



Proceedings of the Fifth International

Erlang/OTP

User Conference

September 30, 1999, Älvsjö, Stockholm





Erlang User Conference 1999 - Programme

Prof Bengt Jonsson, Uppsala University, chairman

- 08.00 *Registration*
- 09.00 Welcome Address
Bjarne Däcker, Ericsson Utvecklings AB
- 09.15 Telia CallGuide
Hans Nahrngbauer, Telia Promotor AB
- 10.00 Use of Erlang/OTP in the Brainpool M/3 Communication System
Fredrik Ström, Brainpool AB
- 10.30 *Coffee and Demos*
- 11.00 Status Report on the ETOS Erlang to Scheme Compiler
Patrick Piché, Université de Montréal
- 11.30 Mail Robustifier Product based on Erlang/OTP
Håkan Millroth, Bluetail AB
- 12.00 Hatchet
Per Bergqvist, Ericsson Radio AB
- 12.30 *Lunch*
- 14.00 A Modular WAP Reference Stack Protocol Implementation
Johan Blom, Ericsson Wireless Internet AB
- 14.30 An Experimental SIP Implementation in Erlang
Hans Nilsson, Ericsson Utvecklings AB
- 15.00 Coming Releases of Erlang/OTP
Magnus Karlson, Ericsson Utvecklings AB
- 15.15 *Coffee and Demos*
- 16.00 Towards an Event Modelling Language
Maurice Castro, SERC
- 16.30 Proposals for and Experiments with an Erlang Bit Syntax
Claes Wikström, Bluetail AB
- 17.00 *Bus Transfer to Waxholm III*
- 18.00 *Conference Dinner*
- Demo Brainpool M/3 Communication System
Fredrik Ström, Brainpool AB
- Demo HACT - High Availability Computer Telephony
Stefan Björnelund, Ericsson Utvecklings AB
- Poster Erlang Verification System
Thomas Arts, Ericsson Utvecklings AB
- Poster HiPE - High Performance Erlang
Mikael Pettersson, Uppsala University



Fifth International Erlang/OTP User Conference

Stockholm - September 30, 1999



- Location:** Älvsjö Conference Center, Ericsson, Götalandsvägen 230, Stockholm, Sweden.
- Date:** September 30, 1999. Registration opens 08.00, programme starts at 09.00.
- Fee:** 950 SEK (including Swedish V.A.T). This includes printed material, lunch, coffee and conference dinner.
- Registration:** Credit card: (VISA or MasterCard). Please mail or fax a signed copy of the registration form. The receipt will be returned at the registration.
- Invoice: Please fax a copy of the registration form or send a mail to euc99@erlang.ericsson.se stating the address where to send the invoice.
- Registr form:** http://www.erlang.org/invitation_euc99.html
- Conference adm:** euc99@erlang.ericsson.se
- Conference scope:** This is the first Erlang/OTP User Conference since Erlang/OTP was made available "open source" and the first part of the conference will be spent on describing exciting new applications. The second part deals with new technical developments primarily in the area of protocol implementations.

Conference Programme

~ Registration ~

Telia CallGuide
Hans Nahringsbauer, Telia Promotor



Telia Promotor shares their experiences from the development and maintenance of a successful commercial Computer Telephony Integrated (CTI) system based on Erlang/OTP. Their experiences are accumulated from more than one year of different customer installations.

The core of Telia CallGuide is a CTI-server developed in Erlang/OTP, operating on a Windows NT Server. Telia CallGuide is the next generation CTI-system, which integrates telephony, email, fax and IP telephony, with far more functionality than the first generation of CTI-systems based on PBX technology.

**The use of Erlang/OTP in the Brainpool M/3
communication system**
Fredrik Ström, Brainpool AB



Brainpool AB decided during 1998 to make a new version of the Brainpool MMS product due to new customer demands and ideas about how to exploit Erlang/OTP on the NT platform. The MMS is a software server that is used to send text messages to GSM/SMS and pagers from client software in a computer network. The communication server uses one or several modems or X.25 circuits and is preconfigured to communicate with service providers in the Nordic countries. New demands for the new version include scalability, robustness, the ability to run on multiple computers, dynamic load balancing, two-way communication and country independence.

Brainpool chose to use Erlang/OTP for the development of the new generation of the product - the Brainpool M/3. We based this choice on what this platform promised, and on earlier experience using the Erlang language. The M/3 sends SMS messages through an Ericsson GM12 GSM module, which makes the M/3 easy to deploy in different countries. This also allows mobile phones to send messages to the server and connected computer applications, thus permitting two-way communication.

This presentation discusses why Erlang/OTP was chosen and our experience with this platform. We also plan to make a demonstration of the Brainpool M/3 communication system.

Status report on the ETOS Erlang to Scheme compiler
Patrick Piché, Université de Montréal



The ETOS compiler for Standard Erlang has been under development for the past two years at Université de Montréal. The current state of the compiler (including benchmarks) and future plans will be discussed. We will focus on the compilation approach which allows it to generate high-performance executable code, in particular: pattern matching compilation, tail recursion in C, real-time GC and native code generation.

~ Coffee ~

Mail robustifier product based on Erlang/OTP
Håkan Millroth, Bluetail AB

BLUETAIL

Bluetail Mail Robustifier is a software product that makes handling of e-mail more efficient for Internet Service Providers. The product was released on July 1, 1999 and is in now in operation at large Internet Service Providers in Sweden. In this presentation we focus on the role of Erlang in the development of this product: how it affected time-to-market, software quality, etc.

Hatchet

Per Bergqvist, Ericsson Radio AB

Hatchet or MicroMTX (the first AXE 110 application) is the smallest AXE 10 application and it is based on only two boards. One runs the AXE CP-software (written in Plex) using an emulator (SIMAX) and the other consists of SwitchBoard plus daughter boards, a complete switch with processor, which emulates the AXE 10 hardware. SwitchBoard is scalable and controlled through Distributed Erlang.



The talk will focus on experiences from using Erlang to create this system and to tie all the different subcomponents together

~ ~ ~ **Lunch and Demos** ~ ~ ~

A modular WAP reference stack protocol implementation

Johan Blom, Ericsson Wireless Internet AB

ERICSSON

This presentation focuses on a reference implementation of a subset of the de facto standard, Wireless Application Protocol (WAP), for wireless information and telephony services on digital mobile phones and other wireless terminals.

In this project a client and server stack, including a session (Wireless Session Protocol, WSP), and transport (Wireless Transport Protocol, WTP), have been implemented in Erlang. In addition, the basic management functions for starting and stopping the stacks etc. have also been implemented. The protocol stacks have been designed in a very modular way and as a result they have been used in the development of a number of different projects.

An experimental SIP implementation in Erlang Hans Nilsson, Ericsson Utvecklings AB



SIP - Session Initiation Protocol - is a new IETF Protocol for initiating, modifying and terminating multimedia conferences. One simple example is so called Computer Telephony. The protocol is text based (like HTTP) and is carried by either UDP/IP or TCP/IP.

This talk presents why Erlang/OTP was chosen as the basis for the experiments, how the software was organized and some of the results.

Coming releases of Erlang/OTP Magnus Karlsson, Ericsson Utvecklings AB



Presentation of the Erlang/OTP R6 release which will be available on October 27 and includes for example a new compiler for BEAM, ASN.1 extended standard support, Corba transactions and security in Corba using SSL. In addition there are many improvements for example in release handling.

This release will become available as 'open source' shortly after.

~ Coffee ~

Towards an event modelling language Maurice Castro, Michael Dwyer and Geoff Wong, SERC RMIT



Object-oriented programming owes part of its popularity to Booch's notation, Rumbaugh's OMT, and UML. These notations allow object oriented designs to be expressed graphically and furthermore have enabled the development of case tools for object oriented languages.

The lack of a suitable high level graphical notation has been identified as one of the factors discouraging the uptake of functional programming and in particular the language Erlang.

This paper represents a first step in designing a graphical modeling language for functional programming that encourages sound programming practices. The initial target language is Erlang but it is hoped that the notation can be extended to other functional languages.

Proposals for and experiments with an Erlang bit syntax
Claes Wikström, Bluetail AB



Efficient programming of communication protocols like CCITT SS7 requires above all:

- Some method to handle concurrency,
- Some method to describe state-machines,
- Timers,
- Some way to efficiently and beautifully encode and decode PDUs
- An efficient way to manage buffers and avoid unnecessary copying.

At present Erlang is excellent with regard to the first three aspects and can lead highly declarative programs running fast. The so called "bit syntax" is an extension for dealing with the latter two aspects.

The talk will describe the bit syntax and the buffer mechanism and report from an experimental implementation and from experiences in using them for the implementation of actual protocols.

Plans are that the bit syntax (possibly in some revised form) will be included in a future release.

(This work was carried out at the Computer Science Laboratory.)

~ ~ Bus from Älvsjö to Dinner Event ~ ~

Conference chairman: Prof. Bengt Jonsson
Dept of Computer Systems
Uppsala University.



Demonstrations: Brainpool M/3 communication system, WAP and HACT (High Availability Telephony).

Conference Dinner: On a boat trip with m/s Waxholm III.



Organization Committee:

Bjarne Däcker, Computer Science Laboratory
Ericsson Utvecklings AB

Lilian Åhlberg, Open Systems Consulting
Ericsson Utvecklings AB

Anna Fedoriv, Open Systems Product Management
Ericsson Utvecklings AB

Torbjörn Keisu, Software Architecture Laboratory
Ericsson Utvecklings AB

How to get there: By air

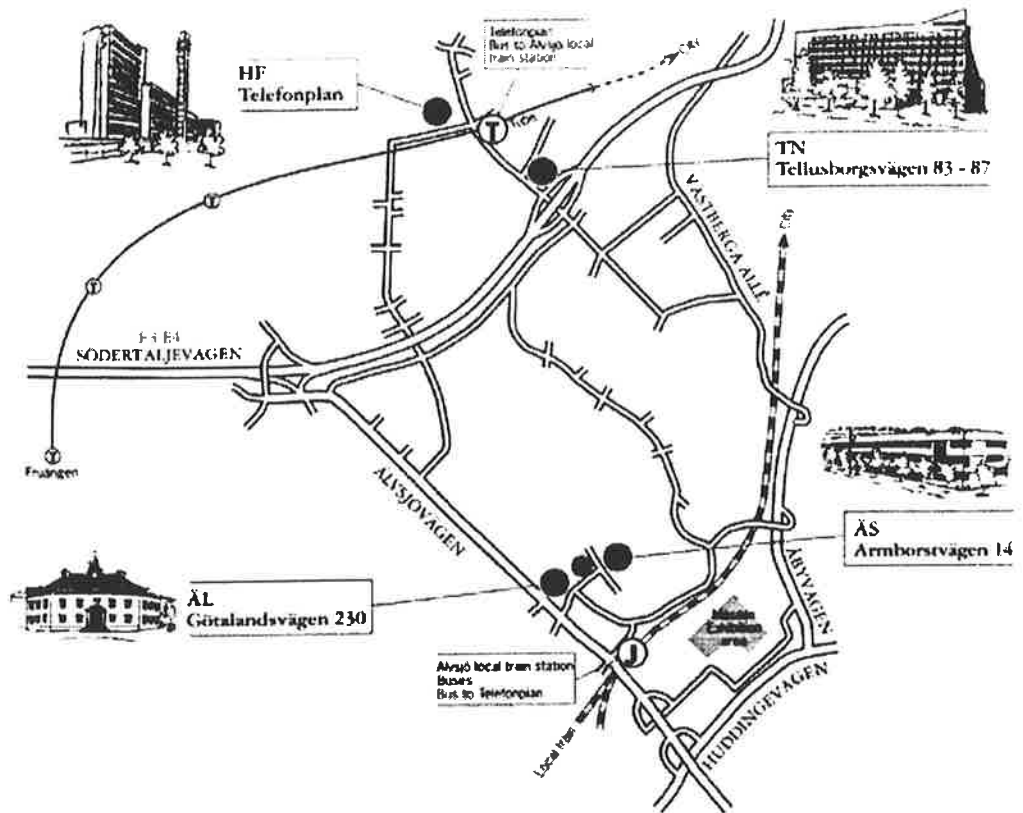
1. From Arlanda airport you take the bus (Flygbussarna) to Stockholm. This costs about SEK 60 and takes about 35 minutes. The bus stop (Cityterminalen) is almost on top of Stockholms Central Railway Station. Going by taxi is not much more convenient or faster, but much more expensive (about SEK 350). If you do go by taxi, be sure to agree on the price before starting the trip! It is quite likely that you otherwise will have to pay towards SEK 1000!
2. You then walk down to the railway station and take the local train, see below. Alternatively you could take a taxi but that would cost you almost SEK 200.

Via local train (pendeltåg)

1. Take the "pendeltåg" from Stockholms Central Station (the train station) south to Älvsjö. This takes about 7 minutes and costs about SEK 20. (If you intend to stay a few days, it is worthwhile to buy a strip, "rabattkuponger", instead.)
2. When you exit the station: turn right (not left to the Stockholm International Fair).
3. Walk down the stairs, turn right and cross Johan Skyttes väg and walk towards "Handelsbanken" and some other shops. The big road you have to cross to get to Handelsbanken is Götalandsvägen. However, don't cross the road: Go to the left and follow Götalandsvägen about 200 meters. You will bump into the main gate of the Ericsson buildings in Älvsjö. This is where you enter.

By car

1. Drive the E4 road south from the city and take off towards Älvsjö about 5 kilometers from Stockholm center.
2. At the first roundabout/circulation place on Älvsjövägen turn left. Park your car and enter the main gate.



Fax +46 8 719 89 40

EUC99
 Ericsson Utvecklings AB
 P.O. Box 1214
 SE-164 28 Kista
 SWEDEN



Erlang User Conference 1999 - Participants

Chairman and speakers			
Prof Bengt Jonsson	Uppsala University	Sweden	bengt@minsk.docs.uu.se
Per Bergqvist	Ericsson Radio Systems AB	Sweden	per.bergqvist@era.ericsson.se
Johan Blom	Ericsson Radio Systems AB	Sweden	johan.blom@ewi.ericsson.se
Maurice Castro	SERC	Australia	maurice@serc.rmit.edu.au
Magnus Karlson	UAB/Open Systems	Sweden	mk@erlang.ericsson.se
Håkan Millroth	Bluetail AB	Sweden	hakanm@bluetail.com
Hans Nahrungbauer	Telia Promotor AB	Sweden	hans.h.nahrungbauer@telia.se
Hans Nilsson	UAB/CSLab	Sweden	hans@erix.ericsson.se
Patrick Piché	Université de Montréal	Canada	piche@iro.umontreal.ca
Fredrik Ström	Brainpool AB	Sweden	fredrik.strom@brainpool.se
Claes Wikström	Bluetail AB	Sweden	klacke@bluetail.com
Participants			
Jukka Alapoikela	Ericsson Inc.	USA	jukka.alapoikela@ericsson.com
Jörgen Andersson	Ericsson Radio Systems	Sweden	jorgen.andersson@switchboard.ericsson.se
Kristoffer Andersson	Brainpool AB	Sweden	kristoffer.andersson@brainpool.se
Marcus Arendt	Marcus Arendt AB	Sweden	marcus@arendt.se
Joe Armstrong	Bluetail AB	Sweden	joe@bluetail.com
Thomas Arts	UAB/CSLab	Sweden	thomas@erix.ericsson.se
Mike Begley	Ericsson Systems Expertise	Ireland	michael.begley@etx.ericsson.se
Per Bengtsson	Telia Promotor AB	Sweden	per.x.bengtsson@telia.se
Johan Bevemyr	Bluetail AB	Sweden	jb@bluetail.com
Martin Björklund	Bluetail AB	Sweden	mbj@bluetail.com

Stefan Björnelund	UAB/Open Systems	Sweden	stefanb@erlang.ericsson.se
Lars Björnfot	UAB/SARC	Sweden	bjornfot@erix.ericsson.se
Hans Bolinder	UAB/Open Systems	Sweden	hasse@erix.ericsson.se
Kent Boortz	UAB/Open Systems	Sweden	kent@erlang.ericsson.se
Göran Båge	Ericsson Radio Systems AB	Sweden	goran.bage@era-t.ericsson.se
Richard Carlsson	Uppsala University	Sweden	richardc@csd.uu.se
Tuula Carlsson	UAB/Open Systems	Sweden	tuula@erix.ericsson.se
Francesco Cesarini	UAB/Open Systems	Sweden	cesarini@erlang.ericsson.se
Gennady Chugunov	SICS	Sweden	gena@sics.se
Marcus Claus	Brainpool AB	Sweden	marcus.claus@brainpool.se
Mats Cronqvist	Ericsson Telecom AB	Sweden	mats.cronqvist@etx.ericsson.se
Mary Daly-Scanlon	Ericsson Systems Expertise	Ireland	Mary.DalyScanlon@eei.ericsson.se
Bjarne Däcker	UAB/CSLab	Sweden	bjarne@erix.ericsson.se
Tommy Fagerberg	Enea Data AB	Sweden	tommy.fagerberg@enea.se
Anna Fedoriw	UAB/Open Systems	Sweden	anna@erlang.ericsson.se
Henrik Forsgren	UAB/Open Systems	Sweden	henrik@erlang.ericsson.se
Lars-Åke Fredlund	SICS	Sweden	fred@sics.se
Magnus Fröberg	Bluetail AB	Sweden	magnus@bluetail.com
Joakim Grebenö	Bluetail AB	Sweden	jocke@bluetail.com
Rickard Green	UAB/CSLab	Sweden	d95-rgr@d.kth.se
Dan Gudmundsson	UAB/Open Systems	Sweden	dgud@erix.ericsson.se
Dilian Gurov	SICS	Sweden	dilian@sics.se
Björn Gustavsson	UAB/Open Systems	Sweden	bjorn@erix.ericsson.se
Lars-Erik Gustavsson	Ericsson Business Networks	Sweden	lars-erik.gustavsson@ebc.ericsson.se
Radosman Gutierrez	Ericsson Telecom AB	Sweden	etxgura@kk.etx.ericsson.se
Thomas Verner Hansen	LM Ericsson A/S	Denmark	thomas.v.hansen@lmd.ericsson.se
Per Hedeland	UAB/CSLab	Sweden	per@erix.ericsson.se
Joakim Hirsch	UAB/Open Systems	Sweden	joke@erlang.ericsson.se
Bengt Holmström	Ericsson Telecom AB	Sweden	bengt.holmstrom@etx.ericsson.se
Gunilla Hugosson	UAB/Open Systems	Sweden	gunilla@erlang.ericsson.se
Anders Jacobsson	Ericsson Business Networks	Sweden	anders.jacobsson@ebc.ericsson.se
Erik Johansson	Uppsala University	Sweden	happi@csd.uu.se
Ylva Johansson	Sjöland & Thyselius	Sweden	ylva.johansson@st.se
Torbjörn K Johnson		Sweden	torbjorn.k.johnson@swipnet.se

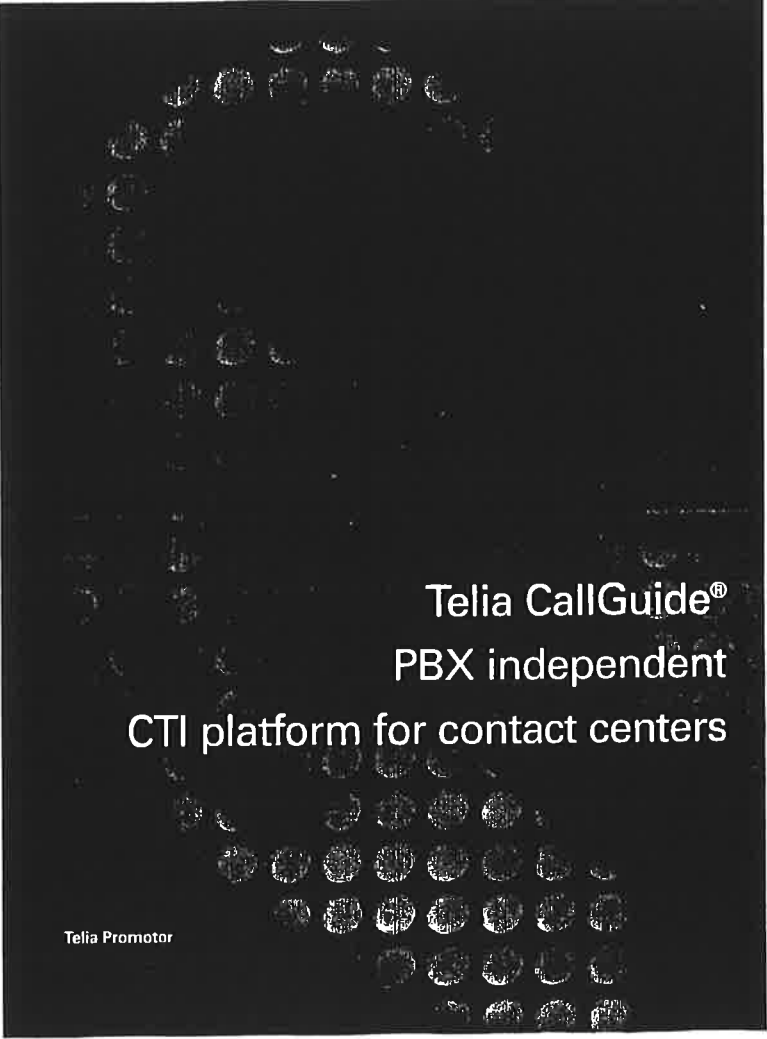
Henrik Jonasson	Ericsson Business Networks	Sweden	henrik.jonasson@ebc.ericsson.se
Per-Johan Josefsson	Frontec Tekniksystem AB	Sweden	per-johan.josefsson@sth.frontec.se
Micael Karlberg	UAB/Open Systems	Sweden	bmk@erix.ericsson.se
Bertil Karlsson	UAB/CSLab	Sweden	d95-bka@d.kth.se
Magnus Karlsson	UAB/Open Systems	Sweden	lagga@erlang.ericsson.se
Mikael Karlsson	Creado Systems	Sweden	mikael.karlsson@creado.com
Roland Karlsson	UAB/CSLab	Sweden	roland@erix.ericsson.se
Torbjörn Keisu	UAB/SARC	Sweden	keisu@erix.ericsson.se
Kjell Kristiansen	Kvatro Telecom AS	Norway	kjell.kristiansen@kvatro.no
Markus Kvisth	UAB/Open Systems	Sweden	markus@erlang.ericsson.se
Thomas Lange	Ericsson Radio AB	Sweden	thomas.lange@switchboard.ericsson.se
Magnus Lennartsson	Ericsson Telecom AB	Sweden	magnus.lennartsson@ericsson.com
Andreas Lindgren	Sjöland & Thyselius	Sweden	andreas.lindgren@st.se
Thomas Lindgren	Bluetail AB	Sweden	thomasl@bluetail.com
Stefan Lorenz	Ericsson Telecom AB	Sweden	etxstlz@kk.etx.ericsson.se
Peter Lund	Ericsson Radio Systems AB	Sweden	Peter.Lund@era.ericsson.se
Peter Lundell	Ericsson Telecom AB	Sweden	peter.lundell@ericsson.com
Kenneth Lundin	UAB/Open Systems	Sweden	kenneth@erix.ericsson.se
Matthias Läng	UAB/CSLab	Sweden	mml@erix.ericsson.se
Stefan Martinsson	Ericsson Telecom AB	Sweden	stefan.martinsson@etx.ericsson.se
Håkan Mattsson	UAB/CSLab	Sweden	hakan@erix.ericsson.se
Chandry Mullaparthi	Ericsson Systems Expertise	Ireland	eeicmui@eei.ericsson.se
Patrik Niklasson	Ericsson Business Networks	Sweden	patrik.niklasson@ebc.ericsson.se
Bengt Nilsson	UAB/Open Systems	Sweden	nibe@erix.ericsson.se
Raimo Niskanen	UAB/CSLab	Sweden	raimo@erix.ericsson.se
Sven-Olof Nyström	Uppsala University	Sweden	svenolof@csd.uu.se
Janine O'Keefe	Ericsson Telecom AB	Sweden	janine.okeefe@etx.ericsson.se
Arne Ohlsson	Ericsson Business Networks AB	Sweden	arne.ohlsson@ebc.ericsson.se
Simon Olofsson	Ericsson Telecom AB	Sweden	simon.olofsson@etx.ericsson.se
Mikael Pettersson	Uppsala University	Sweden	mikpe@csd.uu.se
Dan Sahlin	UAB/CSLab	Sweden	dan@erix.ericsson.se
Ola Samuelsson	Cyberode IT AB	Sweden	ola@cyberode.se
Denise Stack	Ericsson Systems Expertise	Ireland	eeidsk@eei.ericsson.se

Per Sternas	Ericsson Business Networks	Sweden	per.sternas@ebc.ericsson.se
Hishmat Sultani	Telia Promotor AB	Sweden	Hishmat.S.Sultani@telia.se
Henrik Swerin	UAB/Open Systems	Sweden	henriks@erlang.ericsson.se
Sebastian Strollo	UAB/Open Systems	Sweden	seb@erix.ericsson.se
Erik Strömbäck	Ericsson Radio AB	Sweden	erik.stromback@switchboard.ericsson.se
Göran Stupalo	UAB/Open Systems	Sweden	stupalo@erlang.ericsson.se
Ulf Svarte Bagge	Ericsson Radio AB	Sweden	ulf.svarte_bagge@switchboard.ericsson.se
Lars Thorsén	UAB/CSLab	Sweden	lars@erix.ericsson.se
Magnus Thoäng	Ericsson Telecom	Sweden	etxmagt@etxb.ericsson.se
Johan Tjäder	Ericsson Telecom	Sweden	etxjtj@etxb.ericsson.se
Robert Tjärnström	Ericsson Telecom AB	Sweden	etxtjr@al.etx.ericsson.se
Markus Torpvret	Connecta Teknik AB	Sweden	markus.torpvret@connecta.se
Christoffer Törnkvist	UAB/SARC	Sweden	crippa@erix.ericsson.se
Åke Uddén	Ericsson Telecom AB	Sweden	etxaal@kk.etx.ericsson.se
Robert Virding	Bluetail AB	Sweden	rv@bluetail.com
Jane Walerud	Bluetail AB	Sweden	jane@bluetail.com
Carl Wilhelm Welin	UAB/CSLab	Sweden	calle@erix.ericsson.se
Per Harald Westby	Kvatro Telecom AS	Norway	per.h.westby@kvatro.no
Sverker Wiberg	UAB/CSLab	Sweden	sverkerw@erix.ericsson.se
Ulf Wiger	Ericsson Telecom AB	Sweden	ulf.wiger@etx.ericsson.se
Jerker Wilander	Ericsson Radio AB	Sweden	jerker.wilander@era.ericsson.se
Mike Williams	UAB/Open Systems	Sweden	mike@erix.ericsson.se
Patrik Winroth	Bluetail AB	Sweden	patrik@bluetail.com
Lulseged Zerfu	Ericsson Telecom AB	Sweden	etxluze@kk.etx.ericsson.se
Lennart Öhman	Sjöland & Thyselius	Sweden	lennart.ohman@st.se

UAB = Ericsson Utvecklings AB

CSLab = Computer Science Laboratory

SARC = Software Architecture Laboratory



Telia CallGuide®
PBX independent
CTI platform for contact centers

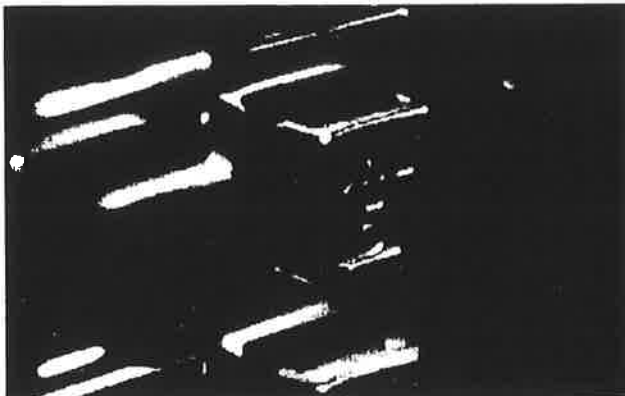
Telia Promotor

Customer care and service increase competitiveness

Top-quality customer care and customer service keep you ahead of the competition. And a contact center is the best way to manage customer care and service. How? By uniting computer-integrated telephony with other IT solutions.

