either over a network to a remote machine, or locally between processes running on the same machine. A socket connects two parties, the initiator and the connector. The initiator is the UNIX process that first opens the socket. It issues a series of system calls to set up the socket and then waits for another process to create a connection to the socket. When the connector makes its connections, it also issues a series of system calls to set up the socket. Each process then continues running. The communications channel is bound to a file descriptor that each process uses for reading and writing.

References

1 Däcker, B.; Erlang – A new programming language. Ericsson Review 70 (1993:2), pp. 51-57.