Telia Promotor offers Telia CallGuide – a technical platform for contact centers. With this platform, your company can meet the most stringent requirements for accessibility and service, regardless of whether your customers contact you by phone, fax, or e-mail. The platform is also prepared for Internet telephony.

Telia CallGuide provides all necessary tools for effectively communicating with and following up on customers.



Independent of phone branch exchange

Telia CallGuide is a service solution with audio response (IVR) that can be connected to a phone branch exchange (PBX) with TAPI support. You can also mix different PBXs in the same customer-service.

You need no private PBX. Telia CallGuide can be connected directly to AXE or to an analog telephone. Customer-service reps are not tied to a permanent office; they can work from their homes.

Simple installation without an PBX link

Most CTI solutions require an PBX link for communication with the telephone PBX.

PBX links are expensive and generally quite complex to install.

With Telia CallGuide, you need not purchase costly PBX links nor update the exchanges. Installation is much easier and more secure.

Open interfaces and standard software


Telia CallGuide is based on open interfaces, which simplifies integration into customer databases, HR systems, and action-request (message) systems.

Telia CallGuide uses standard software, such as MS SQL Server, MS Windows NT, and CrystalReports. The applications use a Windows-based interface without scripting language.

Intelligent call control

Because calls are virtually queued using IVR, ACDs are not required. You can queue an unlimited number of calls and competence groups at your customer-service sites.

You can also provide each customer with a personal customer-service rep to whom the customer is always forwarded automatically.



Functions for managing queues and controlling calls

IVR – automatic services

- IVR with customer-specific menus and functions.
- Automatic services with connections to arbitrary support systems.
- Customers receive continuous updates concerning their place in the queue and estimated time of wait.
- Voice mailboxes.
- Fax.
- Analog or digital connection of IVR to the exchange.

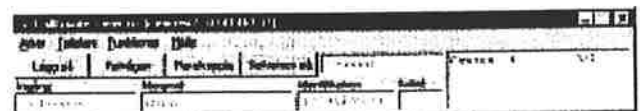
Competence-based call control

- Calls are forwarded to:
 - the customer-service rep with the right competence.
 - an available customer-service rep who recently helped the customer.
 - a personal customer-service rep.
 - external numbers when, for example, load is high or queues are long.
- During operations, customer-service reps can be assigned to an area of competence and to a group.

- Escalation/overflow of calls between queues.
- Different queue priorities.

Screen-based telephony – Telia CallGuide TeleList

- Telia CallGuide TeleList provides telephony functions in a Windows-based environment.
- Customer-service reps can monitor queue lengths and determine how many active customer-service reps fall within a certain competence area.
- Customer-service personnel can:
 - create a personal telephone book.
 - refer themselves via a referral function.
- Customer-service reps can put the customer on hold – to confer with a colleague. They can send questions to a competence area or group, which



TeliaList screen-based telephone.

means that they need not know the names of their colleagues; this is particularly advantageous when customer-service is distributed among several locations and serves many competence areas.

- Calls can be transferred to a customer-service rep, to a competence area, or to a group.
- Three-party conferences.
- Oscillation between calls.
- Pause-connection and post-service follow-up.
- Confidentiality.

Screen pop

- Information on the incoming call is retrieved from the support system and presented on the screen (screen pop) as the call arrives.
- The screen pop is shared during conference calls and accompanies calls that are transferred to a new customer-service rep.

Outgoing telephony – preview dialing

- With CallMeBack:
 - *IVR* – the customer can, via audio response, ask to be called back at a more convenient time.
 - *Web* – the customer can, via a Web page, ask to be called back immediately or at a more convenient time.
- Calling lists are:
 - used for CallMeBack and during campaigns.
 - sorted by priority, requested action, date, and so on.



The screenshot shows a window titled 'CallGuide Informant' with a menu bar containing 'Arkiv', 'Visa', 'Verktyg', and 'Hjälp'. Below the menu bar is a tab labeled 'CentralVäntelista'. The main area contains a table with the following data:

Ankomst	Identifikation	B-nummer	Ingång	Ärende
19990609 09:47:59	Saknas	020241024	Ordinarie	100

At the bottom of the window, there is a status bar with the text: 'Inloggad på 1 tjänst', 'Ansluten till CallGuide2: 211', and 'Summa kontakter: 1'.

Example of a campaign list.



- Campaigns – customer-service staff are reminded to survey their customers regarding action requests and company campaigns.
- Competence-based control also applies to outgoing telephony.

E-mail

- Competence-based control of incoming e-mail via Telia CallGuide Mail.
- Campaigns in the form of e-mail, which reminds customer-service reps to follow up customer requests.

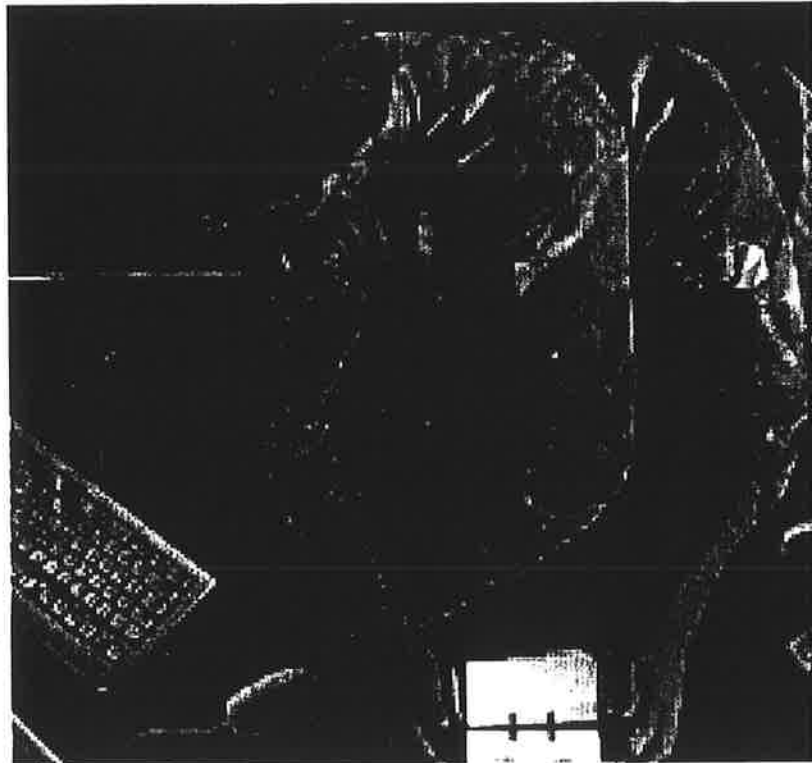
Real-time information, statistics and other functions

Administrative program

- Telia CallGuide Admin permits customer-service managers to administer queues, competencies, groups, and extensions via a simple Windows-based interface.
- Customer-service reps can be removed from and added to CallGuide Admin. Managers can easily create and change the profiles of customer-service staff.
- Personal customer-service reps or groups can be defined for certain customers.
- Business hours can be set in IVR.
- Messages can be sent to every customer-service rep.

Statistics

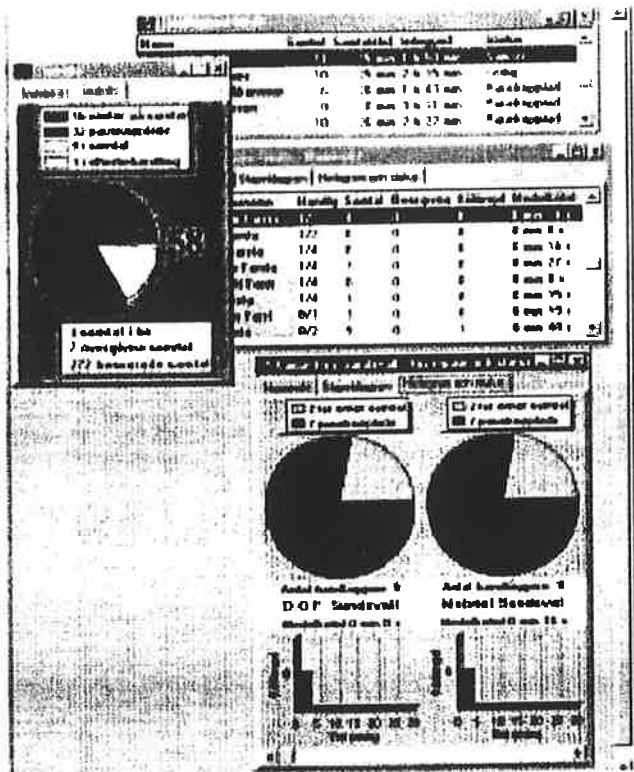
- Call-related. Number of incoming calls, number of dropped calls, and number of calls per requested action. Statistics are generated per half-hour, day, and week.
- Staff-related. Call duration, number of calls, total time per requested action, available status in percent, and percent of calls per customer-service rep.
- Outgoing telephony-related. Staff-related statistics and call-related statistics for campaigns.
- Seagate's CrystalReport, one of the world's most commonly used reporting tools, is included in the platform, along with several standard, customer-service reports.



- Telia CallGuide Stat generates statistics and enables managers to manipulate the information in standard business software, such as Excel and Word.

Real-time information on the contact center

- Current information is displayed in real time. Telia CallGuide Pulse reports on the number of queued calls, answered calls, abandoned calls, average queue time, number of reps logged into the system, representative status, and so on.
- The real-time information is presented numerically and graphically and may be displayed on screen.



CallGuide Pulse, real-time supervision.

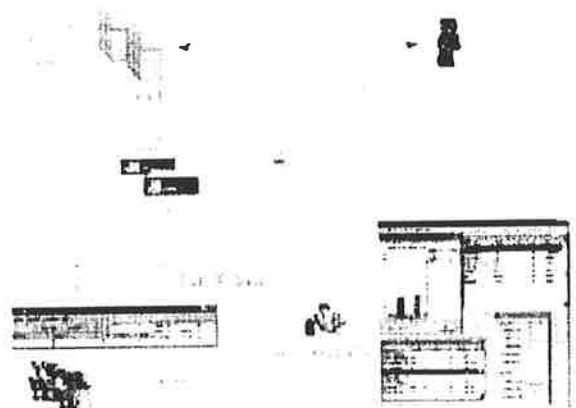
Other functions

- Customer-service reps are not required to work from a permanent site. The system registers the extension from which a representative is currently working (hot desking).
- Unlimited number of customer-service staff members or work sites.
- Distributed work sites; for example, home-based work sites can connect to the system via LAN, ISDN, or modem.
- The number of IVR lines can be expanded dynamically.

Operating system

- CallGuide Server runs under MS Windows NT 4.0 and uses MS SQL-Server as a database. See <http://www.microsoft.com>
- All client software is installed on Windows 95/98 or Windows NT 4.0.

System overview



Dependability

- Telia Promotor is the market-leading supplier of computer-integrated telephony in the Nordic region. In 1998, Telia Promotor delivered advanced CTI solutions for more than 2,500 customer-service reps.
- Dependable server technology with double RAID disks and Ericsson's Erlang/OTP technology. Ericsson uses Erlang code in its new-generation, public-network, ATM exchanges. Erlang offers many unique advantages, for example, the software code can be updated while in full operation.
- Tested audio response with nearly 1,000 installations in Sweden.
- System supervision with Telia CallGuide Alarm. Alarms are generated whenever any part of the system fails. Telia CallGuide can be based on IBM's Netfinity software.

Telia CallGuide® is a registered trademark of Telia AB. MS SQL Server and MS Windows NT are trademarks of Microsoft. Netfinity is a trademark of IBM. CrystalReports is a trademark of Seagate.

Acronyms and what they stand for

ACD: Automatic call distributor	OTP: Open telecom platform
ATM: Asynchronous transfer mode	PBX: Phone branch exchange
CTI: Computer telephony integration	RAID: Redundant array of independent disks
DDE: Dynamic data exchange	TAPI: Telephony API
IVR: Audio response	
LAN: Local area network	

Telia Promotor
www.promotor.telia.se



Goteborg	Haninge	Malmo	Solna	Uppsala
Lilla Bommen 1	Box 168	SE 205 21 Malmo	Box 2069	Box 1218
SE 411 04 Goteborg	SE 136 23 Haninge		SE 171 02 Solna	SE 751 42 Uppsala
+46 31 771 24 00	+46 8 707 35 00	+46 40 90 100	+46 8 764 35 00	+46 18 18 94 00



brainpool
consulting

Fredrik Ström M.Sc.

Vice President
Senior Consultant

brainpool
consulting

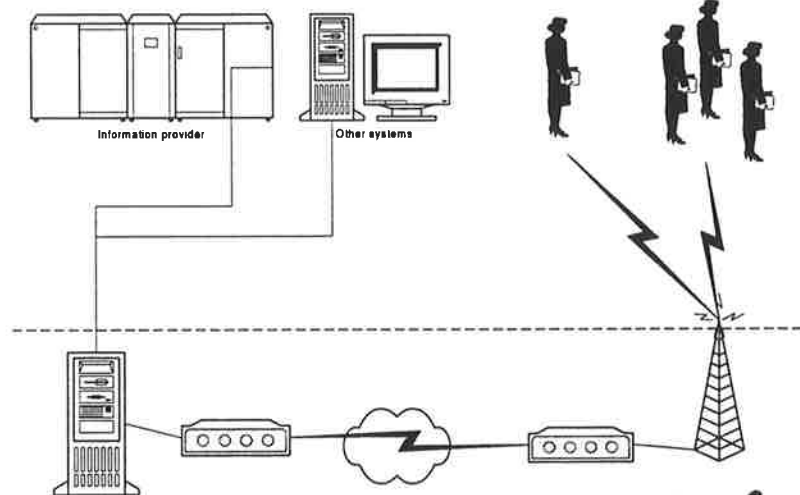
Brainpool at a glance



- IT Consulting company
- 23 consultants
- Business Areas:
 - Information Systems
 - Internet
 - Workflow
 - Mobility

brainpool
consulting

What is a MMS?



brainpool
consulting

Brainpool MMS



- 16 bit DOS
- Poor scalability
- Modem
- Country dependent
- One-way communication

brainpool
consulting

New Requirements



- Quicker transmission
- Two-way communication
- Country independent
- Scalable
- Robust

brainpool
consulting

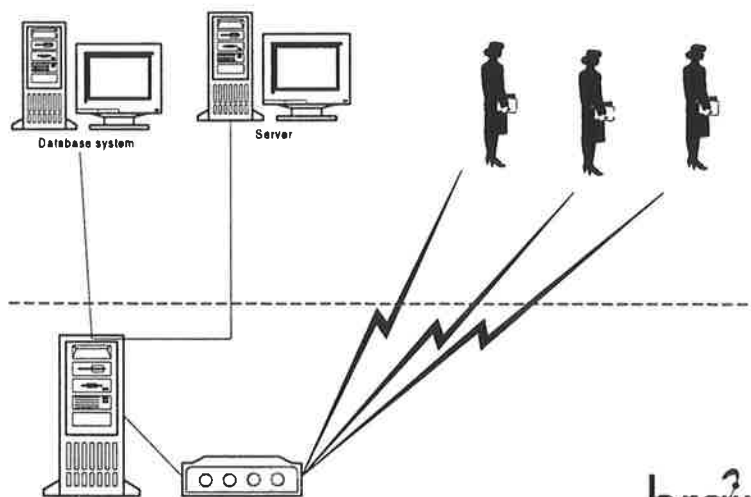
Brainpool M/3



- Erlang System/OTP
- Ericsson GM12
- ANSI C
- Windows service

brainpool
consulting

Brainpool M/3



brainpool
consulting

Erlang



- Rapid development
- Fewer bugs - more stable system
- Some functions missing
- Differs from “normal” development
- IG is powerful
- Mnesia is powerful



Conclusion



- We at Brainpool have found that the Erlang System/OTP is a reliable and usable platform.
- We look forward to working with Erlang in the future, as well as increasing our cooperation with Ericsson.
- We intend to train a larger group of consultants in Erlang to meet this goal.



More information



- Please contact
 - Fredrik Ström
 - 08 - 446 60 42
 - fredrik.strom@brainpool.se

 - www.brainpool.se

brainpool
consulting



brainpool
consulting

Företagspresentation

Författare

Fredrik Ström
Vice VD

Brainpool AB

BRAINPOOL AB – Företagspresentation

Affärsidé

Brainpools vision är att hjälpa företag och organisationer utveckla och förbättra sin verksamhet genom att använda informationsteknologi på rätt sätt. Företaget grundades 1991 och ägs av tre personer, samtliga arbetande konsulter i företagsledande befattning.

Vår affärsidé är att i nära samarbete med kunden analysera, utforma och implementera avancerade informationssystem och kommunikationstillämpningar i syfte att hjälpa kunden att arbeta och kommunicera effektivare, automatisera arbetsflödes- och beslutsprocesser samt presentera och distribuera information till interna och externa intressenter. Detta skall ske genom att utnyttja såväl kundens befintliga tekniska plattform och organisation som att utvärdera och rekommendera användning av nya produkter, verktyg och metoder inom IT.

Samtliga Brainpools medarbetare arbetar med såväl analys, design och modellering som med teknisk rådgivning, programmering, optimering och förvaltning av affärskritiska informations- och kommunikationssystem. Våra konsulter arbetar som utvecklare, systemarkitekter och projektledare och samtliga har någon av följande akademiska examina inom datateknik/datavetenskap: civilingenjör, fil. kand. och fil. mag.

För att bibehålla en teknisk bredd och kontinuerligt tillföra Brainpool kunskaper inom nya, relevanta områden samt att kommunicera ut vår profil mot en större marknad är vi partners med ledande tillverkare och aktörer på IT-marknaden. Brainpool AB är Oracle Certified Solution Partner, Microsoft Certified Solution Provider, certifierad JetForm Partner samt Lotus Business Partner.

Brainpool nyutvecklar, vidareutvecklar och integrerar både traditionella och webbaserade affärskritiska informationssystem för företag i olika branscher som till exempel distribution och handel, offentlig sektor, intresseorganisationer och myndigheter, medicin, telekommunikation, hotell och restaurang, förlag och media, industri och inte minst bank och finans. Bland våra kunder och samarbetspartners finns allt från mindre till mycket stora organisationer. Bland Brainpools kunder inom området för konsulting finns SBAB, SKTF, Liber, Scandic, Örebro Läns Landsting, ABB, Telia, Handelshögskolan, Bilprovningen, Linné Group och Ericsson. Bland företag som använder våra kommunikationsprodukter märks speciellt Market Makers Technology, SEB, Securitas, Telia MegaCom och ABB.

Personella resurser och kompetensområden

Samtliga Brainpools medarbetare arbetar med såväl analys, design och modellering som teknisk rådgivning, programmering, optimering och förvaltning av affärskritiska informations- och kommunikationssystem.

Våra konsulter arbetar som utvecklare, systemarkitekter och projektledare och samtliga har någon av följande akademiska examina inom datateknik/datavetenskap: civilingenjör, fil. kand. och fil. mag.

Som en naturlig del i våra konsulters, projektledares och kundansvarigas arbete ingår att hela tiden följa den tekniska utvecklingen, delta i och hålla internutbildningar och seminarier samt att gå på formella kurser. Vidare sker certifiering av våra medarbetarna inom ramen för olika certifieringsprogram hos de leverantörer vi samarbetar med; Microsoft, Oracle, IBM/Lotus m fl.

Teknik och partners

Beträffande verktyg och tekniska plattformar så arbetar vi med ledande produkter inom data/IT och har samarbete inom ramen för olika typer av partnerskap med bland annat följande aktörer på marknaden.

IBM

IBM satsning på lösningar för eBusiness sammanfaller med Brainpools fokus på områdena för Internet och elektronisk handel och kunnande inom relationsdatabasteknik och komponentorienterad systemutveckling. Partnerskapet är viktigt för oss eftersom flera av Brainpools kunder arbetar med blandade miljöer där IBM-plattformar utgör basen för olika typer av produktionssystem (ERP) och dessa behöver integreras effektivt med nya system för eBusiness. Vi arbetar speciellt med teknologier som DB2 och WebSphere/Java.

JetForm

Brainpool samarbetar med JetForm i Sverige och fungerar som kompetenskonsulter på områden som systemintegration och utveckling av blankettbaserade workflow lösningar och system för blankettdrivna centraliserade utskrifter i större system.

Lotus

Sedan 1994 arbetar vi med Lotus Notes/Domino, och är numera Lotus Business Partner med fokus på lösningar som baseras på teknologier som Lotus Notes och Domino.

Detta är ett naturligt steg för oss eftersom många företag och organisationer inom vår marknad arbetar med såväl relationsdatabasteknik som system för workflow och dokumenthantering och vi menar att det finns goda skäl att kombinera de olika teknikerna på ett bra sätt, inte minst i olika workflow-lösningar och i web-tjänster.

Microsoft

Brainpool är *Microsoft Certified Solution Provider (MCSP)*. Ett flertal av Brainpools medarbetare är certifierade på olika tekniker hos Microsoft.

Bland teknologier från Microsoft arbetar vi speciellt med de olika utvecklingsverktygen, komponentarkitekturen DCOM och BackOffice-plattformen, t ex med SQL Server, MS Transaction Server och Internet Information Server.

Oracle

Brainpool har av Oracle Svenska AB valts ut som en strategisk partner och är sedan 1998 en *Oracle Certified Solution Partner*. Vi har jobbat med Oracle teknologi sedan 1994 och varit Oracle Business Alliance Member och Oracle Value Service Provider sedan 1996.

Vi har i många olika projekt utvecklat affärskritiska informationssystem och webbtjänster baserat på Oracles databaser och Oracle Application Server. Exempel är beslutstöd- och produktionssystem i Intranätmiljö, shoppingtjänster på Internet, datalager och driftrutiner kring dessa.

Nya tekniker

Inom ramen för utveckling av vår tekniska strategi arbetar vi också med andra plattformar som vi bedömer är intressanta; Java på IBM minis/mainframes, DB2, Enterprise Java Beans och CORBA samt utveckling av distribuerade kommunikationssystem och feltoleranta applikationer baserade på språket och plattformen Erlang från Ericsson.

Speciellt intressanta är dessa tekniker för segmenten elektronisk handel, distribution, media, bank/finans och telekommunikation.

Inom området för workflow och blankethantering arbetar vi också med teknologi från JetForm, speciellt med koppling till mailsystem, databaser och webben.

Sammanfattning av den kompetens som Brainpool tillhandahåller

- Systemutveckling; projektledning, verksamhetsanalys, processmodellering, datamodellering, logisk systemdesign, prototypning/RAD, implementering, testmetodik, förvaltning/vidareutveckling, utbildning
- Modellering; datamodellering, ER/ERX, UML, flödesmodeller, processmodeller, tillståndsdiagram, objektorienterad analys och design
- Databaser; fysisk databasdesign, övervakning, prestandaanalys, optimering. Speciellt plattformarna Oracle, Sybase, MS SQL Server, DB2, Lotus Notes; distribuerade databaser, replikering, gränssnitt mot externa system och integration av heterogena databaser, datalager, N-skiktets client/server
- Programmering; Visual Basic, C/C++, Delphi, Object Pascal, Java, DCOM/ActiveX, CORBA, Oracle PL/SQL, SQL, MS Transact-SQL, Sybase Transact-SQL, Centura, Oracle Developer, MS ASP, MS Office, MS Access, Paradox, Lotus Notes, Erlang
- Internet/Web; HTML, Javascript, HTTP, Java, MS IIS, Oracle Web Application Server, IBM WebSphere., Apache, Microsoft Transaction Server, Lotus Notes/Domino, system för elektronisk handel, säkerhet; certifikat, kryptering
- Operativsystem; Windows alla versioner, OS/2, DOS, Netware, UNIX, Linux; system-API:er, drifrutiner
- Kommunikation; Seriell kommunikation, TCP/IP, integration av mailsystem med databaser, drifrutiner och applikationsprogramvaror, meddelandeprotokoll för mobila mottagare (SMS, personsökning).
- Egen produktfamilj Brainpool MMS och Brainpool M/3 för integration av mobiltelefoner, personsökare och datorsystem som sublicensieras och är i drift hos företag som t ex Securitas, ABB, Alfaskop, Telia och kunder till dessa.

Kvalitetssäkring

Organisation och policy

Styrelsen har slagit fast att Brainpool i alla avseende kontinuerligt skall förbättra och effektivisera arbetsmetoder för kvalitetssäkring; från dokumentation och ledning av administrativt arbete och säljprocesser till projektledning och tekniskt utvecklings- och förvaltningsarbete.

Vägledande i detta arbete är tankegångarna i CMM – Computer Maturity Model och dess fokus på repeterbarhet, mätbarhet, ledning och optimering av processer inom nyckelområden, *key process areas*. Detta kombineras med ett i grunden iterativt synsätt där optimering av arbetsmetoder introduceras och utvärderas stegvis i syfte att åstadkomma ett smidigt och icke-byrokratiskt arbetssätt.

Kvalitetsansvarig är VD, Marcus Claus.

Arbetsfördelning

Samtliga medarbetare arbetar efter en metodhandbok framtagen inom Brainpool AB. Den inkluderar rekommendationer av tillvägagångssätt, val av olika hjälpmedel och ger många praktiska exempel på rutiner för planering, ledning och uppföljning samt för tekniska tester och kvalitetskontroll.

Inom ramen för våra uppdrag är det respektive projektansvarigs ansvar att utbilda och skapa förståelse för kvalitetsorienterat arbete, samt att gentemot företagsledningen ansvara för att lämpliga metoder och konventioner läggs fast för uppdraget och följs.

Flera stödsystem är egenutvecklade och innehåller bland annat följande: Distribuerade dokumentdatabaser för lagring, kategorisering och åtkomst av affärsdokument, avtal med mera, avancerad projektdatabas med stödsystem för tidsstyrning och uppföljning, verktyg för systemdokumentation samt ett distribuerat system för hantering av defekter, önskemål, åtgärder och arbetsunderlag (*Configuration Management*). Information på filnivå hanteras med standardverktyg för versionskontroll.

Samtliga medarbetare har tillgång till de interna stödsystemen via säkra kanaler över Internet.

Kontinuerlig förbättring

Varje projekt avslutas med en uppföljning där projektgruppen och någon medlem i företagets ledning medverkar. Syftet är att säkerställa återkoppling och kommunikation, så att vunna erfarenheter kommer företagets övriga projektledare och konsulter till godo.

Vid utvärdering ses till sådant som faktiskt tidsåtgång i förhållande till planerad tid, precision i analys och dokumentations-/specifikationsarbetet, tätheten av defekter såväl på funktions- som på systemnivå, dokumentationskvalitet samt förvaltnings- och testrutiner.

Utbildning av nya medarbetare och annan personal sker i form av interna seminarier och föredrag.

Certifiering av medarbetare inom ramen för våra partnerskap med inflytelserika data/IT-företag, är vidare ett sätt att regelbundet kontrollera och garantera den höga kompetensnivån hos våra medarbetares inom viktiga tekniska områden.

Dokumentation av rutiner

Metodhandboken, mallar och exempel uppdateras kontinuerligt. All dokumentation och annan information relaterad till enskilda uppdrag lagras och nås enligt en standardmodell som alla Brainpools medarbetare har att följa.

Miljö

Styrelsen har slagit fast att Brainpool skall vara ett miljömedvetet företag. I detta ligger att beakta miljötekniska faktorer och alltid sträva efter det ur miljöhänseende bästa valet av leverantörer av dator- och kontorsutrustning samt förbrukningsmaterial.

Exempel är valet av modern datorutrustning med energispar-funktioner, hög grad av återvinnbara material samt undvikande av klorblekta material, t ex papper i kontorsmaterial.

Miljöansvarig är Administrativ Chef, tillika inköpsansvarig, Pontus Marcelius.

2000-Strategi

Styrelsen har kommit till slutsatsen att samtliga Brainpools egna system för administration, ekonomi, lön, affärsstöd, telefoni och data samt vår websajt, alla är 2000-säkra.

De system vi som konstruerar för kund är säkra avseende såväl design o konstruktion i alla avseenden.

I den händelse våra lösningar samverkar med system från tredje part eller bygger på program- och/eller maskinvara från annan leverantör bistår vi kunden med utredning, tester och anpassning i de fall dessa delar av systemen visar sig ej klara övergången till år 2000.

Status report on the ETOS Erlang to Scheme compiler

<http://www.iro.umontreal.ca/~etos>

<http://www.iro.umontreal.ca/~gambit>

<http://www.iro.umontreal.ca/~feeley>

Marc Feeley, Patrick Piché, Sylvain Beaulieu,
Martin Larose, Mario Latendresse

Université de Montréal



ETOS goals

- **Conform to Standard Erlang (final draft 0.6, june 1998)**
- **Generate fast code**
- **Reuse Gambit-C Scheme compiler technology**
 - Mature compiler
 - dynamic module loading
 - FFI (Foreign Function Interface)
 - Unicode support
 - Generates fast portable ANSI-C code (Unix, Windows, Mac)
 - Specially modified for ETOS

Summary

- Performance comparison
- Gambit-C optimisations
- Pattern matcher
- Future work

ETOS 2.3 compared to 1.4 and others

- ETOS 2.3 + Gambit-C 3.1f + gcc 2.8.1
- JAM/BEAM 47.4.1, Hipe 0.27

	Run time relative to ETOS			
	UltraSparc 143MHz		Pentium 400	
	Hipe	BEAM	Jam	BEAM
fib	1.65	3.27	7.24	3.27
huff	1.35	4.19	12.48	6.28
length	1.55	4.31	15.66	3.61
smith	2.20	3.56	7.55	7.38
tak	1.56	6.29	8.94	7.89
barnes	16.01	20.62	18.18	19.62
nrev	.89	1.47	5.13	2.15
qsort	1.11	5.21	11.10	7.07
ring	.57	.76	1.15	.70
stable	.91	.61	1.15	.54

- In general, v2.3 faster than v1.4 (better inlining & better intermodule calls by Gambit-C)
- Slower in a few cases (support for dynamic module loading causes more intermodule calls)
- Processes are now better (still not great due to remaining intermodule calls to kernel)

What is Standard Erlang?

- A "Cleaner" Erlang:
 - Character type and support for Unicode
 - New constructs: `all_true`, `some_true` and `try`
 - Order of evaluation = left to right
 - Recognizer BIFs begin with "`is_`" prefix
 - Function type

Using the ETOS compiler

```
emacs@baikal.IRO.UMontreal.CA
Buffers Files Tools Edit Search Complete In/Out Signals Help
bash$ etos -w -DETOS test.erl
"/u/feeley/etos/etos-2.3/test.erl"@4.5-4.10: Warning: unused variable Width
"/u/feeley/etos/etos-2.3/test.erl"@4.11-4.17: Warning: unused variable Height
bash$ estart test
[hello,100]
bash$

-:~* *shell* Sun Sep 26 17:47 0.02 (Shell:run)--L3--A11-----
-module(test).
-export([start/0]).

f(X,Width,Height) -> X*X.

start() -> io:write([hello,f(10,20,30)]), io:nl().

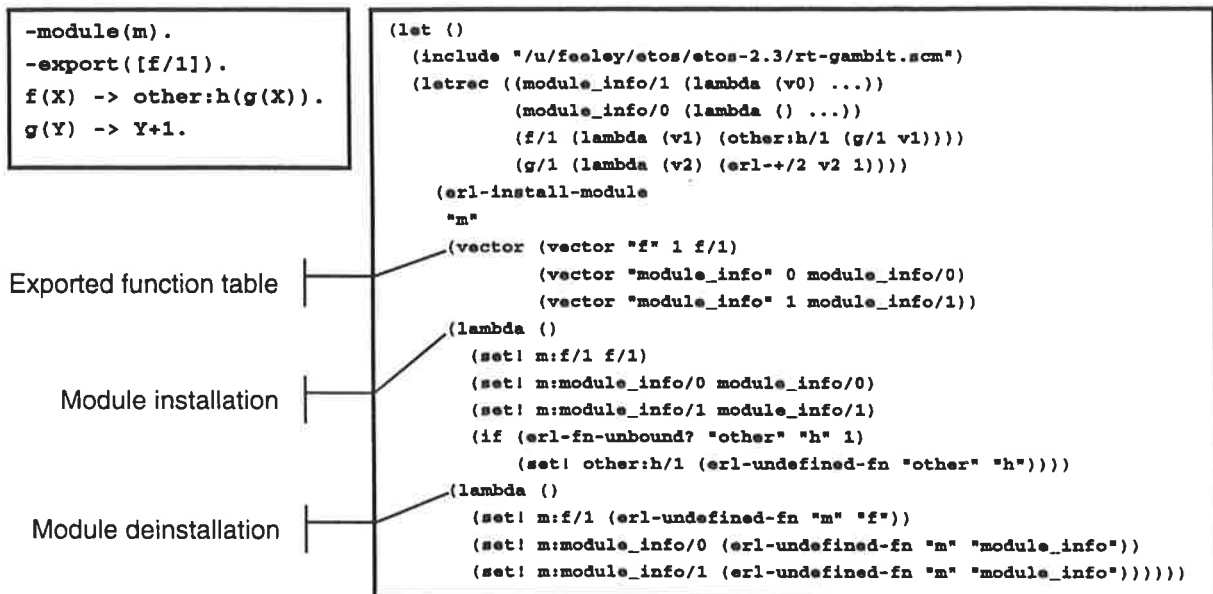
-:-- test.erl Sun Sep 26 17:47 0.02 (Text)--L4--A11-----
```

Compilation of an Erlang module (part 1)

Erlang source "m.erl":

```
-module(m).
-export([f/1]).
f(X) -> other:h(g(X)).
g(Y) -> Y+1.
```

Scheme code generated by ETOS front-end:



Compilation of an Erlang module (part 2)

Gambit-C expands this program to:

```
(let ((module_info/1 (lambda (v0) ...)
      (module_info/0 (lambda () ...)
        (f/1 (lambda (v1)
              (other:h/1
                (if (and (##fixnum? v1)
                        (##fixnum.< v1 (##fixnum.+ v1 1)))
                    (##fixnum.+ v1 1)
                    (erl-generic-+/2 v1 1)))))))
      (erl-install-module
       "m"
       (vector (vector "f" 1 f/1)
              (vector "module_info" 0 module_info/0)
              (vector "module_info" 1 module_info/1))
       (lambda ()
        (set! m:f/1 f/1)
        (set! m:module_info/0 module_info/0)
        (set! m:module_info/1 module_info/1)
        (if (##unbound? other:h/1)
            (set! other:h/1
              (lambda x (erl-undefined-fn-handler x 'other 'h))))))
       (lambda ()
         ...)))
```

- erl-+/2 inlined
- g/2 inlined and removed
- This is compiled to C code (not shown!)
- gcc compiles this to the shared library "m.o1"

Intermodule jumps in C

- Both Erlang and Scheme require constant space tail-calls
- Jumping from point "A" to "B" is a problem because A and B are possibly in different C functions (given that modules are compiled separately)
- Gambit-C uses a trampoline technique
 - Each control point is represented by a structure containing a pointer to the C function hosting that control point
 - A dispatcher function passes control from one host C function to the next

Simplified example

```
/*----- runtime library -----*/
ctl_pt *ret;          /* return address */
ctl_pt *erlang_add; /* initialized to &tbl_erlang[1] */
ctl_pt *m_f;         /* initialized to &tbl_m[0] */
ctl_pt *other_h;    /* initialized to &tbl_other[1] */

main () { ... dispatcher (m_f); } /* call m:f/1 */

void dispatcher (ctl_pt *pc) { while (1) pc = pc->host (pc); }

/*----- module "m" -----*/
ctl_pt tbl_m[] = { (host_m), (host_m) };

ctl_pt *host_m (ctl_pt *pc)
{ jump:
  switch (pc - tbl_m) {
    case 0: ... ret = &tbl_m[1]; pc = erlang_add; goto jump;
    case 1: ... pc = other_h; goto jump;
  }
  return pc;
}

/*----- module "erlang" -----*/
ctl_pt tbl_erlang[] = { (host_erlang), (host_erlang) };

ctl_pt *host_erlang (ctl_pt *pc)
{ jump:
  switch (pc - tbl_erlang) {
    case 0: ...
    case 1: ... pc = ret; goto jump;
  }
  return pc;
}
```

Example's details

- Remote computed jumps are slow (1 fn return, 1 fn call, 2 switch, 1 goto)
- Local computed jumps are faster (1 switch, 1 goto)
- Local direct jumps are fastest (1 goto)
- Virtual machine registers are cached in local variables for fast access (read on entry, write on exit), which makes remote jumps even slower

Specialized code for gcc

module "m"

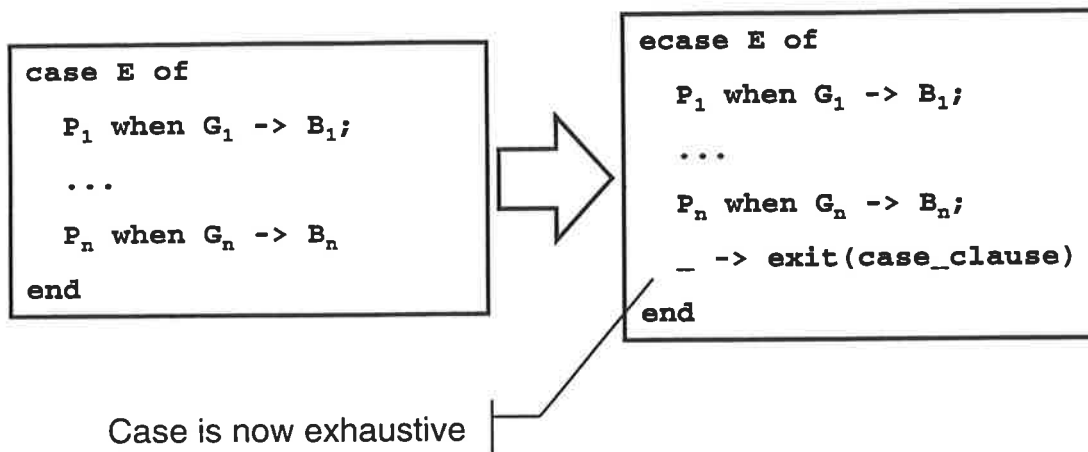
```
ctl_pt tbl_m[] =
{ {host_m,0}, {host_m,0} }; /* label field initialized later */

ctl_pt *host_m (ctl_pt *pc)
{ static void *lbls[] = { &label0, &label1 };
  if (pc == NULL) return lbls; /* to initialize label field */
  goto *pc->label;
  label0: ... ret = &tbl_m[1]; pc = erlang_add;
          if (pc->host != host_m) return pc;
          goto *pc->label;
  label1: ...
}
}
```

- gcc's "computed goto" can replace the switch statements
- Remote computed jump is 2 times faster
- Local computed jump is 3 times faster
- gcc's computed goto only works within one function (Mercury beware!)

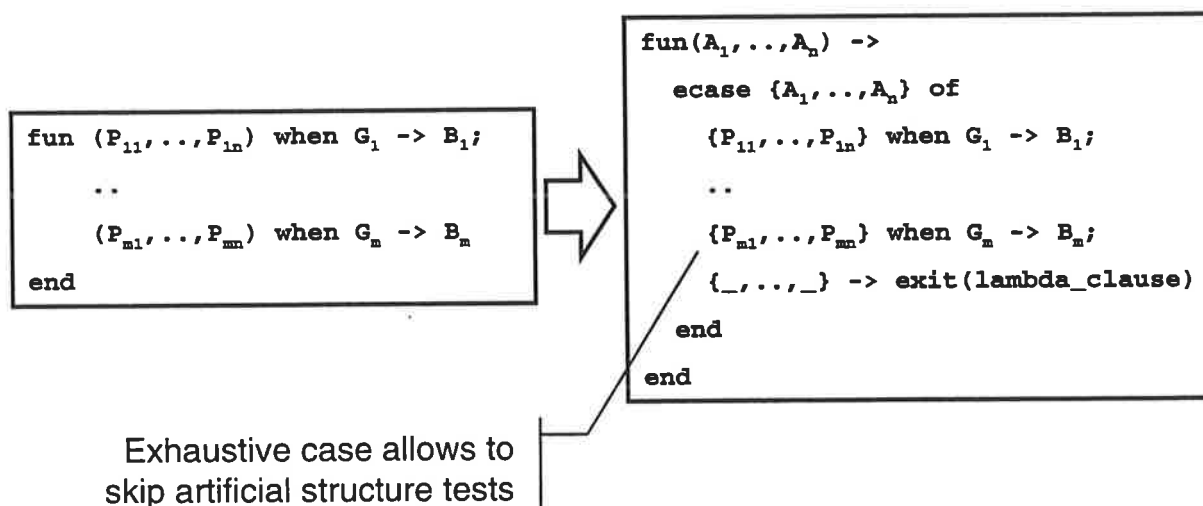
Pattern matching based on exhaustive case (ecase)

- A new pattern matching construct



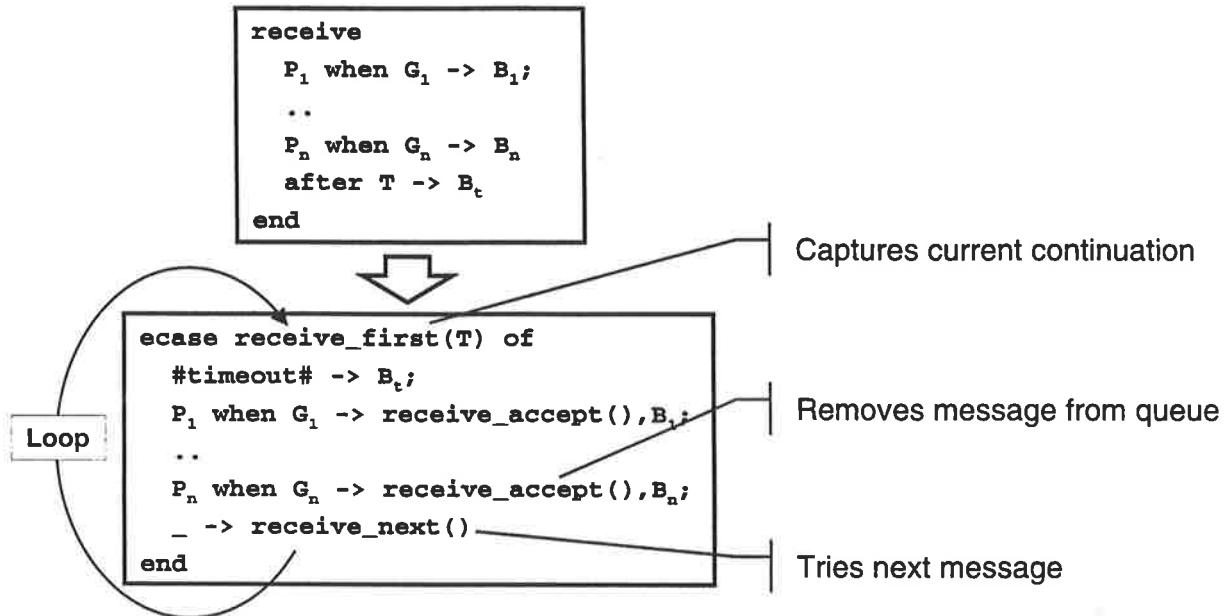
Function definition

- Function definition with ecase



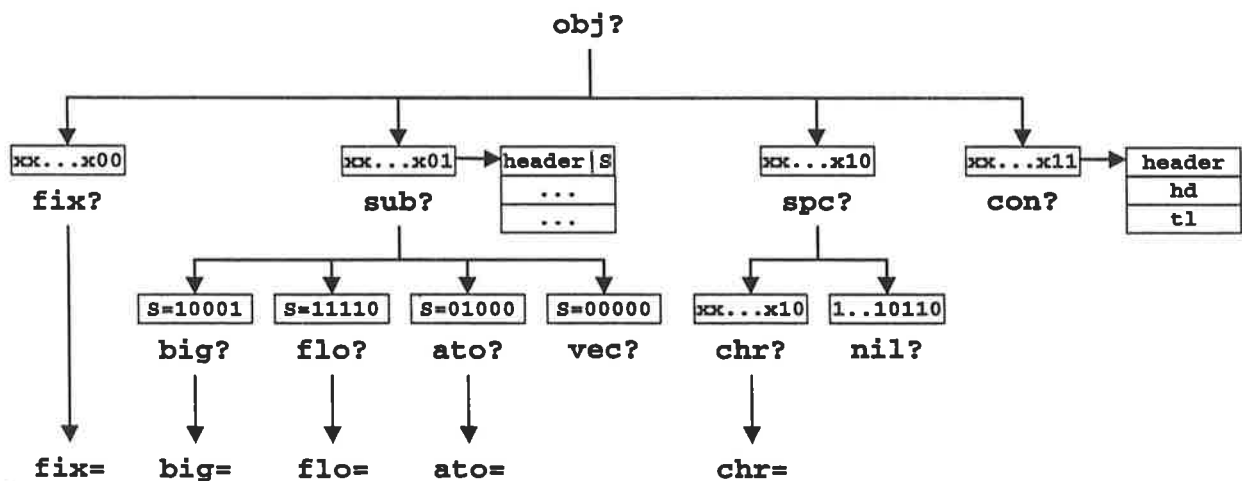
Implementation of receive

- call/cc mechanism allows simple transformation of receive into ecase



Type test expressions

- Type test tree based on Gambit-C data representation



Test operators

- Operators working on tests
 - Boolean
 - all, and, one, or, not, true, false
 - Predictive
 - kt(T) {T known to be true}
 - kf(T) {T known to be false}

Test prediction

- Relationships in data type hierarchy helps predict tests

```
PREDICT(TEST) =>
if(kt(<one child>)) TRUE
else
  if(!kt(<parent>))
    if(kf(<parent>)) FALSE else TEST
  else
    if(kt(<one brother>)) FALSE
    else
      if(kf(<one brother>)) TRUE else TEST
```

Example

```
ecase X of
  [$a$b] -> 1;
  [H|_] when is_char(H) -> 2;
  _ -> 3
end
```



```
(if (con? -x)
  (let ((h (erl-hd -x))
        (if (chr= h #\a)
              (if (chr= (erl-tl -x) #\b)
                    1
                    2)
              (if (chr? h) 2 3)))
    3)
```

chr? predicted to be true
(the child test chr= is known to be true)

Sestoft algorithm

- **Perform tests left to right, top down**
- **Accumulate positive and negative information**
- **Predict tests using this information**

Baudinet-MacQueen algorithm

- **Heuristic approach to choose test order**
- **Suggests three heuristics:**
 - relevance
 - First perform tests that discriminate first clause
 - branching factor
 - First perform tests that discriminate most clauses
 - Note: Modified for use with dynamic types
 - arity factor
 - First perform tests that span less new tests

ETOS pattern matcher

- Currently, Sestoft algorithm implemented and tested
- Implementation suited for easy upgrade to MacQueen
- Work currently in progress:
 - Development of better heuristics (example follows)
 - Use of hash tables for selection in vast atomic clauses

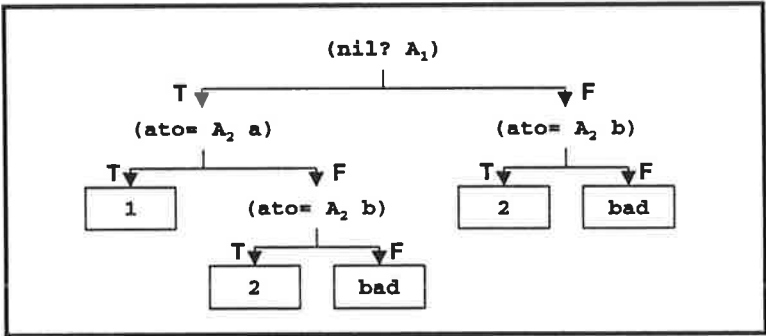
Pattern matching example

Erlang expression

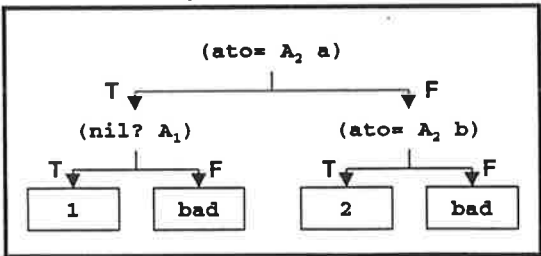
```

ecase {A1,A2} of
  {[],a} -> 1;
  {_,b} -> 2;
  {_,_} -> bad
end.
  
```

Sestoft and Baudinet-MacQueen test tree



Optimal test tree



- Sestoft: Left to right, top down
- BM: Unable to choose
- New heuristic: 2nd clause relevance

Future: real-time GC

- Experimented with various GC algorithms

- Stop & Copy

- Gambit-C's standard GC
 - 2 semispaces
 - "direct" access to objects



- Mark & Compact

- using non-movable handles



- Mark & Compact Real-Time

- handles allow fast relocation

- Brooks' Real-Time

- movable handle
 - 2 semispaces



- Brooks' Real-Time + Generational

- In design

Real-time GC overhead (Gambit-C 2.7)

	Alloc MB/s	Time secs	M&C	M&C R-T	Brooks R-T
boyer	3.14	16.46	.91	.87	1.67
compiler	.91	49.99	1.19	1.49	1.41
puzzle	.92	21.88	1.20	1.31	1.25
browse	3.95	33.36	1.25	1.70	1.48
conform	2.88	25.52	1.29	1.81	1.72
traverse	5.61	10.93	1.37	1.68	1.62
peval	5.86	35.40	1.52	1.81	1.68
fft	12.58	5.05	1.53	1.56	1.02
maze	16.18	11.58	1.65	2.43	2.34
simplex	15.34	10.91	1.70	2.18	1.76
earley	8.42	36.13	1.87	2.22	1.52
dderiv	22.76	39.39	2.04	3.00	1.83
destruc	19.35	15.29	2.11	2.67	1.77
cpstak	46.94	13.95	2.41	2.44	1.53
fibfp	47.05	11.80	2.43	2.95	1.29
deriv	27.43	32.69	2.50	3.11	1.82
mbrot	60.20	13.03	2.87	3.64	1.33
divrec	54.92	16.66	3.15	3.88	1.49
sumfp	71.12	85.82	3.21	4.28	1.38
diviter	123.03	7.44	6.51	8.12	1.69

- M&C = 2.14 * S&C (on average)
- M&C R-T = 1.24 * M&C (on average)
- Brooks R-T = 1.58 * S&C (on average)

Real-time GC pauses

•M&C R-T on a 133Mhz DEC Alpha 21064/ Digital UNIX v4.0 (\approx 100MHz Pentium)

	Avg(ms)	Max(ms)	% GC
compiler	3.69	11	4
puzzle	2.12	8	4
conform	2.86	5	5
peval	2.67	5	8
boyer	2.83	6	9
browae	2.86	5	10
traverse	3.57	12	13
simplex	3.01	6	20
destruc	2.85	6	21
earley	3.47	10	22
dderiv	2.88	6	24
fft	3.31	6	24
deriv	2.85	6	29
maze	3.12	15	35
cpstak	2.39	5	40
divrec	2.89	6	45
diviter	2.89	6	48
fibfp	3.31	6	52
mbrot	3.35	7	55
sumfp	3.45	7	57

•Avg = 2.1 to 3.7 ms / Max = 5 to 15 ms

Future: native code generation

- **Pro:** would allow faster intermodule calls
- **con:** low portability and need to redo C compiler's optimizations
- **Approach**
 - use intermediate "RTL" code (RISC style register machine)
 - expand RTL instructions to native code
 - use a generic instruction scheduler
- **Easy to port to a new machine (\approx 2 weeks) and reasonable performance**

Gambit-RTL performance

- Prototype implementation of Gambit-RTL compared to Gambit-C 3.0 (below 1 is good for Gambit-RTL)

	Alpha	MIPS	PPC	USPARC	PII	M68K
tak	.56	.69	.91	.70	.33	.26
fib	.75	.71	.85	.46	.53	.22
mazefun	1.07	.66	1.53	.36	.37	.35
assq	1.10	.73	.85	.60	.29	.98
nrev	.81	.65	3.18	.81	.35	.41
sort	1.67	.80	3.27	.65	.44	.70
takl	1.53	.83	1.22	.92	.48	.49



Mail Robustifier Whitepaper

H. Millroth

hakanm@bluetail.com

Overview

The Bluetail Mail Robustifier improves reliability, scalability and managability of standards-based third-party mail servers. That is, it makes your favorite mail server system more robust and manageable.

The mail robustifier is a software-only product. It works together with almost all conceivable mail architectures: separate machines for the different mail protocols, all mail protocols on each machine, etc.

Here are some examples of what it does:

- Makes it easy to add, remove or upgrade machines or individual mail servers without service outages.
- Masks server failures by automatically redirecting mail sessions to other servers.
- Dynamically load balances between mail servers in a better way than round-robin DNS.
- Smoothly handles overload situations by throttling connection attempts from new mail clients and by prioritizing clients with existing connections over new clients.
- Makes it easy to implement differentiated service classes (for example “gold class” users, who are given special privileges), by determining target server based on information in external databases.
- Supports spam filtering and admission control based on information in external databases.

Architectural overview

As can be seen from Figure 1, the basic architectural principle is to put a new layer between the mail clients and the mail servers. This layer manages the mail traffic to and from the servers.

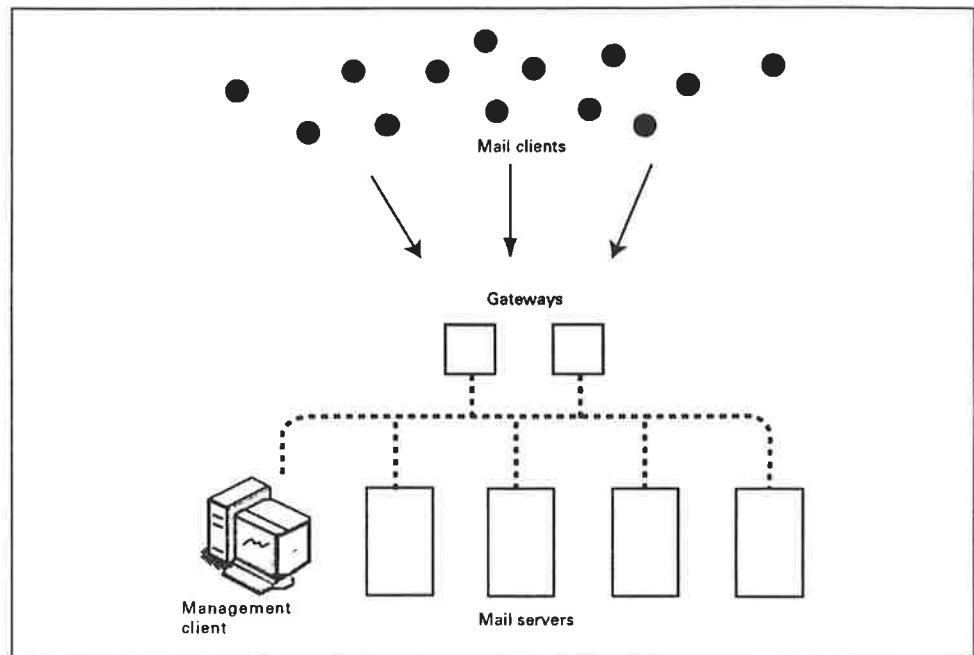


Figure 1: *Typical mail architecture using the Mail Robustifier.*

The robustifier connects the machines in clusters. If one machine in a cluster fails, then the other machines in the cluster cooperate to provide the functionality of the machine that has failed. Individual machines in a cluster may be taken out of service in a controlled manner without interrupting system operations. Additional machines can be added to a cluster without interrupting operations.

Some of the concepts discussed in this overview are described in greater detail in the *Configuring the System* and *Operating the System* chapters.

Gateways and mail servers

Incoming mail traffic is routed through *gateways* which perform traffic control and dispatch the traffic to the appropriate mail servers.

The system supports three types of gateways and mail servers: POP3, IMAP4, and SMTP. For each protocol type we can have one or more gateways and mail servers in the system.

DNS is set up to point to the gateways, not the mail servers. If there are several gateways for a protocol, use DNS round-robin to distribute requests.

The mail servers are not part of the Mail Robustifier product. Any mail server that conforms to the protocol standards works with the product. It is also possible to mix different servers for the same protocol in the system. For example, you can have one *Sendmail* machine and one *qmail* machine and load-balance between them. You can also migrate from, say, *Sendmail* to *qmail* (or vice versa) without service interruption, one machine at the time.

The management client and the management server

The robustifier is managed from a *management client* that runs on a standard PC or workstation. There are two management clients: one with a graphical user interface and the other with a command-line interface.

The management client communicates with the *management server* that handles system configuration and operational requests. If the server machine crashes, a new management server is started on another machine. The old server's IP address is re-mapped to the new machine and the management service is resumed, so this failover is transparent to the management client. That is, the client will have contact with the system as long as there is at least one machine up and running.

Multiple management clients can be connected simultaneously to the system.

Frontend and backend machines

The gateways and the management server run on *frontend* machines. The mail servers run on *backend* machines.

Nodes

The mail robustifier software runs on *nodes*. A node is a virtual machine that executes within a single operating system process. Each node can be configured to perform single or multiple tasks.

Each frontend and backend machine has one node (*frontend nodes* and *backend nodes*). The backend nodes measure the load on the backend machines and report this

information to the gateways. Thus there are two kinds of entities running on backend machines: backend nodes and mail servers.

All nodes in the system are connected and check each other's status through a low-level *heart beat* mechanism. This means that the system will quickly notice if any node stops working. In that case the machine is considered dead – see the section *Machine Failures* on page 6 for a description of which actions are taken in the event of failure.

Figure 2 shows two frontend machines, each running an SMTP and IMAP gateway, and three backend machines. The SMTP servers run on all three backend machines, while there is an IMAP server on only one backend machine. There is also one load-reporting node on each backend machine.

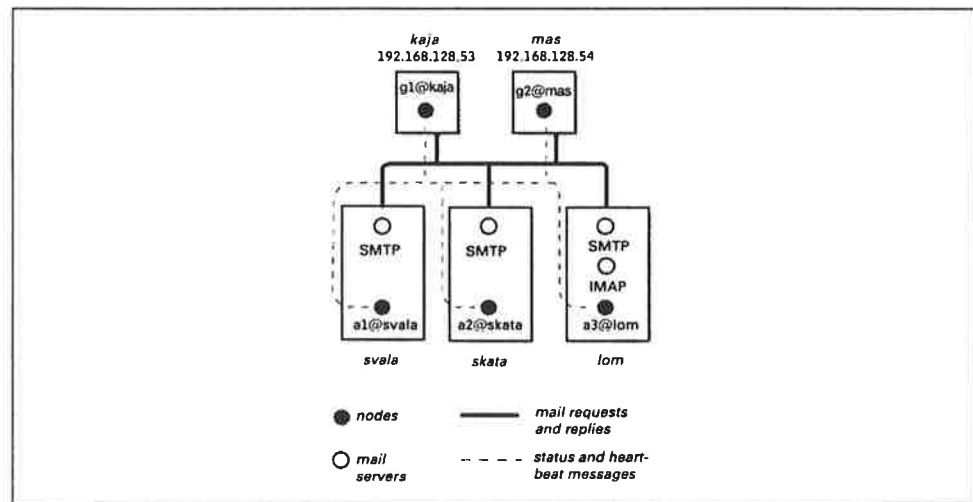


Figure 2: Configuration with two frontend machines and three backend machines.

In another scenario, a single node might run all three gateways as well as the management server. In a third scenario, gateways and the management server run on a backend node – this is useful mainly for failover and testing purposes (it follows from this that a machine can be both a frontend and a backend machine, although this is not normally the case).

Clusters

A *cluster* is a collection of nodes and mail servers that work together. A *frontend cluster* is a cluster running gateways and/or the management server. A *backend cluster* is a

cluster running mail servers and backend nodes.

The purpose of frontend clusters is to define failover sets: gateways fail over to other nodes in their cluster. Frontend clusters can be heterogenous: a particular cluster may run gateways of different types as well as the management server.

Backend clusters, in contrast, are used for purposes of load balancing and differentiated service classes. Backend clusters are homogeneous: a particular cluster can only run a single type of mail server.

Figure 3 shows a system with one frontend cluster and two backend clusters.

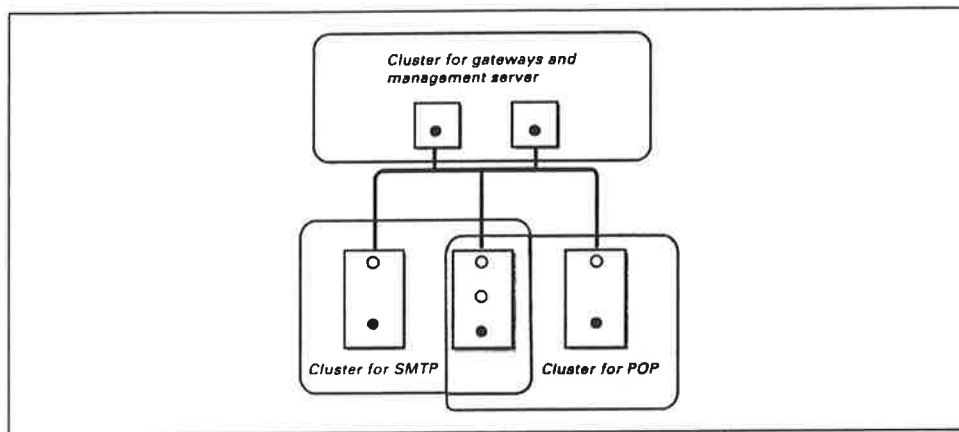


Figure 3: System with one frontend cluster and two backend clusters.

Monitors

Each machine running a gateway or mail server also runs a *monitor*. The monitor regularly “pings” the gateway or server using an appropriate protocol-specific request. This is used both to collect response times and to determine gateway/server failure.

Thus, the heart-beat mechanism check the status of *nodes* and monitors check the status of gateways and mail servers.

Load thresholds

Each node has a default *load threshold* that indicates the load at which the machine should be considered overloaded.

An alarm is generated if the average load of the nodes in a backend cluster reaches their

average load threshold. In addition, the overload control algorithm (see the *Overload control* section on page 9) is activated.

A single backend machine participating in several clusters has one load threshold per cluster.

For example, consider the two backend machines in Figure 4 which implement two clusters. Both the standard POP3 cluster and the gold customer POP3 cluster consist of nodes A and B. To prioritize gold customers, we set a higher load threshold in the gold cluster than in the standard cluster. This way gold customer get a larger share of the machine resources.

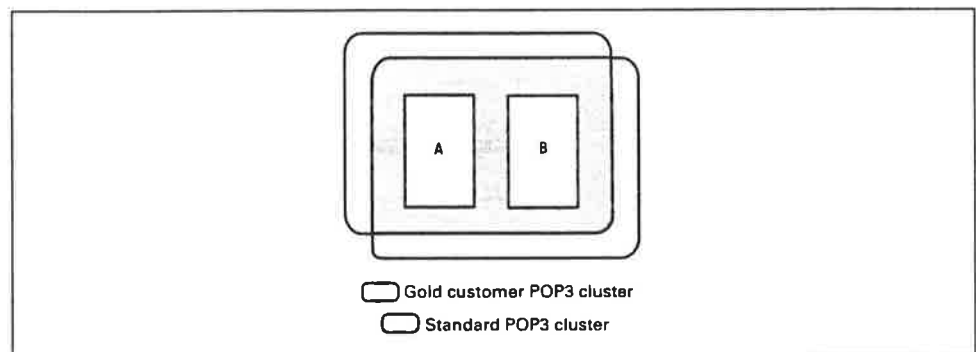


Figure 4: Configuration with two backend clusters

Machine failures

The mail robustifier can detect a machine crash in two ways:

- Node failures can be discovered by the heart-beat mechanism.
- A gateway or mail server may fail to answer its local monitor.

If a frontend machine is considered dead, the gateways and/or management server running on it fail over to other frontend nodes. The IP addresses of the gateways/management server are re-mapped to the new machines. When the failed machine comes back up, the gateways normally migrate back to it. Gateways can be configured to be *sticky*, that is they stay on the failover node when the crashed machine returns. The management server, for example, is sticky.

If a backend machine is considered dead, new requests from the gateways are redirected to another backend machine, until the first backend machine recovers.

Interfaces

Each gateway uses a default *interface* for network communication. If this interface is broken, or if the gateway fails over to another node, it uses one of the available interfaces on the node (these interfaces are called *failover interfaces*).

Data tables

Cluster selection and admission control / spam filtering can be based on information stored in external databases.

The information is preprocessed and then loaded into the system and stored on the local disks of every frontend machine. This allows for orders of magnitude faster searches than using an external database server on a separate machine.

POP3 and IMAP4 gateways each use a single table type for both admission control and cluster selection. SMTP gateways use four types of tables for spam filtering.

Features

Online system maintenance

Mail servers can be *blocked* — meaning that new requests to the server are directed to another server — and later *deblocked*. Using this feature, maintenance on backend machines can be done without disturbing normal operations. The procedure is:

- Block the mail server. Now the server no longer accepts new requests; these are directed to another server instead.
- Wait until most existing mail sessions on the server terminate (as determined, for example, by looking at the statistics monitors in the GUI).
- Stop the mail server. (Client sessions that still use the server will now be disconnected. If the clients reconnect, they will be connected to another server.)
- Do whatever maintenance that needs to be done: install a new mail server, upgrade to a new OS version, replace a broken interface card, replace the machine, etc.
- Start the mail server.

- Unblock the mail server. The server now accepts requests again.

To avoid the capacity loss of taking one machine out of service, a temporary backend machine can be added to the system before starting this procedure.

Software release handling

The mail robustifier is upgradable while in operation. To achieve this functionality it includes a sophisticated software release handling system for its own software.

Once installed, the mail robustifier takes care of managing different releases of its software: unpacking and installing new releases as well as rolling back to old releases. This is done by the system itself, since all frontend and backend machines must run the same release of the software to ensure correct behavior.

If a new release is installed while a machine is down, the release will be automatically installed when the machine returns. If the installation of a new release fails, the system automatically rolls back to the earlier release on all machines.

New releases of the mail robustifier software can normally be installed while the system is running, without disturbing normal operations. (However, major new releases may require that some or all nodes are stopped before the software is installed.)

Load balancing

The mail robustifier supports load balancing of backend machines on a per cluster basis. That is, the gateways dispatch requests to mail servers so that all backend machines within that cluster have the same load (more precisely, their load-to-threshold ratios are the same), so machines with different capacity handle different loads.

The load of machines belonging to different clusters is not balanced. (However, if the clusters overlap the load will automatically be balanced between these clusters. For example, the three machines in Figure 3 on page 5 will be load balanced, since the clusters have one machine in common.)

The load of a machine is measured by CPU load as reported by the Unix *uptime* command.

If more than one gateway is used, then each gateway is given a separate IP address and DNS round-robin is used to share the load between the gateways. A problem with DNS round-robin — stale IP addresses in DNS caches and clients — is avoided, since the IP address of a gateway is re-mapped if the gateway fails over to another machine.

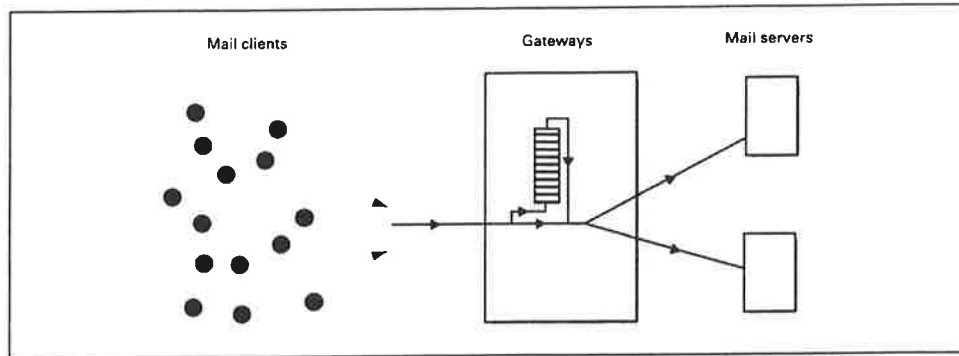


Figure 5: *Overload control.*

Overload control

In order to retain a high service level under heavy load, the robustifier can reject new client requests when the load exceeds the given threshold. Initially, only a small number of requests are rejected. If the overload situation continues, the rejection rate is increased until the overload situation is resolved. POP3 and IMAP4 clients that have been rejected can be queued in the gateway: when they are first in line, they are admitted even if the overload situation is not yet resolved (see Figure 5).

POP3 and IMAP4 clients that have been given access to a mail server are given a *prioritized time slot* during which they are assigned a “virtual session” during which their next command is prioritized over commands from new clients in overload situations.

prioritized in overload situations when they send new commands to the server. This is to ensure that a mail session made up of a sequence of individual commands is either accepted in its entirety or rejected before it even begins.

Rejected POP3 and IMAP4 clients are notified by a “connection rejected” notification, which is usually reported to the user in a popup dialog. Rejected SMTP clients or mail transfer agents get a “temporary error” return code indicating that they should try again later.

Admission control and spam filtering

New client requests can be rejected based on external table data. For example, we can specify that users matching a table entry should be blocked from our IMAP service.

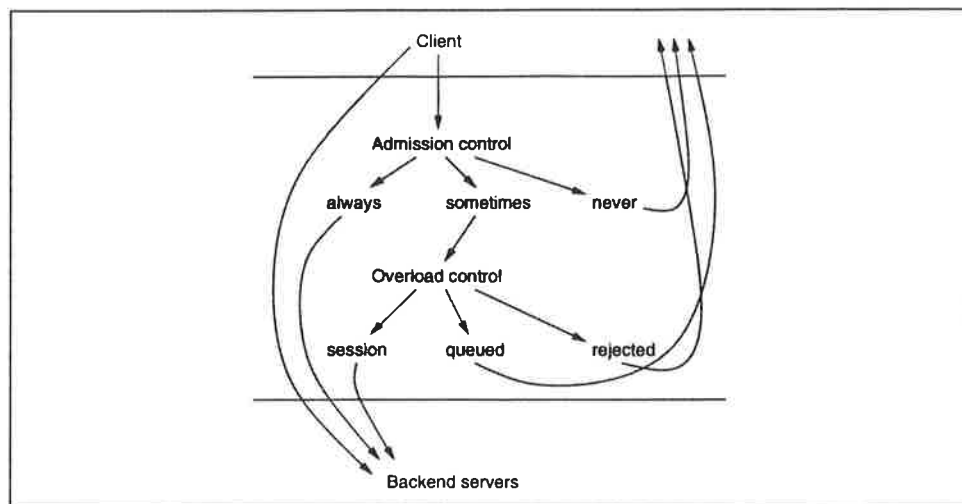


Figure 6: *Overload control.*

Spam filtering by the mail robustifier is done by SMTP admission control following the recommendations of RFC 2505. The mail robustifier supports the following table-based filtering methods:

- *Peer-name blocking.* The connecting host can be accepted or rejected based on its domain name or its IP address.
- *HELO blocking.* Mails from particular senders, as identified by the HELO command, can be accepted or rejected.
- *MAIL FROM blocking.* Mails from particular senders, as identified by the MAIL FROM command, can be accepted or rejected.
- *RCPT blocking.* Mails to particular recipients can be accepted or rejected.

The possible outcomes of overload control and admission control, and the relationships between them, are described in Figure 6.

Table-based cluster selection

POP3 and IMAP4 user requests can be dispatched to different servers based on external table data as shown in Figure 7.

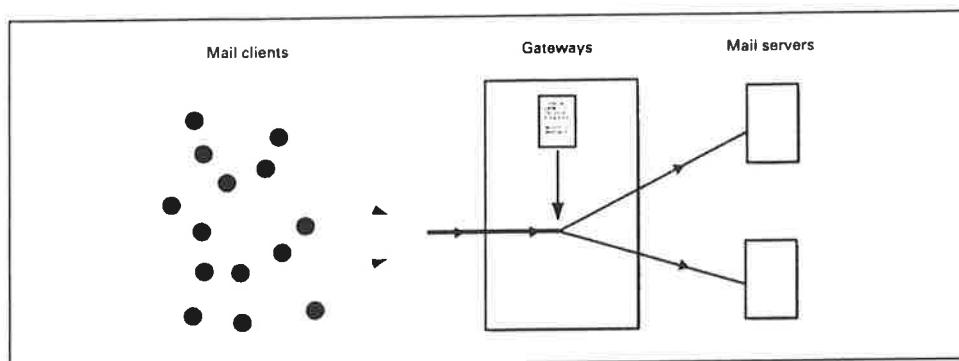


Figure 7: Table-based cluster selection.

For example, we can specify that POP3 users in *Table42* should be dispatched to the POP3 cluster *Cluster42*. An application of this feature is to dispatch premium customers to one backend cluster and regular customers to another.

Handling mail server failures

If a mail server crashes, the frontend gateway will automatically send new requests to other servers. If and when the server recovers, the gateway will automatically send it new requests.

Ongoing SMTP transactions are transparently moved to a new server if the current server crashes (the transaction state is stored in the gateway). That means that the message can be accepted although the current SMTP server crashes.

Handling gateway crashes

If a gateway crashes, it migrates to another frontend machine. As part of the migration, its IP address is re-mapped to the new machine. This means that the IP addresses handed out by DNS (and possibly cached by mail clients) will still work after a crash.

Existing connections are terminated if a gateway crashes; however, the clients can immediately reconnect and be served by the migrated gateway.

Example Configurations

This chapter describes four examples of mail system architectures that use the BLUETAIL Mail Robustifier.

Configuration files for the examples can be found in the *examples* directory of the program distribution.

A basic redundant system

The first example is a minimal architecture that exploits most benefits of the robustifier. It consists of one frontend machine running a combined POP3/IMAP4/SMTP gateway, and two backend machines running POP3, IMAP4 and SMTP servers (see Figure 8).

There is one frontend cluster consisting of all three machines, with the frontend machine as the primary node for the gateway. This means that the gateway will run on the frontend machine unless that machine is down; in that case the gateway will failover to one of the backend machines. Thus, in case the frontend machine crashes or is taken out of service, one of the backend machine will work as host for both the gateway and its mail servers.

The management server normally runs on the frontend machine. It will failover to a backend machine if the frontend is out of service, just as the gateway does.

There are three backend clusters, each consisting of both backend machines. This means that the two machines share the load for all three types of traffic (POP3, IMAP4 and SMTP). If one of the backend machines is out of service, the other machine handles all traffic. Possible overload due to reduced server capacity is handled by the overload control mechanism in the gateway.

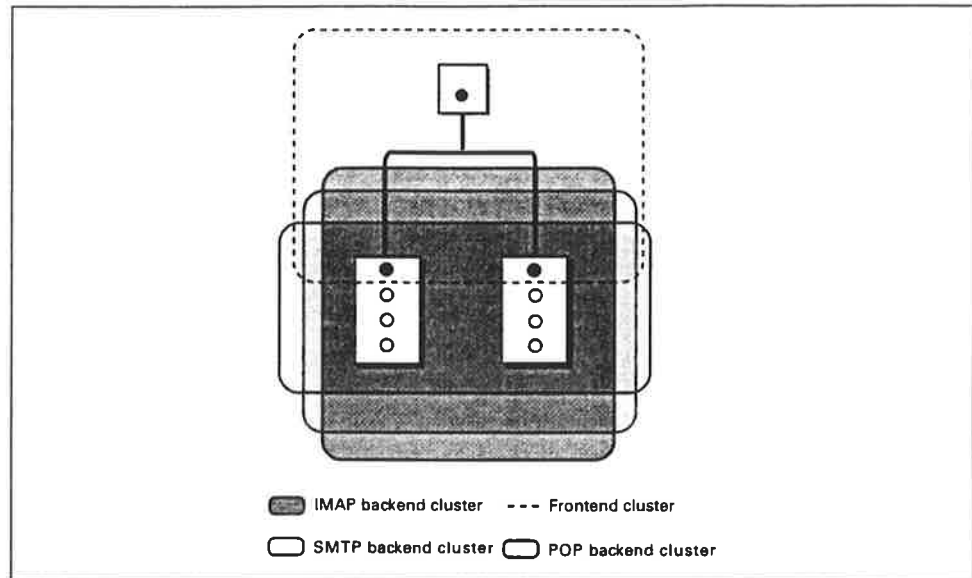


Figure 8: A basic redundant system (*basic.conf*).

This basic architecture can be varied in a number of ways, for example:

- More frontend machines can be added to increase gateway capacity.
- More backend machines can be added to increase mail server capacity.
- If there are several frontend machines, they can all run the same type of traffic, or each can be dedicated to one type of traffic (for example, one POP3 gateway and one SMTP gateway).
- Similarly, the backend machines can be partitioned in different ways: all machines can run all protocols or each machine may run only one or two of the protocols.
- For each type of traffic (POP3, for example), the backend machines can be partitioned to handle several service classes. For example, “free email” customers can be directed to a specific server while business customers can use all available servers and, in addition, have priority on the machine used for “free email.” This can be done without static configuration of client software; all clients use the same host name.

In the following examples we illustrate some of these options.

A large coarse-grained system

There are two basic ways to build a large-scale mail server system:

- A coarse-grained architecture with a few big server machines. Each machine runs several types of traffic.
- A fine-grained architecture with many small server machines. The servers are partitioned so that each machine only runs one type of traffic.

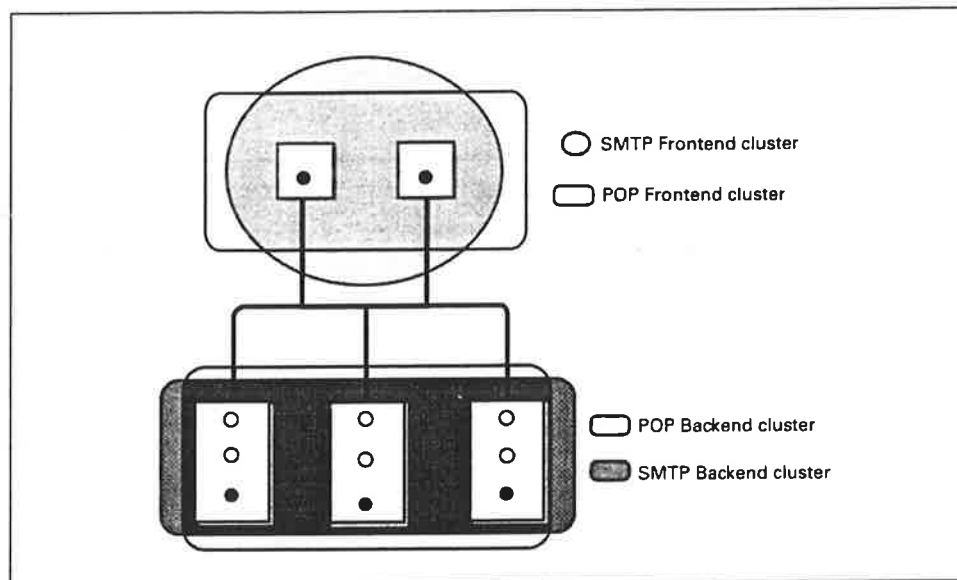


Figure 9: *A coarse-grained system (coarse.conf)*

Figure 9 shows a system of the former type. This system supports only POP3 and SMTP. It consists of two frontend machines and three big backend machines.

One frontend machine runs a POP3 gateway and the other runs an SMTP gateway. There are two frontend clusters, one for POP3 and one for SMTP. Each cluster consists of both machines – the POP3 gateway is also a standby for the SMTP gateway and vice versa.

Each of the three backend machines runs both POP3 and SMTP servers. The load thresholds for POP3 and SMTP can be differentiated, to prioritize one of the protocols over the other on each machine.

A large fine-grained system

Figure 10 shows how a large-scale mail system can be realized with a fine-grained architecture. The frontend cluster is the same as in the previous example.

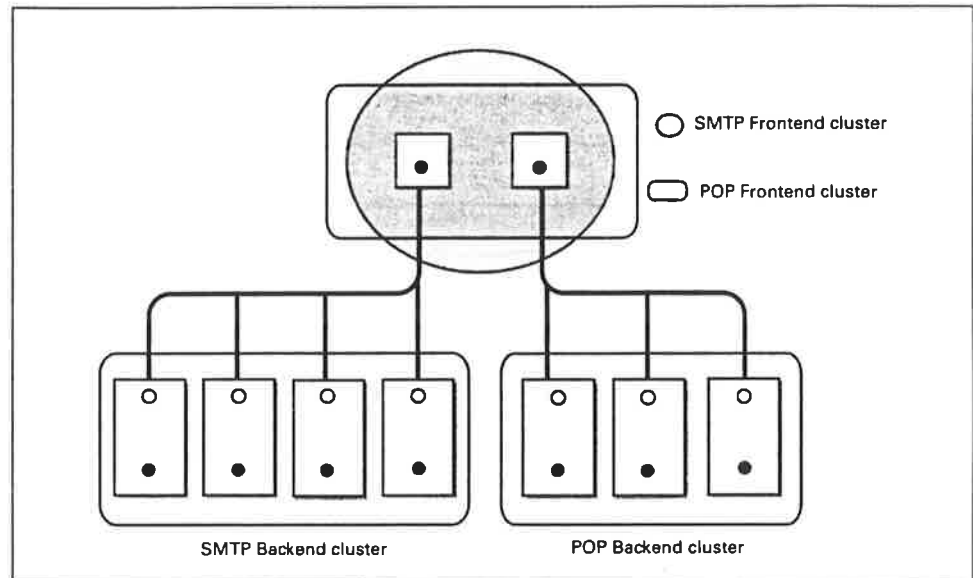


Figure 10: *A fine-grained system (fine.conf).*

The backend tier consists of four dedicated SMTP machines and three dedicated POP3 machines. The SMTP traffic is load-balanced within the SMTP cluster and the POP3 traffic within the POP3 cluster.

A fine-grained architecture like this has some advantages: there are small failure zones, and the system can grow incrementally by adding more small machines.

A system with differentiated service classes

The system shown in Figure 11 implements three different service classes for IMAP4 access, using three backend machines. Each service class is realized as a separate backend cluster:

- The “free email” cluster consists of one server.
- The “normal” cluster consists of the “free email” server and one additional server.

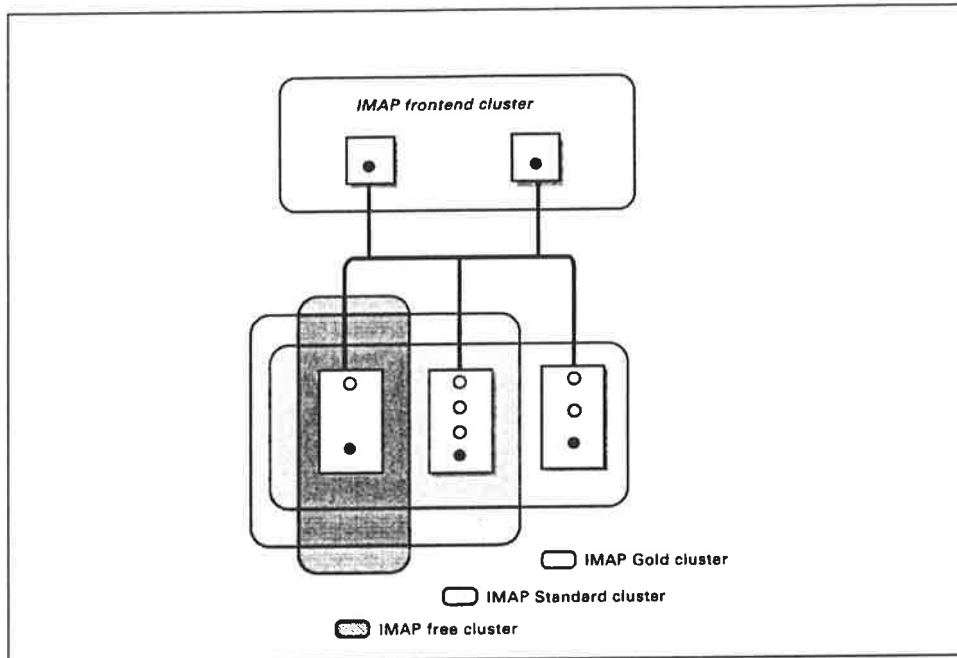
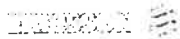


Figure 11: A system with differentiated service classes (*cos.conf*).

- The “business customer” cluster consists of all three servers.

This clustering scheme ensures that higher service classes have higher total capacity, and more reliability through redundancy. In addition, higher service classes can be further prioritized by having higher load thresholds than lower service classes on the shared machines (see the *Overview* chapter, page 6).





More Ericsson Topics [Go](#)

-Site Navigation- [Go](#)

[Info Center](#) - [Publications](#) - [Contact](#)

INFORMATION center

- Home
- News
- ▶ Publications
 - Connexion
 - ▶ Contact
 - Kontakten
 - Ericsson Review

Quick search

Advanced search

[Contact us](#)

[Index](#)

New ideas behind technology

[First published in Contact, 22 April 1999]

The fact that it was possible to develop a new exchange has had a lot to do with the way the work was conducted. It has been marked by new ways of thinking, questioning old routines and requirements, as well as the reuse of earlier research and knowledge.

The development of Simax, SwitchBoard and AXE110 has consisted of small-scale, entrepreneurial endeavors, bringing together individuals with various skills. The local Ericsson workforce has been supplemented with experts from subsidiaries as well as external resources. The fact that Per Bergqvist was able to borrow Simon Cornish from the company in Australia was decisive for the success of the project. Thanks to his key expertise, it was possible to assemble the building blocks for the AXE110 and

show off the first demo in only a few weeks.

"The work we did at Ellemtel at the beginning of the 1990s on the AXE research project was essential to the compact AXE110 exchange," says Staffan Skogby. Staffan subsequently met Karin Werhagen in 1996 who became excited about the idea and dared to invest in the technology. Out of that, the Simax simulation platform was created. The new method of implementing existing AXE10 software using the Simax emulator, makes it possible to take advantage of the millions of hours that were invested in the research and development of AXE over the years.

The process of implementing existing AXE10 software using a new processor and new group switch software has been patented.

SwitchBoard is the other key component in AXE110. SwitchBoard makes use of the latest technology in the way of programmable logic and digital signal processors, incorporating construction designs from military applications.

SwitchBoard, which is already used in the mobility server, replaces all essential AXE telephony hardware in the AXE110, including group switch, interface, digital multi-junctor, tone and recorded message generation/tone detection and so forth.

Read more at: <http://mega.al.etx.ericson.se/>

Written by Lars Cederquist
lars.cederquist@lme.ericsson.se
22 April 1999

Published 20 September 1999

contact online

Select a section:

- Products
- Organization
- Ericsson Worldwide
- Business
- Environment & Health
- Finance
- Human Resource Issues
- IS/IT
- Marketing
- Market Place
- Production & Supply
- Research & Development

Related topics

- ◆ Increased sales with the new AXE architecture
- ◆ Strong growth in Romania
- ◆ Internet soon free for Italians
- ◆ Olympian investments in telecom. AXE products
- ◆ AXE to be year-2000 compliant
- ◆ Fighting for their plant

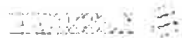
[Legal](#)

[Contacts](#)

[Finder](#)

[Home](#)

[Top](#)



More Ericsson Topics [Go](#)

-Site Navigation- [Go](#)

[Info Center](#) - [Publications](#) - [Contact](#)

INFORMATION center

- [Home](#)
- [News](#)
- ▶ [Publications](#)
 - [Connexion](#)
 - ▶ [Contact](#)
 - [Kontaktten](#)
 - [Ericsson Review](#)

Quick search

Advanced search

Contact us

Index

Smallest AXE in the world

[First published in Contact, 22 April 1999]

Working in Stockholm, two small groups of innovative developers have designed a portable, miniature version of the AXE10 exchange. This could be the first step for the millions of people who lack good telephone networks or who have never made a telephone call.

The demand for small, reasonably priced exchanges is enormous, particularly in Russia and former Eastern Bloc countries.

It all began just over a year ago, during the autumn of 1997, on two different fronts:

At what was then known as Public Networks, a group who were working on the Simax simulation platform thought it was time to turn their attention away from testing equipment to real exchanges. At the same time, Per Bergqvist, who had been researching mobile applications for several years, wanted to try something new.

"I wanted to do something that would really be valuable for Ericsson and came up with the idea of building a small exchange for both fixed and mobile telephony. I knew that other attempts had been made before and that it was considered to be impossible, somewhat akin to the fact that bumblebees really shouldn't be able to fly."

Very great demand

According to Svante Axling, who has worked on developing business in Russia, operators there were practically screaming for a small exchange which could be put into operation locally at a reasonable cost. Staffan Skogby, Johan Olsson and others formed an innovation cell based on a local exchange at Public Networks, while Per Bergqvist, Christer Palmgren and Simon Cornish started a "skunk works" focusing on the NMT exchange. These two groups maintained ongoing, but informal, cooperation with each other.

After examining Ericsson's technology inventory, the Simax emulator - at that time a testing tool with very good performance - was chosen as the main component, along with SwitchBoard, which had earlier been successfully used in a mobility server.

The two systems were housed together, forming what could be perceived as "real" AXE hardware. SwitchBoard was also the most cost-effective alternative according to an external comparison. "We combined various components into a whole and improved the real time properties so that we could test real traffic," says Per. "Things moved quickly. In March 1998, we already had a working prototype for NMT."

contact online

Select a section:

- [Products](#)
- [Organization](#)
- [Ericsson Worldwide](#)
- [Business](#)
- [Environment & Health](#)
- [Finance](#)
- [Human Resource Issues](#)
- [IS/IT](#)
- [Marketing](#)
- [Market Place](#)
- [Production & Supply](#)
- [Research & Development](#)

Related topics

- ◆ Increased sales with the new AXE architecture
- ◆ Strong growth in Romania
- ◆ Internet soon free for Italians
- ◆ Olympian investments in telecom. AXE products
- ◆ AXE to be year-2000 compliant
- ◆ Fighting for their plant

The present system fits into a small box and is really a variation of the cabinet used to house the Business Phone small business exchange. The exchange, which has the working name AXE110, weighs approximately 20 kilos and can easily be carried onboard airplanes. It really only consists of two components and is very simple to put into operation. Although it is a copy of the real AXE10 exchange, it is not as robust and fault tolerant with dual processors and so forth.

"But that is of no significance for the environments it is designed for," says Holger

Ronquist, Ericsson's market manager for Central and Eastern Europe. "Within our geographic area there are approximately 450 million inhabitants and only every fifth person has a telephone. In addition, about 60 percent of the existing lines are old and are in need of replacing."

In other words, the needs are urgent, particularly for local exchanges since almost all traffic consists of local calls. Moreover, half of this gigantic area is sparsely populated.

Low startup costs

The advantage of this compact exchange is that it offers an operator in, say a village of 1 000 inhabitants, the chance of starting up traffic at a low initial cost.

The alternative - starting up directly with a large exchange - is, in reality, an impossibility for most companies. The start-up costs for a country such as Kazakhstan, with 16 million inhabitants, would amount to a couple of billion U.S. dollars. The interest alone on such a loan would amount to almost USD 500 million per year.

"Since there is so little hardware in our mini-exchange, we're able to tear down that initial hurdle and rent out equipment, providing us with ongoing income for the software," explains Holger Ronquist. "In just a few years, based on a calculated cost of USD 10 per month per subscriber, we can receive as much money as a large exchange would cost."

In Holger Ronquist's opinion, Ericsson has always been good at building up a large customer base, but less successful at making money from those customers.

"This is the way in which we should charge our customers in the future."

Svante Axling emphasizes that the new exchange should be viewed as a start-up package for the market. Once an operator has generated some revenue and experienced market growth, they can upgrade their system to an AXE10 exchange.

"The fact that our little exchange uses all the same software and has the same functions as the AXE10, means that our old AXE customers will recognize the system and be able to use the equipment immediately," explains Svante.

"That is an advantage over our competitors, who are also trying to develop compact exchanges, but are six months or a year behind us. They lack a similar customer base. And nobody has yet succeeded in developing a good mini-exchange for both fixed and

mobile telephony."

Written by Lars Cederquist
lars.cederquist@lme.ericsson.se
22 April 1999

Published 20 September 1999

[Legal](#)

[Contacts](#)

[Finder](#)

[Home](#)

[Top](#)

WAP REFERENCE STACK AND GATEWAY

1. INTRODUCTION

This project is about a WAP reference stack, where selected parts of the WAP standard are completely implemented to conform to WAP 1.1. Both client (typically part of a mobile phone) and server (typically part of a WAP gateway) versions of the stack are implemented with the UDP/IP bearer.

The WAP stacks are complemented with a management module that adds additional ability to start multiple stacks, send message events, logging and check status on ongoing traffic. Additionally each stack component might be tested by separate optional test modules with the possibility to destroy and corrupt messages.

An IDL/CORBA interface have been implemented on the server side.

The development environment used is Erlang/OTP.

2. BUILDING A WAP STACK

The flexible architecture of a WAP stack implies several possible configurations. Thus an implementation of a WAP reference stack aims for a modular approach and a strategy on which order the stack should be built and tested. This implementation proposal aim to support such a strategy by keeping each possible WAP layer in separate Erlang modules, where each component on a layer might be easily exchangeable with another component on the same layer. In addition, test modules needs to be easily removable. This can be achieved by storing the API to a layer in specific *glue* modules such that surrounding layers have a common interface. The only functionality of such a glue module is to map requests from one component to another and thus all needed is to replace such a module whenever a stack component is replaced or a test module added or removed.

Thus we have the system components as in Figure 1. Note that we regard the security layer (WTLS) and the control message protocol (WCMP) are not yet implemented. The management facility is limited to a number of configurable parameters.

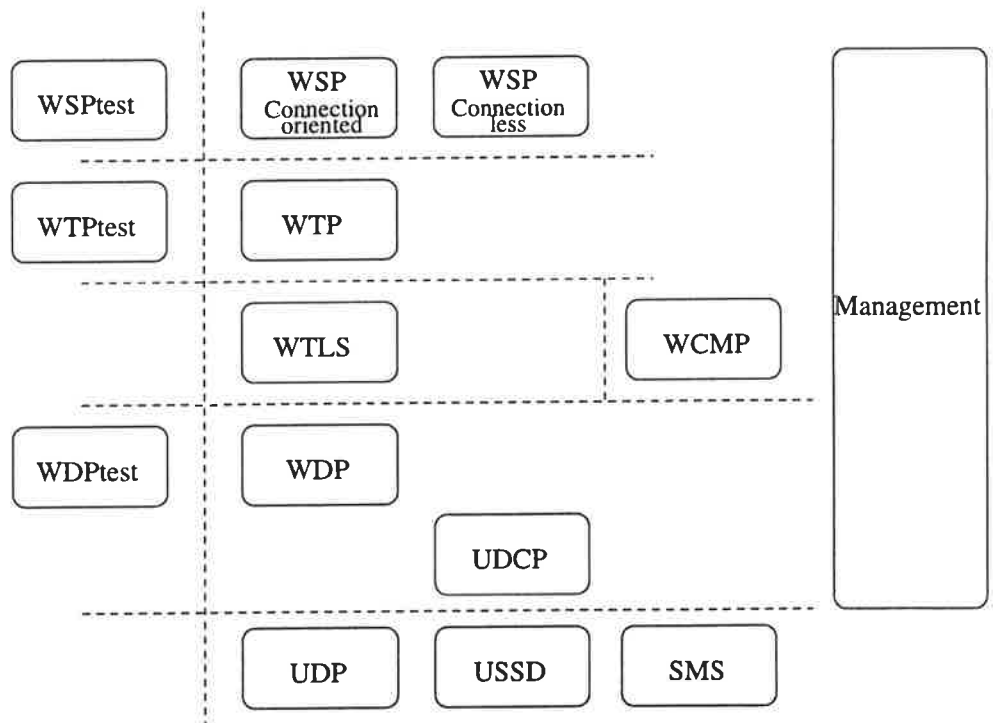


Figure 1 Overview of system components.

Each component has a set of configurable parameters and an associated test module with the ability to corrupt and destroy messages, trigger timers etc

The WAP stack consists of a number of static and dynamic processes. Typically a component consist of a single static process responsible for management of dynamic processes in the component. The latter, dynamic, processes are shadowed in the figures below. Concurrent requests, particularly important in the WAP gateway, are handled by creating new processes for each request.

Apart from these processes there is also a need for additional support in case of a system failure. More precisely; a supervisor process to restart static processes and a safe database that stores vital information.

3. WIRELESS SESSION PROTOCOL (WSP)

3.1. REFERENCE LAYER

The session layer protocol offers services most suited for browsing applications, similar to HTTP/1.1, but with additional functionality for long-lived sessions, capability negotiation etc.

Both client and server versions of the WSP layer are similar in structure and descriptions will therefore apply to both. The necessary functionality can be structured in blocks, corresponding to processes, as depicted in Figure 2 and Figure 3. The session layer may have several concurrent sessions each making several concurrent requests.

3.1.1. WSP Connection-oriented service

The WSP version that requires a *connection* to be set up before user data is transferred between two parties and confirms all requests. The facilities included involve the handling of sessions that can be suspended and resumed.

For each process, except *wsp_manager*, outlined in Figure 2 there is a corresponding state table defined in the WAP WSP standard. The *wsp_manager* process only responsibility is to keep track of existing *wsp_session* processes or start new ones, if necessary, and forward messages. Thus all communication with surrounding layers will go through *wsp_manager*. A session might live for a long time, but the session process might time out much earlier. Thus the *wsp_manager* might also need to restart a *wsp_session* process for an already active session.

Each concurrent session has a dedicated *wsp_session* process that is responsible for setting up, suspend and resume the session. All session specific data such as negotiated capabilities, session id etc. is stored in a separate database. Each new method invocation is handled by a separate *wsp_method* process and each new push by a separate *wsp_push* process. The related *wsp_session* process creates both.

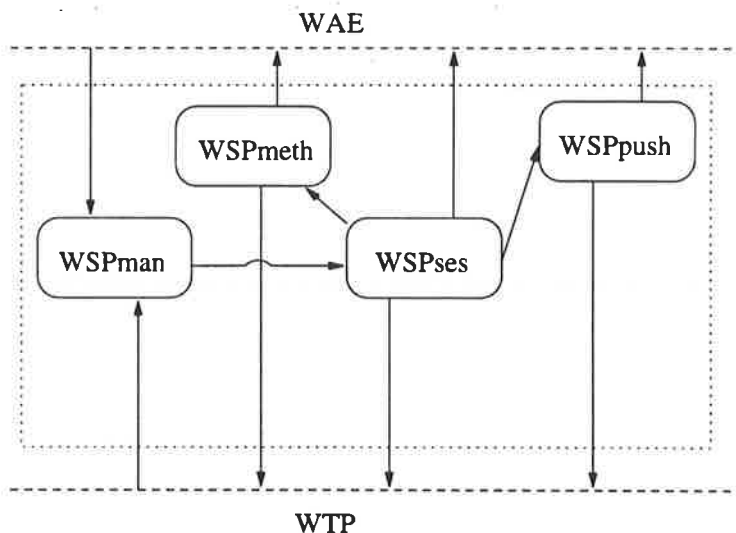


Figure 2 Erlang processes in the WSP layer with Connection mode.

3.1.2. WSP Connection-less service

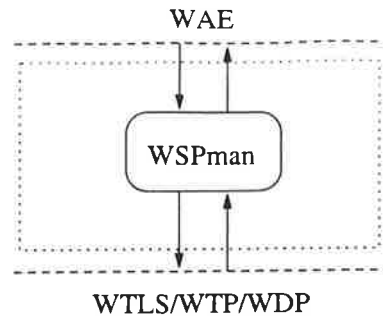


Figure 3 Erlang processes in the Connection-less WSP layer.

In the second version of the WSP layer all facilities are non-confirmed, thus communication may be unreliable.

The processes in Figure 2 contains the protocols for handling confirms etc for requests etc. In the connectionless version these protocols are not needed and therefore all functionality might be encapsulated in a single *wsp_manager* process, as illustrated in Figure 3.

The connection-less version operates directly above the WDP transaction layer protocol and thus utilises a different API to the underlying bearer as compared with the Connection mode version.

4. WIRELESS TRANSACTION PROTOCOL (WTP)

4.1. REFERENCE LAYER

The WTP layer improves reliability to a datagram service and efficiency to a connection oriented service.

There are three classes of WTP transactions, each needed on both client and server side, in a complete implementation of a WTP layer.

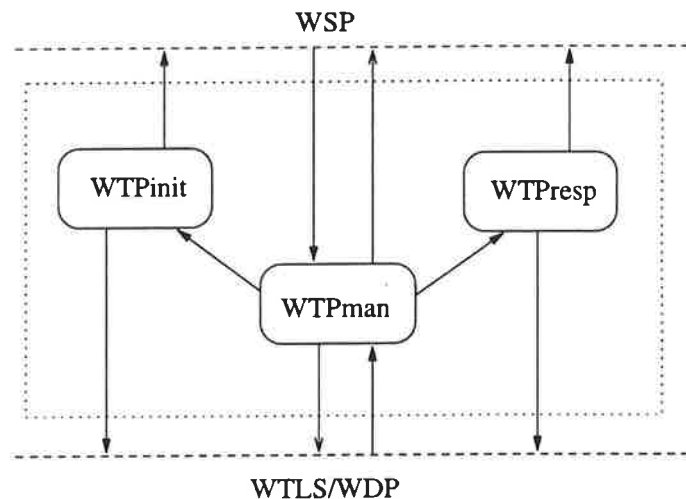


Figure 4 Erlang processes in the WTP layer with Class 0-2 transactions.

The initiator and responder state machines defined in the WAP-WTP standard are implemented in the *wtp_init* and *wtp_resp* process respectively. Whenever a class 1 or 2 transaction is detected such a process is started and all forthcoming messages associated to the transaction are forwarded from *wtp_manager*. The *wtp_manager* process is then responsible for detecting which transaction class is used for a message. During a class 1 or 2 transaction it also needs to decide if the message from the bearer should be forwarded to the initiator or responder. In the case of a class 0 transaction the *wtp_manager* process takes care of all functionality

A summary of the current status for the transaction classes.

- Class 0 transaction - Provides unreliable invoke messages with no result messages. Current status: Not implemented.
- Class 1 transaction - Provides reliable invoke messages with no result messages. Current status: Not implemented.

- Class 2 transaction - Provides reliable invoke messages with result messages. Current status: Some functionality missing. Mainly timers and multiple PDU:s in a single message.

5. WIRELESS DATAGRAM PROTOCOL (WDP)

As the UDP/IP doesn't require any additional adaptation layer the implementation of UDP/IP in WDP is a straightforward issue.

6. GATEWAY/PROXY APPLICATION

The application above the WAP server stack is the bridge between WSP and HTTP.

In addition the following features are implemented

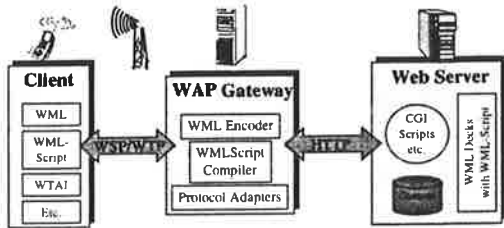
- Encoding of WML and WTA content
- Character set conversion, including Big5, Shift JIS, KSC5601 (Korean) and Unicode
- Push interface (not completely compliant with suggested standard)

7. CONCLUSION AND CHARACTER OF PROPOSED SOLUTION

The following list of issues gives an indication of pros and cons of the solution.

- The design is chosen to give a very close relationship between the state tables as defined in the standard from WAP forum and the implementation.
- The design takes a lot use of processes as it simplifies the design. Message passing between Erlang lightweight processes is regarded as very cheap, still message passing might be a bottleneck.
- WSP sessions might live for a long time (several weeks), thus to save memory WSPses processes might timeout, if the session is idle, after a certain much shorter time. Whenever the session is used again the WSPses process then needs to be restarted and forced into the state it was when it timed out. This implies that the current state as well as other data specific for the session needs to be stored in a specific database. Unexpected failures (as a system crash) might then also be handled without the need of a reestablishment of the session, method and push invocations, however, will be lost.

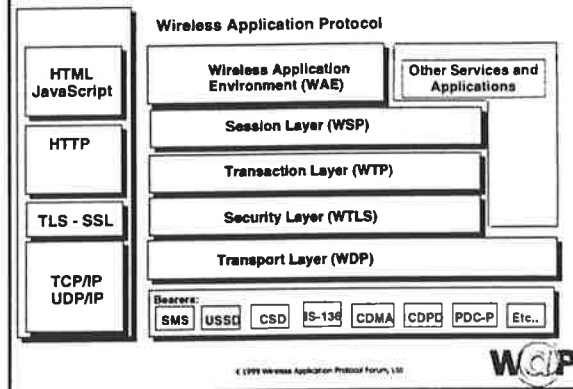
The WAP Architecture



© 1999 Wireless Application Protocol Forum, Ltd



Comparison between Internet and WAP Technologies



© 1999 Wireless Application Protocol Forum, Ltd



WAP specifies...

- Wireless Application Environment
 - WML Microbrowser
 - WMLScript Virtual Machine
 - WMLScript Standard Library
 - Wireless Telephony Application Interface
 - WAP Content Types
- Wireless Protocols
 - Wireless Session Protocol (WSP)
 - Wireless Transport Layer Security (WTLS)
 - Wireless Transaction Protocol (WTP)
 - Wireless Datagram Protocol (WDP)
 - Wireless network interface definitions

© 1999 Wireless Application Protocol Forum, Ltd



WHY WAP ?

- Wireless networks and phones
 - have specific needs and requirements
 - not addressed by existing Internet technologies.
- Only be met by participation from entire industry.
- WAP enables any data transport
 - TCP/IP, UDP/IP, GUTS (IS-135/6), SMS, or USSD.
- The WAP architecture
 - several modular entities
 - together form a fully compliant Internet entity
 - all WML content is accessed via HTTP 1.1 requests.



WHY WAP ?

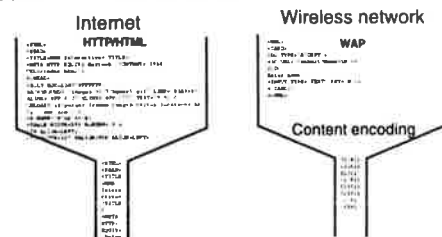
- WAP utilizes standard Internet markup language technology (XML)
- Optimizing the content and airlink protocols
- The WML UI components map well onto existing mobile phone user interfaces
 - no re-education of the end-users
 - leveraging market penetration of mobile devices
- WAP utilizes plain Web HTTP 1.1 servers
 - leveraging existing development methodologies
 - CGI, ASP, NSAPI, JAVA, Servlets, etc.

© 1999 Wireless Application Protocol Forum, Ltd



Why is HTTP/HTML not enough?

Big pipe - small pipe syndrome



© 1999 Wireless Application Protocol Forum, Ltd



WHY WAP ?

- Good relationships with standards
 - Several Liaisons with ETSI
 - ETSI / WAP compliance profile for GSM and UMTS.
 - CTIA official Liaison Officer to the WAP Forum
 - WAP is actively working with the W3C and IETF
 - HTML-NG (HTML Next Generation)
 - HTTP-NG (HTML Next Generation)

© 1999 Wireless Application Protocol Forum, Ltd



Architecture Group Current Work

- End-to-end security
- Billing
- Asynchronous Applications
- Bearer selection
- Gateway switching
- PUSH Architecture
- Persistence Definition
- Meeting format changes

© 1999 Wireless Application Protocol Forum, Ltd



WAP Application Environment

WML and WMLScript
Wireless Telephony Architecture
Content Formats
Push
User Agent Profile



WAE Goals

- Network-neutral application environment;
- For narrowband wireless devices;
- With an Internet/WWW programming model;
- And a high degree of interoperability.

© 1999 Wireless Application Protocol Forum, Ltd



WAE Requirements

- Leverage WSP and WTP
- Leverage Internet standard technology
- Device Independent
- Network Independent
- International Support

© 1999 Wireless Application Protocol Forum, Ltd



Requirements (cont.)

- Vendor-controlled MMI
- Initial focus on phones
 - Slow bearers
 - Small memory
 - Limited CPU
 - Small screen
 - Limited input model

© 1999 Wireless Application Protocol Forum, Ltd



WAE First Generation

- Architecture
 - Application model
 - Browser, Gateway, Content Server
- WML
 - Display language
- WMLScript
 - Scripting language
- WTA
 - Telephony services API and architecture
- Content Formats
 - Data exchange

© 1999 Wireless Application Protocol Forum, Ltd.



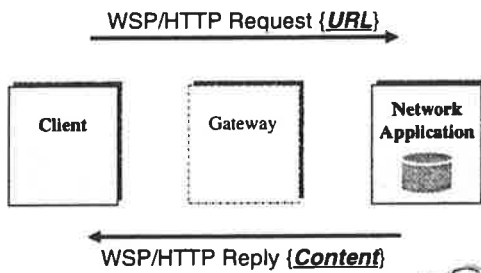
WML Second Generation

- Extensions and enhancements
 - Currently under development
- User Agent Profiling
 - Content customized for device
- Push Model
 - Network-initiated content delivery
- Performance Enhancements
 - Caching, etc.

© 1999 Wireless Application Protocol Forum, Ltd.



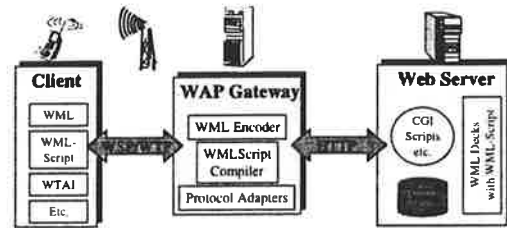
WAE Abstract Network Architecture



© 1999 Wireless Application Protocol Forum, Ltd.



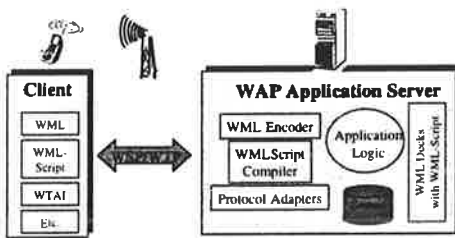
Network Example #1: WAP Gateway



© 1999 Wireless Application Protocol Forum, Ltd.



Network Example #2: WAP Application Server



© 1999 Wireless Application Protocol Forum, Ltd.



WML

- Tag-based browsing language:
 - Screen management (text, images)
 - Data Input (text, selection lists, etc.)
 - Hypertinks & navigation support
- W3C XML-based language
- Inherits technology from HDML and HTML

© 1999 Wireless Application Protocol Forum, Ltd.



WML (cont.)

- Card metaphor
 - User interactions are split into cards
 - Navigation occurs between cards
- Explicit inter-card navigation model
 - Hyperlinks
 - UI Event handling
 - History
- State management and variables
 - Reduce network traffic
 - Results in better caching

© 1999 Wireless Application Protocol Forum, Ltd



All decks must contain...

- Document prologue
 - XML & document type declaration
- <WML> element
 - Must contain one or more cards

```
<?xml version="1.0"?>
<!DOCTYPE WML PUBLIC "-//WAPFORUM//DTD WML 1.0//EN"
"http://www.wapforum.org/DTD/wml_1.xml">

<WML>
  ...
</WML>
```

© 1999 Wireless Application Protocol Forum, Ltd



WML Example

Navigation

Variables

Input Elements

```
<WML>
  <CARD>
    <DO TYPE="ACCEPT">
      <GO URL="#$card" />
    </DO>
    Welcome!
  </CARD>
  <CARD NAME="eCard">
    <DO TYPE="ACCEPT">
      <GO URL="/submit?N=${N}&S=${S}" />
    </DO>
    Enter name: <INPUT KEY="N" />
    Choose speed:
    <SELECT KEY="S">
      <OPTION VALUE="0">Fast</OPTION>
      <OPTION VALUE="1">Slow</OPTION>
    </SELECT>
  </CARD>
</WML>
```

Card

Deck



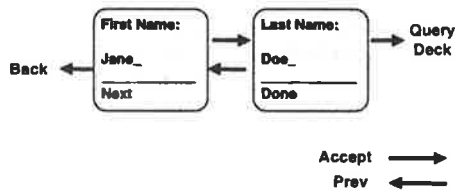
A Deck of Cards

```
<WML>
  <CARD>
    <DO TYPE="ACCEPT" LABEL="Next">
      <GO URL="#card2" />
    </DO>
    Acme Inc.<BR/>Directory
  </CARD>
  <CARD NAME="card2">
    <DO TYPE="ACCEPT">
      <GO URL="#?end=$type" />
    </DO>
    Services
    <SELECT KEY="type">
      <OPTION VALUE="em">Email</OPTION>
      <OPTION VALUE="ph">Phone</OPTION>
      <OPTION VALUE="fx">Fax</OPTION>
    </SELECT>
  </CARD>
</WML>
```

© 1999 Wireless Application Protocol Forum, Ltd



Example: Input Activity



© 1999 Wireless Application Protocol Forum, Ltd



Defining the Navigation Path

```
<CARD>
  <DO TYPE="ACCEPT" LABEL="Next">
    <GO URL="#card2" />
  </DO>
  First name:
  <INPUT KEY="fname" />
</CARD>

<CARD NAME="card2">
  <DO TYPE="ACCEPT" LABEL="Done">
    <GO URL="/?get=person" METHOD="POST"
      POSTDATA="first=$fname&last=$lname" />
  </DO>
  Last name:
  <INPUT KEY="lname" />
</CARD>
```

© 1999 Wireless Application Protocol Forum, Ltd



The DO Element

- Binds a task to a user action
 - Action type: *ACCEPT, OPTIONS, HELP, PREV, DELETE, RESET*
 - Label: *Text string or image (optional)*
 - Task: *GO, PREV, REFRESH, NOOP*
 - Destination: *URL*
 - Post data: *IF METHOD=POST*

```
<DO TYPE="ACCEPT" LABEL="Next">
<GO URL="http://www.mysite.com/myapp.wml"/>
</DO>
```

© 1999 Wireless Application Protocol Forum, Ltd



Anchored Links

- Bind a task to the ACCEPT action, when cursor points to a link
 - TITLE= sets the label string (default = "Link")
 - Links are not allowed in select list options

```
<CARD>
Please visit our
<A TITLE="Visit">
<GO URL="home.wml"/>home page</A>
for details.
</CARD>
```

Please visit
our home
page for
Visit

© 1999 Wireless Application Protocol Forum, Ltd



Task Binding Rules

- User actions are scoped at three levels
 - Deck
 - Card
 - Anchored links & select list options (ACCEPT)
- When tasks are bound to an action at different levels, the action with narrower scope takes precedence
- Default task bindings

User Action	Task
ACCEPT, PREV	PREV
Others	NOOP

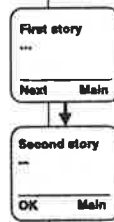
© 1999 Wireless Application Protocol Forum, Ltd



The TEMPLATE Element

- Defines actions & events for all cards in a deck

```
<WML>
<TEMPLATE>
<DO TYPE="OPTIONS" LABEL="Main">
<GO URL="main_menu.wml"/>
</DO>
</TEMPLATE>
<CARD NAME="mag1">
<DO TYPE="ACCEPT" LABEL="Next">
<GO URL="#mag2"/>
</DO>
First story
</CARD>
<CARD NAME="mag2">
Second story
</CARD>
</WML>
```



© 1999 Wireless Application Protocol Forum, Ltd



Handling User Input

- Select lists
 - Choose from a list of options
- Input fields
 - Enter a string of text or numbers
- KEY variables
 - Set by SELECT and INPUT elements
 - How user input is passed to other cards and the application server

© 1999 Wireless Application Protocol Forum, Ltd



The SELECT Element

- Display a list of options
 - Each option may set the KEY variable and/or bind a task to the ACCEPT key
 - TITLE= dynamically sets the label string

```
<CARD>
<DO TYPE="ACCEPT" LABEL="View">
<GO URL="getcity.cgi?location=$city"/>
</DO>
Forecast
<SELECT KEY="city">
<OPTION VALUE="ber">Berlin</OPTION>
<OPTION VALUE="rom">Rome</OPTION>
<OPTION TITLE="Find" ONCLICK="find.cgi">New City</OPTION>
</SELECT>
</CARD>
```

Forecast
1 Berlin
2 Rome
3 New City
Find

© 1999 Wireless Application Protocol Forum, Ltd



Other SELECT Attributes

- **MULTIPLE="TRUE"**
 - Allows user to pick multiple items
 - UP.Browser reserves soft key for item-picker
 - Key value is a semicolon-separated list
- **DEFAULT=key_value**
 - Default KEY value, if one is not chosen
 - Sets cursor to the default choice entry, if a corresponding OPTION / VALUE exists

© 1999 Wireless Application Protocol Forum, Ltd.



A Long Select List

```
<CARD>
<DO TYPE="ACCEPT">
<GO URL="get_addr.cgi?id=$recid"/>
</DO>
Add: [1..9]
<SELECT KEY="recid" MULTIPLE="TRUE" DEFAULT="1,3,5">
<OPTION VALUE="1">Neil</OPTION>
<OPTION VALUE="2">Kurt</OPTION>
<OPTION VALUE="3">Jim</OPTION>
<OPTION VALUE="4">Natasha</OPTION>
<OPTION VALUE="5">Liz</OPTION>
<OPTION VALUE="6">Aneesh</OPTION>
<OPTION VALUE="7">Jennifer</OPTION>
<OPTION VALUE="8">Jesse</OPTION>
<OPTION VALUE="9">Dawnell</OPTION>
<OPTION ONCLICK="#card2">More...</OPTION>
</SELECT>
</CARD>
```

© 1999 Wireless Application Protocol Forum, Ltd.



The INPUT Element

- Prompts user to enter a string of text
 - Use **FORMAT=** to constrain input
- UP.Browser reserves soft key for text entry mode, if necessary

```
<CARD>
<DO TYPE="ACCEPT">
<GO URL="get-person"
METHOD="POST" POSTDATA="userid=$sen"/>
</DO>
Sec Security:
<INPUT KEY="sen" FORMAT="###\-\##\-\####"/>
</CARD>
```

Sec. Security:

287-33-
NUM

Sec. Security:

287-33-7629
OK

© 1999 Wireless Application Protocol Forum, Ltd.



Other INPUT Attributes

- **DEFAULT=key_value**
 - Default KEY variable (displayed to user)
- **FORMAT=format_specifier**
 - If omitted, free-form entry is allowed
- **EMPTYOK="TRUE"**
 - Browser will accept null input, even with format
- **TYPE="PASSWORD"**
 - Special entry mode handled by the browser
- **MAXLENGTH=number**
 - Maximum number of allowed characters

© 1999 Wireless Application Protocol Forum, Ltd.



FORMAT Control Characters

- **N** Numeric character
- **A, a** Alphabetic character
- **X, x** Numeric or alphabetic character
- **M, m** Any character
- Leading backslash specifies forced characters
 - Forced characters included in KEY variable value
- Leading * specifies 0 or more characters
 - Password: **FORMAT="mmmm*m"**
- Leading number specifies 0..N characters
 - Zipcode: **FORMAT="NNNN\-\NN"**

© 1999 Wireless Application Protocol Forum, Ltd.



Displaying Images

- Insert app images or local icons within display text
 - 1-bit BMP format
- Images are ignored by non-bitmapped devices
 - Check **HTTP_ACCEPT** for "image/bmp"

```
<CARD>
<DO TYPE="ACCEPT">
<GO URL="bc2"/>
</DO>
Continue <IMG LOCALSRC="righthand"
ALT="Forward..."/>
</CARD>

<CARD NAME="c2">
<IMG SRC="..Images/Logo.bmp"
ALT="United Planet"/>
<BR/>Welcome!
</CARD>
```



© 1999 Wireless Application Protocol Forum, Ltd.



Special WML Characters

- Use character entities in display text

"	"
&	&
'	'
<	<
>	>
 	Blank space
­	Soft hyphen (discretionary line break)

- Replace the "&" character in URL strings

`URL="query.cgi?first=$fname&last=$lname"`

- Use "\$\$" to display a single "\$" character

© 1999 Wireless Application Protocol Forum, Ltd



Doing more with WML

- Setting card styles to create forms
- Using variables to cache user data
- Using card intrinsic events to trigger transparent tasks
- Using timers
- Securing WML decks
- Bookmarking decks

© 1999 Wireless Application Protocol Forum, Ltd



WMLScript

- Scripting language:

- Procedural logic, loops, conditionals, etc.
- Optimized for small-memory, small-cpu devices

- Derived from JavaScript™

- Integrated with WML

- Powerful extension mechanism
- Reduces overall network traffic

© 1999 Wireless Application Protocol Forum, Ltd



WMLScript (cont.)

- Bytecode-based virtual machine

- Stack-oriented design
- ROM-able
- Designed for simple, low-impact implementation

- Compiler in network

- Better network bandwidth use
- Better use of terminal memory/cpu.

© 1999 Wireless Application Protocol Forum, Ltd



WMLScript Standard Libraries

- Lang - VM constants, general-purpose math functionality, etc.
- String - string processing functions
- URL - URL processing
- Browser - WML browser interface
- Dialog - simple user interface
- Float - floating point functions

© 1999 Wireless Application Protocol Forum, Ltd



WMLScript Example Uses

- Reduce network round-trips and enhance functionality.
- Field validation
 - Check for formatting, input ranges, etc.
- Device extensions
 - Access device or vendor-specific API
- Conditional logic
 - Download intelligence into the device

© 1999 Wireless Application Protocol Forum, Ltd



WMLScript Example

WMLScript is very similar to JavaScript™

Functions	<pre>function currencyConverter(currency, exchangeRate) { return currency*exchangeRate; } function myDay(sunshines) { var myDay; if (sunshines) { myDay = "Good"; } else { myDay = "Not so good"; }; return myDay; }</pre>
Variables	
Programming Constructs	

© 1999 Wireless Application Protocol Forum, Ltd.



WTA

- Tools for building telephony applications
- Designed primarily for:
 - Network Operators / Carriers
 - Equipment Vendors
- Network security and reliability a major consideration

© 1999 Wireless Application Protocol Forum, Ltd.



WTA (cont.)

- WTA Browser
 - Extensions added to standard WML/WMLScript browser
 - Exposes additional API (WTAI)
- WTAI includes:
 - Call control
 - Network text messaging
 - Phone book interface
 - Indicator control
 - Event processing

© 1999 Wireless Application Protocol Forum, Ltd.



WTA (cont.)

- Network model for client/server interaction
 - Event signaling
 - Client requests to server
- Security model: segregation
 - Separate WTA browser
 - Separate WTA port
- WTAI available in WML & WMLScript

© 1999 Wireless Application Protocol Forum, Ltd.



WTA Example

Placing an outgoing call with WTAI:

WTAI Call	<pre><WML> <CARD> <DO TYPE="ACCEPT"> <GO URL="wml:cg/mc;\$(H)"/> </DO> Enter phone number: <INPUT TYPE="TEXT" KEY="H"/> </CARD> </WML></pre>
Input Element	

© 1999 Wireless Application Protocol Forum, Ltd.



WTA Example

Placing an outgoing call with WTAI:

WTAI Call {	<pre>function checkNumber(N) { if (!Lang.isInt(N)) WTAI.makeCall(N); else Dialog.alert("Bad phone number"); }</pre>
-------------	---

© 1999 Wireless Application Protocol Forum, Ltd.



Content Formats

- Common interchange formats
- Promoting interoperability
- Formats:
 - Business cards: IMC vCard standard
 - Calendar: IMC vCalendar standard
 - Images: WBMP (Wireless BitMap)
 - Compiled WML, WMLScript

© 1999 Wireless Application Protocol Forum, Ltd



New WAP Content Formats

- Newly defined formats:
 - WML text and tokenized format
 - WMLScript text and bytecode format
 - WBMP image format
- Binary format for size reduction
 - Bytecodes/tokens for common values and operators
 - Compressed headers
 - Data compression (e.g. images)
- General-purpose transport compression can still be applied

© 1999 Wireless Application Protocol Forum, Ltd



Content Format Example

Example Use of an Image:

Image Element

```
<WML>
<CARD>
  Hello World!<BR/>
  <IMG SRC="/world.wbmp"
  ALT="(Globe)" />
</CARD>
</WML>
```

© 1999 Wireless Application Protocol Forum, Ltd



Push

- Network-push of content
 - Alerts or service indications
 - Pre-caching of data
- Goals:
 - Extensibility and simplicity
 - End-to-end solution
 - Security
 - User friendly

© 1999 Wireless Application Protocol Forum, Ltd



User Agent Profiles (UAProf)

- UAProf is under development
- Goal: content personalization, based upon:
 - Device characteristics, user preferences
 - Other profile information
- Working with W3C on CC/PP
 - RDF-based content format
 - Describes "capability and profile" info
- Efficient transport over wireless links, caching, etc.

© 1999 Wireless Application Protocol Forum, Ltd



WAE Technical Collaboration

- W3C
 - White paper published
 - Technical collaboration
 - CC/PP
 - HTML-NG
 - HTTP-NG
 - Etc.
- ETSI/MEXE
- Others coming soon

© 1999 Wireless Application Protocol Forum, Ltd



Summary: WAE Status

- First generation released
 - Implementations are in progress
 - Specifications include:
 - WAE, WML, WMLScript
 - WBMP, WTA, WTAI, etc.
- Second generation in development
 - Focusing on:
 - Push, Interoperability, UAProf
 - Telephony, Internationalization, etc.

© 1999 Wireless Application Protocol Forum, Ltd

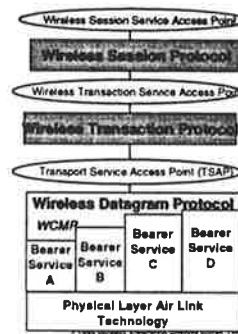


Wireless Transport Protocols

- **Wireless Session Protocol**
- **Wireless Transaction Protocol**
- **Wireless Datagram Protocol**



WAP Protocol Layers



© 1999 Wireless Application Protocol Forum, Ltd



WAP Transport Services

- WSP is the Session Layer Protocol
- WTP is the Transaction-Oriented protocol
- WDP is the Datagram protocol

© 1999 Wireless Application Protocol Forum, Ltd



WSP Overview

- Provides shared state between client and server used to optimize content transfer
- Provides semantics and mechanisms based on HTTP 1.1
- Enhancements for WAE, wireless networks and "low-end" devices
 - Compact encoding
 - Push
 - Efficient negotiation

© 1999 Wireless Application Protocol Forum, Ltd



HTTP 1.1 Basis

- Extensible request/reply methods
- Extensible request/reply headers
- Content typing
- Composite objects
- Asynchronous requests

© 1999 Wireless Application Protocol Forum, Ltd



Enhancements Beyond HTTP

- Binary header encoding
- Session headers
- Confirmed and non-confirmed data push
- Capability negotiation
- Suspend and resume
- Fully asynchronous requests
- Connectionless service

© 1999 Wireless Application Protocol Forum, Ltd.



Why Not HTTP?

- Encoding not compact enough
- No push facility
- Inefficient capability negotiation

© 1999 Wireless Application Protocol Forum, Ltd.



Header Encoding

- Defined compact binary encoding of headers, content type identifiers and other well-known textual or structured values

- Reduces the data actually sent over the network

© 1999 Wireless Application Protocol Forum, Ltd.



Capabilities

- Capabilities are defined for:
 - Message Size, client and server
 - Protocol Options: Confirmed Push Facility, Push Facility, Session Suspend Facility, Acknowledgement headers
 - Maximum Outstanding Requests
 - Extended Methods
 - Header Code Pages

© 1999 Wireless Application Protocol Forum, Ltd.



Suspend/Resume and Push

- Server knows when client can accept a push
- Multi-bearer devices
- Dynamic addressing
- Allows the release of underlying bearer resources

- Push can take advantage of session headers
- Server knows when client can accept a push

© 1999 Wireless Application Protocol Forum, Ltd.



Connection And Connectionless Modes

- Connection-mode
 - Long-lived communication
 - Benefits of the session state
 - Reliability
- Connectionless
 - Stateless applications
 - No session creation overhead
 - No reliability overhead

© 1999 Wireless Application Protocol Forum, Ltd.



Wireless Transaction Protocol

■ Purpose:

- Provide efficient request/reply based transport mechanism suitable for devices with limited resources over networks with low to medium bandwidth.

■ Advantages:

- Operator Perspective - Load more subscribers on the same network due to reduced bandwidth utilization.
- Individual User - Performance is improved and cost is reduced.

© 1999 Wireless Application Protocol Forum, Ltd



WTP Services and Protocols

■ WTP (Transaction)

- provides reliable data transfer based on request/reply paradigm
 - no explicit connection setup or tear down
 - data carried in first packet of protocol exchange
 - seeks to reduce 3-way handshake on initial request
- supports
 - retransmission of lost packets
 - selective-retransmission
 - segmentation / re-assembly
 - port number addressing (UDP ports numbers)
 - flow control
- message oriented (not stream)
- supports an Abort function for outstanding requests
- supports concatenation

© 1999 Wireless Application Protocol Forum, Ltd



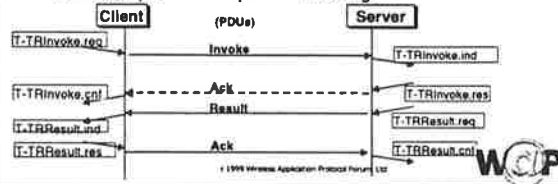
WTP Services and Protocols

■ WTP continued

- uses the service primitives

- T-TRInvoke.req.cnf..ind..res
- T-TRResult.req.cnf..ind..res
- T-Abort.req..ind

- an example of a WTP protocol exchange



© 1999 Wireless Application Protocol Forum, Ltd



WDP Services and Protocols

■ WDP (Datagram)

- provides a connection-less, unreliable datagram service
- WDP is replaced by UDP when used over an IP network layer.
 - WDP over IP is UDP/IP
- uses the Service Primitive
 - T-UnitData.req..ind

© 1999 Wireless Application Protocol Forum, Ltd



Bearers

■ Bearers currently supported by WAP

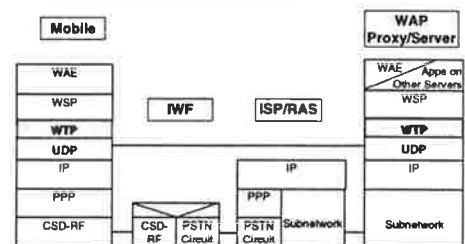
- GSM SMS, USSD, C-S Data, GPRS
- IS-136 R-Data, C-S Data, Packet
- CDMA SMS, C-S Data
- PDC C-S Data, Packet
- PHS C-S Data
- CDPD
- IDEN SMS, C-S Data, Packet
- FLEX and ReFLEX
- DataTAC
- TETRA

© 1999 Wireless Application Protocol Forum, Ltd



Service, Protocol, and Bearer Example

WAP Over GSM Circuit-Switched

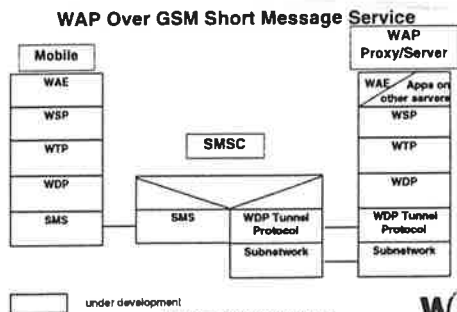


RAS - Remote Access Server
IWF - InterWorking Function

© 1999 Wireless Application Protocol Forum, Ltd



Service, Protocol, and Bearer Example



WAP Security

WTLS Services & Characteristics

WSG Work Area

- Provide mechanisms for secure transfer of content, to allow for applications needing privacy, identification, verified message integrity and non-repudiation
- Transport level security is WTLS, based on SSL and TLS from the Internet community
- Working on various mechanisms for improved end-to-end security and application-level security

WTLS Services and Characteristics

- Specifies a framework for secure connections, using protocol elements from common Internet security protocols like SSL and TLS.
- Provides security facilities for encryption, strong authentication, integrity, and key management
- Compliance with regulations on the use of cryptographic algorithms and key lengths in different countries
- Provides end-to-end security between protocol end points

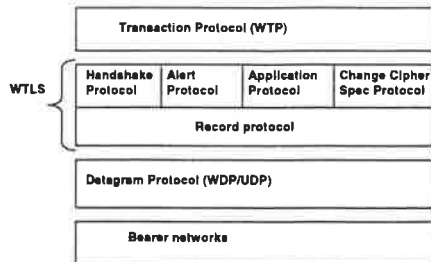
WTLS Services and Characteristics

- Provides connection security for two communicating applications
 - privacy (encryption)
 - data integrity (MACs)
 - authentication (public-key and symmetric)
- Lightweight and efficient protocol with respect to bandwidth, memory and processing power
- Employs special adapted mechanisms for wireless usage
 - Long lived secure sessions
 - Optimised handshake procedures
 - Provides simple data reliability for operation over datagram bearers

Goals and Requirements for WTLS

- Interoperable protocols
- Scalability to allow large scale application deployment
- First class security level
- Support for public-key certificates
- Support for WAP transport protocols

WTLS Internal Architecture



© 1999 Wireless Application Protocol Forum, Ltd



Services and Protocols

- Provide reliable data transfer based on request/reply paradigm
- No explicit connection setup or tear down
- Data carried in first packet of protocol exchange
- Seeks to reduce 3-way handshake on initial request
- Supports port number addressing
- Message oriented (not stream)
- Supports an Abort function for outstanding requests
- Supports concatenation of PDUs
- User acknowledgement or Stack acknowledgement option
 - acks may be forced from the WTP user (upper layer)
 - default is stack ack, 1999 Wireless Application Protocol Forum, Ltd



Classes of Operation

- WTP Classes of Service
- Class 0 Unconfirmed Invoke message with no Result message
 - a datagram that can be sent within the context of an existing WSP (Session) connection
- Class 1 Confirmed Invoke message with no Result message
 - used for data push, where no response from the destination is expected
- Class 2 Confirmed Invoke message with one confirmed Result message
 - a single request produces a single reply

© 1999 Wireless Application Protocol Forum, Ltd



Wireless Datagram Protocol

- Purpose:
 - Provide consistent interface to a fundamental transport service across all wireless bearer networks.
 - Provides a connectionless, unreliable datagram service.
 - WDP is adapted to each particular wireless network to provide the generic datagram transport.
 - The basic datagram service is fundamental to all wireless networks and makes it possible to utilize WAP everywhere.

© 1999 Wireless Application Protocol Forum, Ltd



WDP Continued

- Supports port number addressing
- WDP was initially specified for the following networks
 - IS-136 (GUTS, R-Dats, CSD, Packet Data)
 - GSM (SMS, USSD, GPRS, CSD)
 - CDPD
 - IDEN
 - Flex and ReFLEX
- WAP has since promoted specs for the following networks
 - PHS
 - PDC
 - CDMA
- Example: WDP is UDP when used over an IP network layer.

© 1999 Wireless Application Protocol Forum, Ltd



A SIP-ISDN Gateway

Hans Nilsson

CSLab

Ericsson Utvecklings AB

`hans@erix.ericsson.se`

1999-09-30

EUC'99

1

What is SIP?

- Session Initiation Protocol, RFC 2345
- Initiating, Changing and Terminating
Multimedia conferences (*IP Telephony*)
- Lightweight
- Textbased "*Looks like HTTP*"
- UDP (TCP)

1999-09-30

EUC'99

2

Future of SIP?

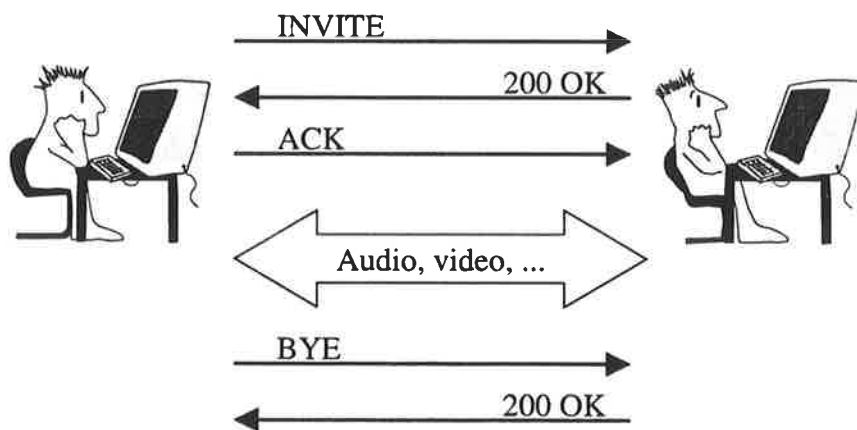
- Easy to implement => many small companies
- Mail servers, HTTP Servers, SIP servers
 - ┆ hans@cslab.ericsson.se
 - ┆ sip:hans@cslab.ericsson.se
- Web integration; Services on web pages
 - ┆ Click-to-dial

1999-09-30

EUC'99

3

SIP, Simple Example

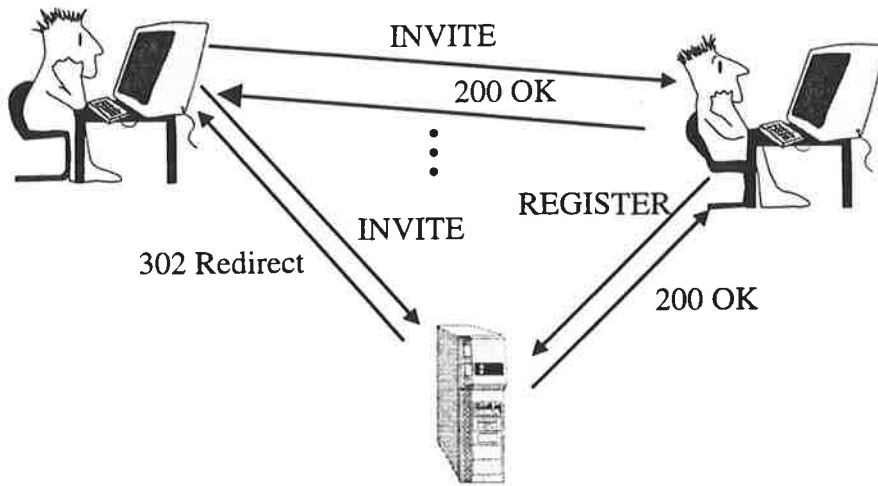


1999-09-30

EUC'99

4

SIP Redirect Server Example

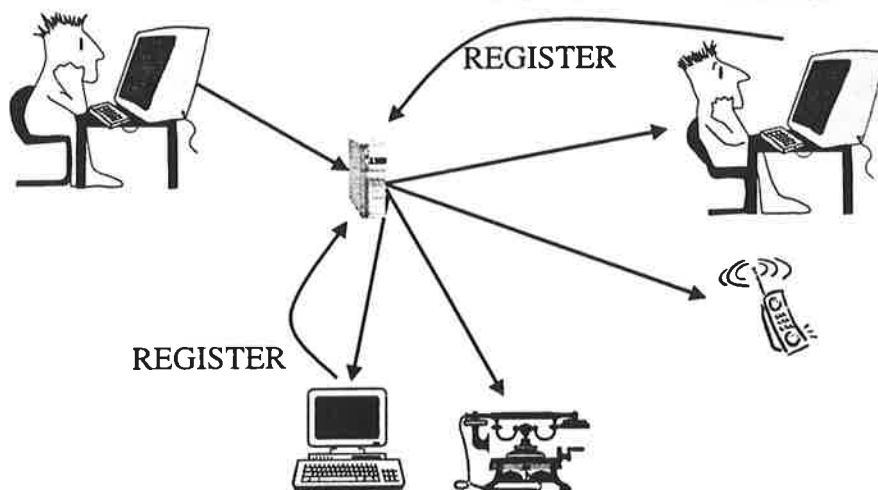


1999-09-30

EUC'99

5

SIP, Other Example

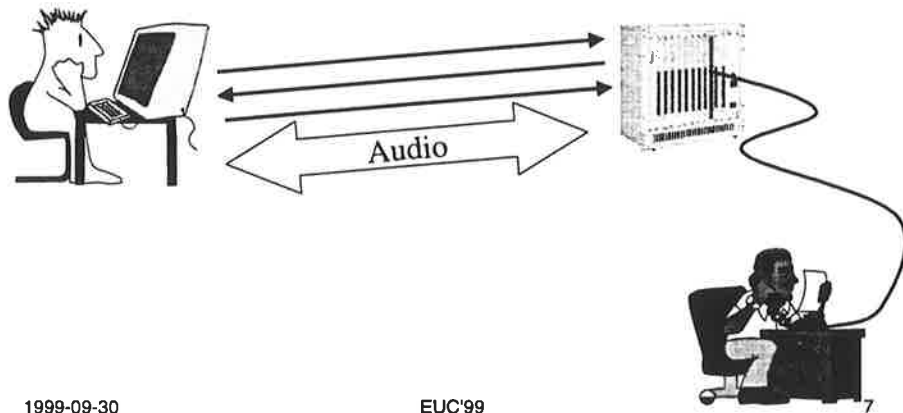


1999-09-30

EUC'99

6

What is a Gateway?

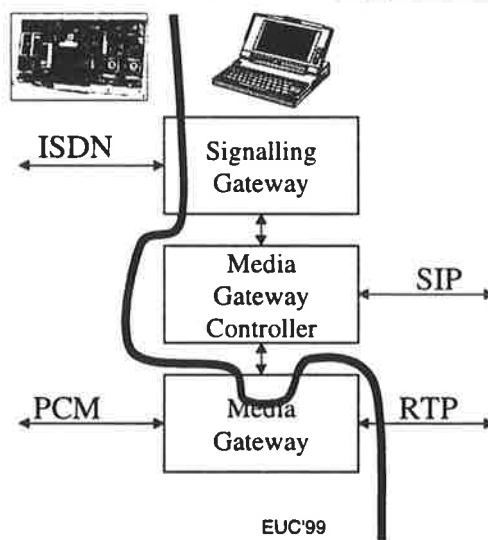


1999-09-30

EUC'99

7

SIP/ISDN Gateway

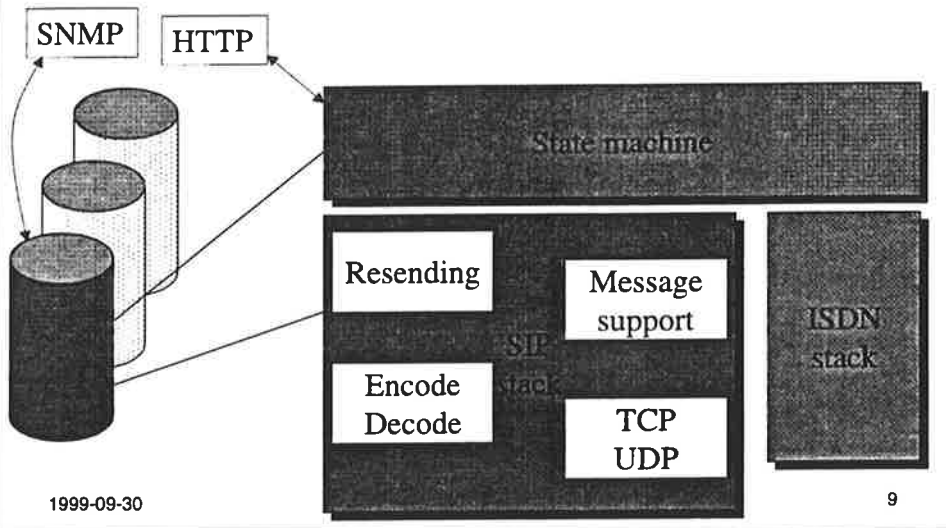


1999-09-30

EUC'99

8

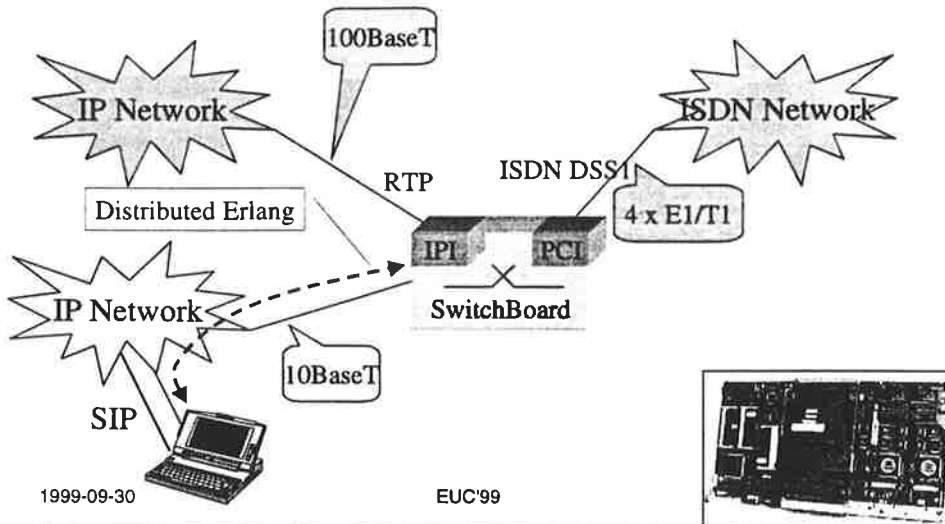
SIP - ISDN



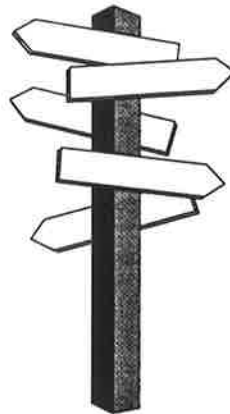
Erlang code sizes in number of source lines

Encode/decode	1511
SIP stack	2476
State Machine SIP <-> ISDN	2285
ISDN stack	6862

SIP/ISDN Gateway - realisation



Why Erlang/OTP?



Some **ERLANG** characteristics

[http://www.erlang.org/white_paper.html]

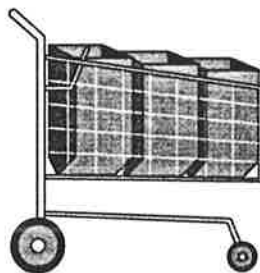
- Concurrency
 - ┆ lightweight processes
- Distribution
 - ┆ transparent message passing
- Robustness
 - ┆ node fail-over
 - ┆ process monitoring
- Soft real-time
- Hot code upgrade
- External interfaces
- Components/Libraries:
 - ┆ Mnesia
 - ┆ SASL
 - ┆ SNMP
 - ┆ Inets
 - ┆ GS
 - ┆ ...

1999-09-30

EUC'99

13

Users

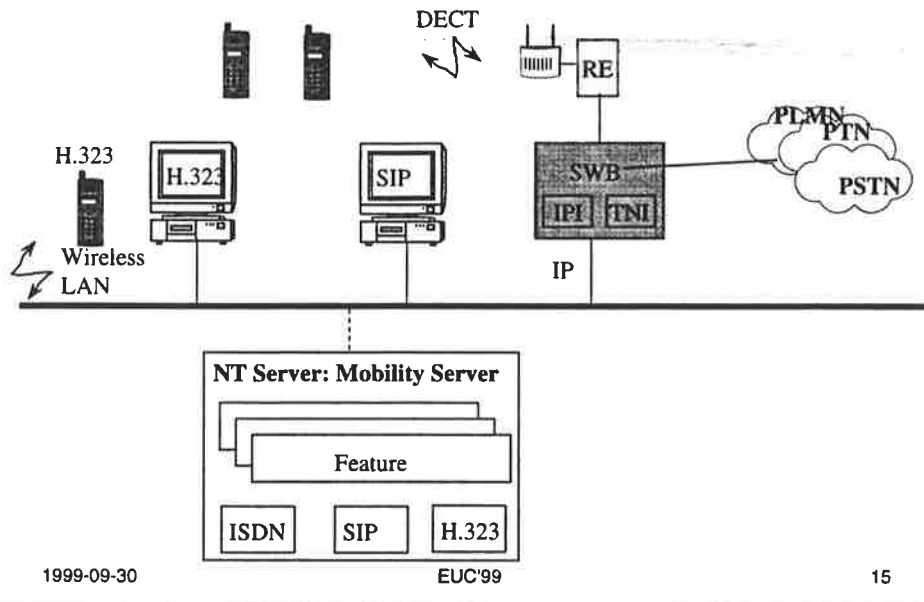


1999-09-30

EUC'99

14

Mobility Server VoIP Prototype Configuration



Mobility Server VoIP Prototype Objectives

- Show Inter-operability between IP Terminals and DECT Terminals or PSTN
- Identify effects from implementing existing Mobility Server services over VoIP
- Create qualified arguments on SIP versus H.323 for the Mobility Server product

The Future



1999-09-30

EUC'99

17

Future: **Work, work, work...**

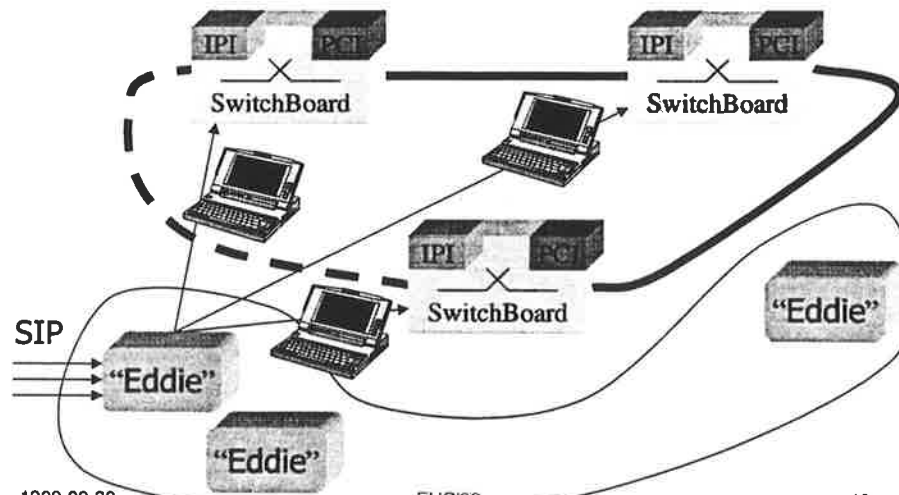
- Standard MG protocol (MGCP,...)
- Less prototypish :-)
- Administration:
 - SNMP
 - HTTP/CGI
- Connect to Ericsson Phone Network

1999-09-30

EUC'99

18

Redundancy and Capacity



1999-09-30

EUC'99

19



Next Erlang/OTP Release - Highlights

CosTransaction

Secure IIOP over SSL

Release handling improvements.

Disklog improvements.

ASN.1 extended standard support.

Extended support for dynamic reconfiguration
and code update in Mnesia.

Performance improvements.

Stacktrace and other trace improvements

New compiler

NO JAM

Release date October 27, 1999

Open source available ---

Towards an Event Modeling Language

Maurice Castro Michael Dwyer Geoff Wong
maurice@serc.rmit.edu.au dwyer@serc.rmit.edu.au geoff@serc.rmit.edu.au

Software Engineering Research Centre
Level 3, 110 Victoria St
Carlton, VIC, 3053, Australia

Abstract

Object-oriented programming owes part of its popularity to Booch's notation [2], Rumbaugh's OMT [7], and UML [8]. These notations allow object oriented designs to be expressed graphically and furthermore have enabled the development of case tools for object oriented languages.

The lack of a suitable high level graphical notation has been identified as one of the factors discouraging the uptake of functional programming and in particular the language Erlang [1].

This paper represents a first step in designing a graphical modeling language for functional programming that encourages sound programming practices. The initial target language is Erlang but it is hoped that the notation can be extended to other functional languages.

1 Introduction

Graphical notations such as Booch's notation [2], Rumbaugh's Object Modeling Technique (OMT) [7], and the Unified Modeling Language (UML) [8] have provided a strong impetus to the Object-oriented (OO) programming paradigm.

The absence of a high level graphical representation for functional programming has been identified as one of the factors inhibiting the uptake of the Erlang [1] programming language.

Graphical design notations allow designers to specify programs at a high level in a manner more closely related to the programming paradigm rather than the target language of the project. The notation should aid communications throughout the product life-cycle, but in particular they should aid communications between designers and implementors by highlighting design issues and leaving programmers free to handle the detail of the implementation. Finally these notations are instrumental in the construction of case tools by providing a high level notation which can be translated into programming language constructs.

This paper describes the first steps towards a graphical notation for the Erlang programming language.

In the following sections the Erlang programming language is summarised and its significant features identified, existing techniques are discussed, the ob-

jectives of the notation are discussed, the notation is introduced, and several examples of the notation are given.

2 Erlang

The Erlang language [1] is a syntactically impoverished functional language employing pattern matching for rule selection. The language has support for concurrent programming and is suitable for soft real time applications.

2.1 Functions & Modules

Erlang functions return a single result which may be assigned to a variable or used as an argument to another function call. Functions are constructed from clauses which are constructed from function calls and the operators of the language. Functions are arranged into modules which are the unit of code loading. Modules are also the unit of code distribution in a distributed system. It should be noted that no storage is associated with a module and that modules are only collections of functions.

2.2 Types

The language is dynamically typed. Variables are untyped hence any data element may be bound to any variable.

The language supports 7 simple data types: integers, floats, atoms, Process IDs (PID), references, binaries and ports. Atoms are a constant name with a value equal to their own name. PIDs uniquely identify a process. References are values which are guaranteed to be unique. Binary data objects are used to encapsulate sequences of bytes. Ports are used to communicate with the external environment.

The compound data types of the language - tuples and lists - use position within the structure to identify fields rather than names. The arity or number of elements in a tuple is fixed at compile time. The number of entries in a list is determined at runtime and operators are provided which allow elements to be added to or removed from a list.

Type errors result when either bindings are attempted between incompatible structures or a built-in function¹ or operator is applied to an inappropriate type.

2.3 Process Model

Processes are the unit of execution. A process consists of a stack, an associative store known as a process dictionary, a message queue, and a thread of execution through a piece of code.

2.4 Messages

A message may contain any data item and is addressed to exactly one process by naming either a PID or a symbol which translates to a PID (registered name). The following properties of a message are guaranteed:

¹Built-in Functions (BIF) are defined as the set of functions provided by the interpreter in an interpreted Erlang system

- to be sent immediately
- to be received without errors if received at all

However, messages are not guaranteed to be delivered.

Erlang also defines the concept of a node. Processes within the same node see the same set of translations between registered names and PIDs. In practice a node is an instantiation of the Erlang interpreter. Processes may be started on a local node or a remote node.

2.5 Dominant Language Features

For the purposes of design, processes and messages are the dominant features of the Erlang programming language. Erlang programs can be modeled as a collection of communicating finite state automata.

Processes may be represented as a finite state automata which jump from internal state to internal state depending on a set of rules encoded as functions. Messages represent the set of events which allow processes to change from state to state.

2.6 Dominant Design Features

Processes are relatively inexpensive and easy to use in Erlang, differentiating Erlang from many other languages and forcing processes to be considered as a major design element.

Generic servers are relatively simple to write and their functionality is distinguished by passing in the module and function name of each function to be called when the server receives a particular event.

3 Existing Notations

Isaksson and Stureson [6] identify a number of existing design notations employed by Erlang programmers. The notations included State Transition Diagrams (STDs) and Data Flow Diagrams (DFDs). This section briefly discusses these techniques and identifies the need for another notation to describe the design of an Erlang program.

3.1 State Transition Diagrams

The State Transition Diagram and its related tabular form represent a process or system as a collection of states connected by a set of transitions. Associated with each transition is a pre-condition and an action – the action may be null. In the diagrammatic form states are usually represented by circles, transitions are represented by arcs and the arcs are annotated with the pre-condition and the action.

When pre-conditions correspond to the presence of a message, STDs are well suited to representing the action of a single process and are typically easily translated into Erlang code. It should be noted that STDs do not provide a one-to-one correspondence with Erlang as Erlang's receive function employs pattern

matching and hence may place an ordering on the received messages. STDs have no mechanism to represent this ordering.

STDs do not offer any assistance in defining the interactions between processes.

3.2 Data Flow Diagrams

Data Flow Diagrams represent the flow of data through a system and the transformations applied to the data at each stage. External entities are represented by boxes, transformations are represented by circles, data items are indicated by arrows, and data stores are represented by a pair of parallel lines.

The mapping between DFDs and Erlang is ambiguous. Specifically, the transformations in a DFD can be modeled as either Erlang processes or function calls within a process, and data flows can be either modeled as the arguments of a function call or a message passed between Erlang processes.

3.3 Deficiencies

Neither STDs nor DFDs allow processes to be shown explicitly with their interfaces. This deficiency prevents easy interfacing to an existing process structure.

The notation proposed in this paper allows processes and their public interfaces to be described. The proposed notation is not intended to replace either of the existing notations completely instead it provides a way of explicitly describing the relationships between processes. STDs may be used in conjunction with the proposed notation to describe the procedural aspects of the operation of a process. DFDs may remain a useful first step in specifying the system. However, it is envisaged that DFDs will have reduced usefulness as the design progresses and the proposed notation is employed.

4 Graphical Design Language Features

This section identifies forces which have influenced the notation presented.

The language was required to be:

- Draw-able by hand without the need for special templates.

As a considerable amount of design work involves a group of people around a white board and one on one communication between a designer and an implementer may need to be carried out at varied locations where only a pencil and paper is available, it was considered essential that the language be usable without special apparatus.

- Machine translatable into skeleton files which can be used by programmers as a starting point for writing programs implementing the design.

This criteria allows CASE tools to be constructed.

- Useful for both forward and reverse engineering a design.

Although the language was introduced to aid in the design stages of a project, it was considered desirable that the language should be usable to draw a representation of an implemented system. This extends the

usefulness of the language into the maintenance phases of projects and into iterative development environments. By making the language useful in the maintenance phase the chance of retaining a correct and useful design is increased, this design can be used as a starting point in later design cycles.

The ability to perform both forward and reverse engineering allows the CASE tool to 'drive' the development process by allowing the tool to be used at each stage of the life cycle.

- Recursive in nature.

It is rare that a complete design for a complex system can be written down immediately and completely. The language was required to allow stepwise refinement of the design. By making language elements recursive a design can be drawn and then refined by redrawing an element at a higher level of detail. This property can be used in two ways: The top level diagram can be re-drafted to contain the refined element, or sub-diagrams can be constructed providing greater detail than the top level diagram.

Graphical languages need to balance the amount of information displayed in written form against overcrowding the diagram. Furthermore, the number and types of elements must be restricted to make the diagrams sufficiently simple to take in at a glance.

Where possible familiar notations from other languages should be used. Conversely, notations which resemble those used in other languages must have an equivalent meaning in the new notation to prevent confusion.

The evolution of OMT [7] and Grady Booch's method [2] into UML [8] suggested that rectangular boxes with square corners were the only acceptable geometric form for a widely used graphical notation.

The ease of writing generic server processes which are distinguished only by the set of functions that they call in reaction to a stimulus makes it essential that the notation support a way of representing a process started with specified values for formal parameters. Of particular interest are module and function names used at the instantiation of the server.

5 Notation

The event model is abstracted into instantiation, processes and interactions. Instantiation is the act of bringing a process into existence. Processes are entities which are in a state and may change state as a result of an interaction. Interactions are events. Typically an interaction is a message or messages sent between processes.

The notation supports recursion. Processes may specify either processes in an implementation, a collection of processes in an implementation, or a collection of processes defined in another event model diagram.

Sources of events outside the model may be implemented as dummy processes.

5.1 Processes

A process (see figure 1) is represented by a box with the following compartments:

- Name compartment: the left side of this compartment contains either the name of the process or an 'X' to indicate that the process is known only by its system allocated PID. The right side may contain an optional (though recommended) unique name which is used to refer to the process within the diagram.
- Startup compartment: This compartment is divided into 2 sections with the second section optional. A single line is used to separate the two sections if the second section is present. The first section is called the parameter section and it contains: the module name, function name and arguments used to start the process. The second section is called the instantiation section and it contains a list of relations. Each relation consists of a formal parameter named in the parameter section, an equals sign, and the value assigned to the parameter when the process is started. Formal parameters maybe unmentioned in which case the designer is not specifying the initial value at this time.
- Process life compartment: A set of symbols indicating the lifetime of a process. An infinity symbol (∞ , the symbol 'inf' may be used where a symbol font is not available) indicates that the process is intended to live indefinitely. If the process is intended to live for a fixed period of time a tau (τ , the symbol 't' may be used if a symbol font is not available) symbol is used. If an estimate of the time or an upper bound on the time is known then an equation using tau may be used. If the process is intended to live to perform a number of jobs and then cease, the letter 'n' is used to denote the number of major tasks the process is to perform.
- Interface compartment: A set of interface boxes, 1 for each public interface provided.

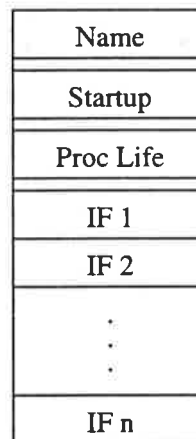


Figure 1: A process

Erlang supports both a direct message based interface and a message based interface that has been encapsulated in functions. An interface box consists of a unique name for the interface typically signifying a purpose for the interface,

and either a description of a set of message structures or a set of module names and function names of the interface functions.

An interface consists of a set of message types or functions that logically belong together to achieve some common task.

A dummy process has no interface compartments and the startup and interface compartments are empty. This type of process represents a source of events from originating from outside the model.

5.2 Instantiation

A line with a double arrowhead (see figure 2) is used to indicate the creation of a process. The parent of the process appears at the blunt end of the arrow. The sharp end of the arrow points towards the child. If the arrow points to a solid dot then multiple processes may be created. A number near the dot indicates the number of processes created. Associated with each line is a piece of text which describes the reason for creating the process.

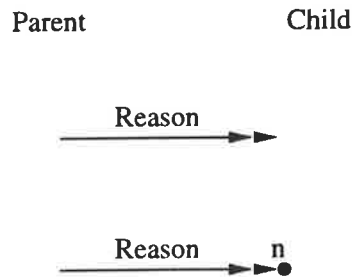


Figure 2: Instantiation

5.3 Interactions

Interactions (see figure 3) are represented as lines with single arrowheads. Interaction lines run from process boxes to interface boxes. It is illegal for a line to connect two interface boxes. If an encapsulated interface is used and a function returns a result caused by receiving a message then a double barb should be used. The double barb indicates the flow of information in two directions.

Associated with each interaction line is a reason for the interaction.

6 Design Technique

The first step in designing an Erlang program is to determine the major concurrent activities of the system. Each of these activities is then defined as a process on the diagram. After the processes are drawn, interactions between the processes are indicated with lines interconnecting processes. Interface boxes are added to the design to provide the detail of the interactions interface. This top level diagram may be stepwise refined by defining processes which compose the processes provided by the top level diagram.

A number of useful 'patterns' are available to the designer:

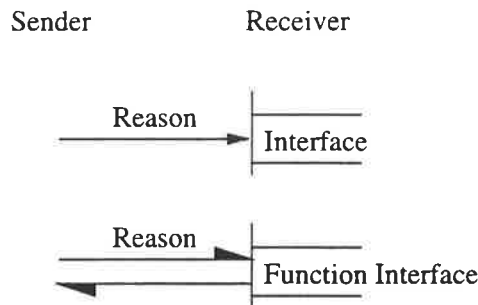


Figure 3: Interactions

- Worker processes - As processes are cheap in Erlang designers may wish to start a worker process to complete some task
- Immortal servers - Most systems will have some co-ordinating process, typically this process will be immortal as this simplifies the problem of contacting the controlling process

A number of decisions are deferred to the implementation:

- The size and borders of the process name space
- The placement of processes on nodes

Description of the program logic can be provided by associating a finite state automata, a message sequence chart, or a textual description with each process.

Although the notation supports both encapsulated and non-encapsulated interfaces. Designers are encouraged to use encapsulated interfaces in nearly all cases. Non-encapsulated interfaces will probably only be used for exposed system interfaces such as `gs` [4] or `Internet` [5] libraries.

Designers are encouraged to abstract common process patterns into generic servers which call functions by name in response to a stimulus. This approach minimises maintenance problems by abstracting common code.

7 Examples

This section develops a simple example and a more complex example of the application of the notation. The section closes by commenting on the design process in relation to protocol stacks.

7.1 Simple Example

WWW servers are easily constructed in Erlang. The notation is used to describe a minimalist WWW server [3]. The server consists of a process that listens for incoming connections and then spawns a new process to handle the request contained in the connection. Figure 4 illustrates the approach.

In addition to the graphical representation of the process a short description of the logic of the *server* and *reqhd* processes is required. The *erts* process

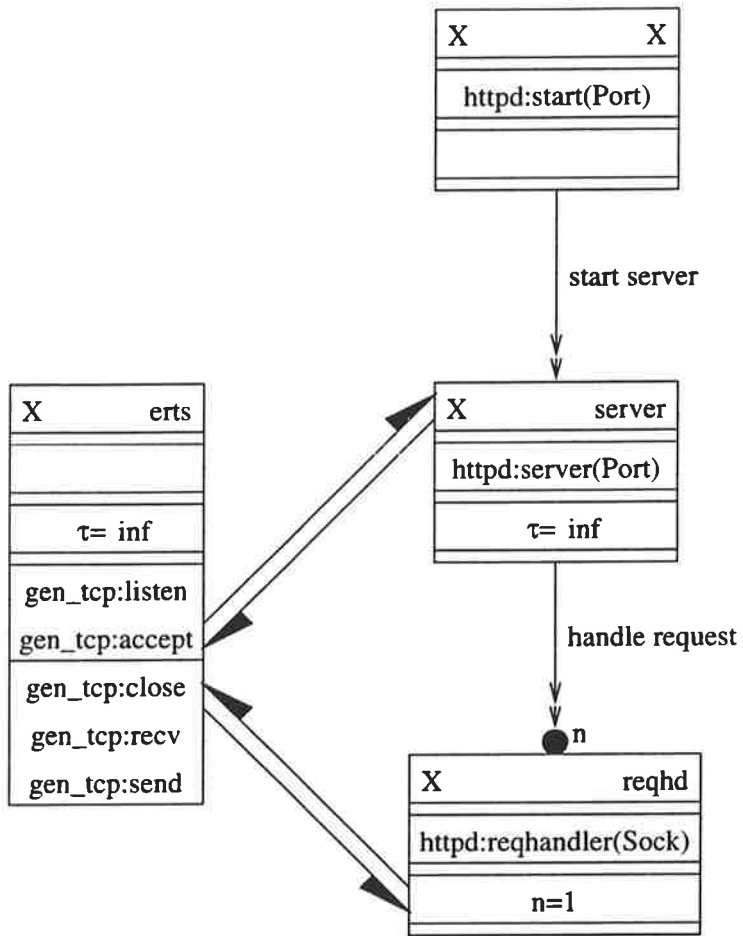


Figure 4: A simple WWW server

represents facilities provided by the Erlang runtime system that are used by the WWW server. The *I* process is used to represent any process that calls the *httpd:start* function and hence starts the server.

7.2 More Complex Example

The second example is an abridged representation of an authenticating WWW proxy server. Many IP networks employ a firewall to control what data flows into and out of their network. A firewall is a computer with at least 2 network interfaces. It prevents traffic from passing from one interface to another until a condition allowing the transfer has been met. One example of such a rule would be that traffic would be prevented from flowing until after the user has successfully identified themselves. One method for implementing an authentication for WWW page access on a firewall is to create a proxy server on a random port on the firewall after the user has successfully authenticated themselves.

This example illustrates the use of a generic server as the basic WWW server is abstracted and reused. Figure 5 shows the major processes of the system. The server code is reused in both the *auth* and *authprxy* processes. The two processes differ only that they are instantiated with the names of different functions associated with the action of getting a new page.

7.3 Protocol Stacks

Protocol stacks are typically a special design case in that they can be implemented either in a single process or in several processes. The design approach presented here is relevant to protocol stacks. As these diagrams do not seek to show the procedural aspects of program flow, a protocol stack implemented in process may not even appear on these diagrams. The authors do not view this as a problem as they consider that the procedural aspects of design are well covered using either a narrative description or STDs, however, current notations neglect processes and their dynamic nature.

8 Iterative Application

The proposed method is recursive and can be applied iteratively. This section will show how the recursion property can be exploited to perform stepwise refinement on a design.

The more complex example (see section 7.2) provided a starting point for an implementation. Further study of the problem indicated that it was simpler and more desirable to separate the roles of authentication and proxying. Furthermore, it was found that it was necessary for the child process which handles an individual connection to be able to shut down the parent WWW server process. This facility would be used if a user failed to authenticate themselves correctly.

Figure 6 shows the separation of the authentication and proxy roles. Note that this could also have been represented in a subdiagram providing greater detail to the combined *authprxy* process (see figure 7). The subdiagram shown includes the interaction with *erts* although this is optional.

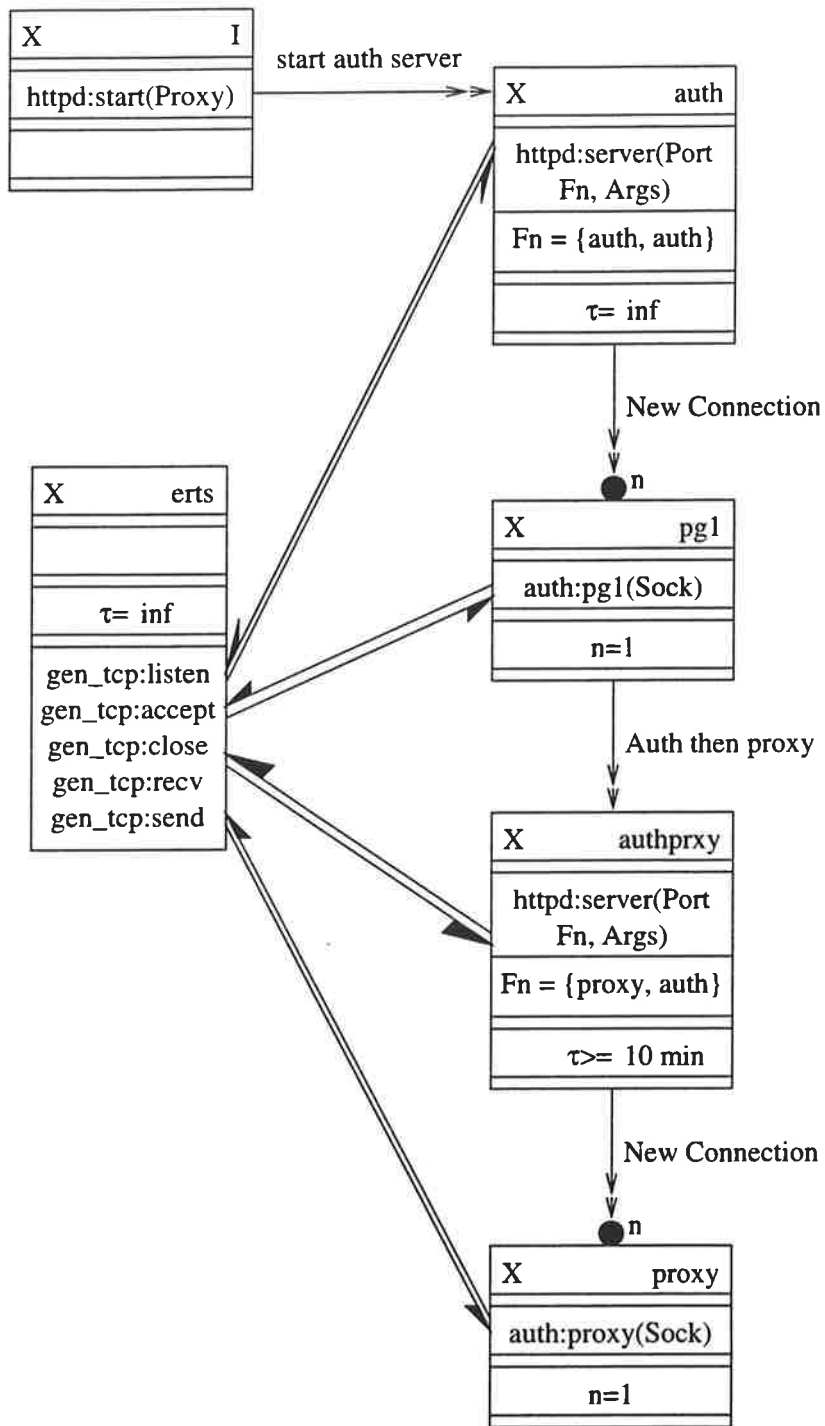


Figure 5: An authenticating WWW proxy server

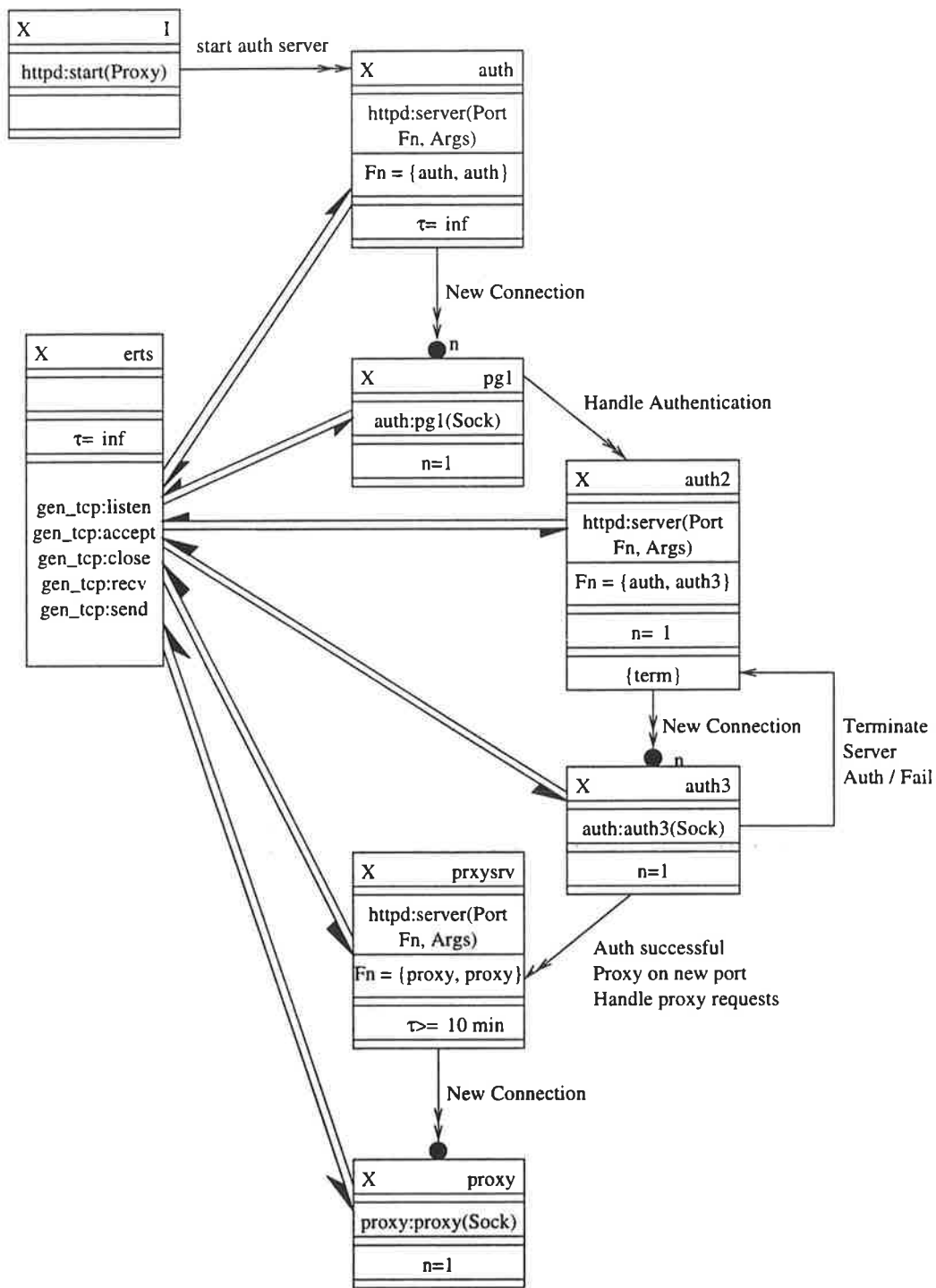


Figure 6: Modified authenticating WWW proxy server

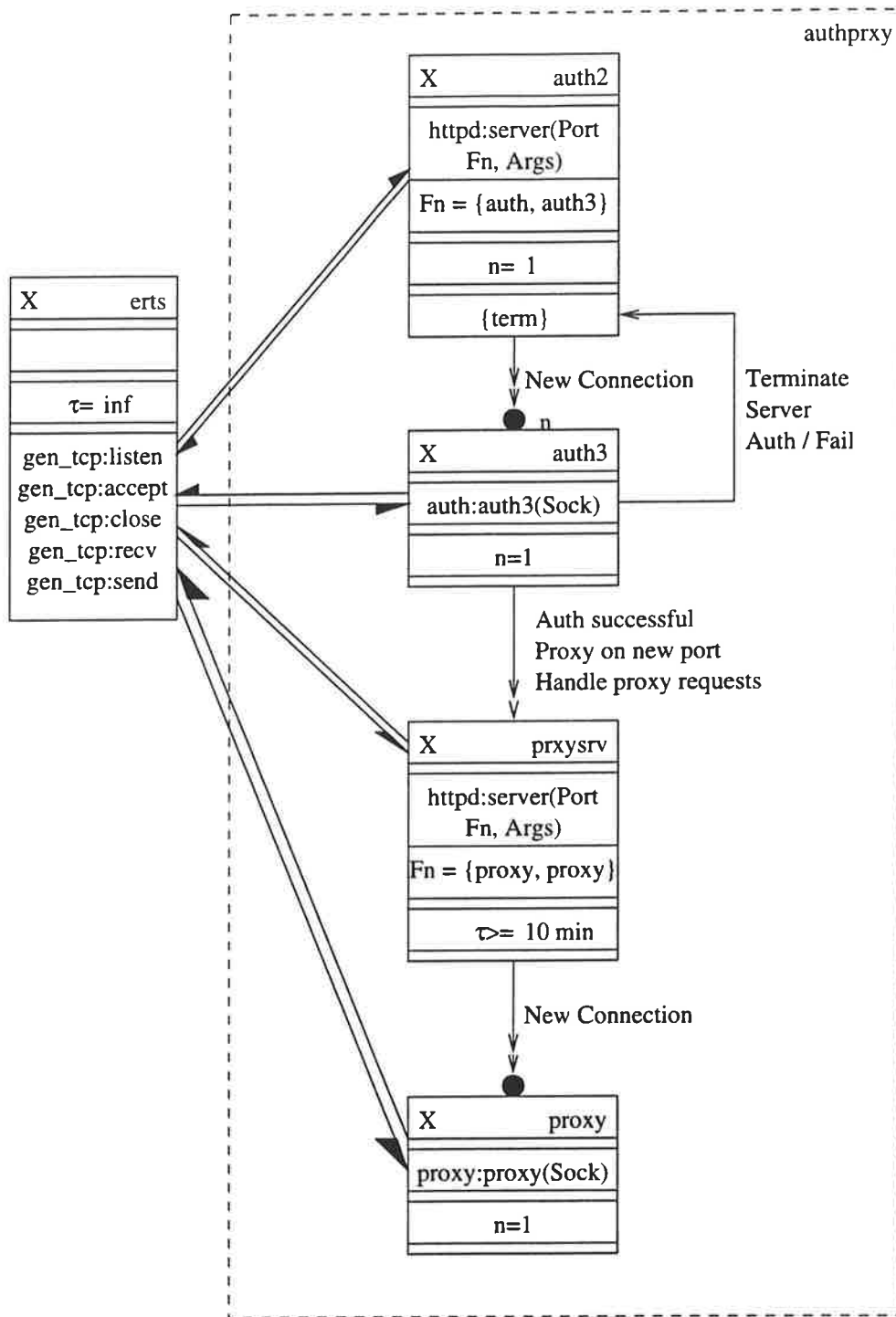


Figure 7: Detail of *authprxy*

9 Conclusion

The beginnings of a notation and a design technique for Erlang applications have been introduced. The notation is intended to assist both in the design and maintenance phases of project development. At this stage the notation has been used on only a few examples which have been strongly connected with WWW applications, however, the authors have found it helpful in both formulating and communicating their ideas. The notation has been shown to be suitable for reverse engineering existing code, and has been shown to be readily modifiable as design ideas change.

References

- [1] Joe Armstrong, Robert Virding, and Mike Williams. *Concurrent Programming in Erlang*. Prentice-Hall, 1993.
- [2] Grady Booch. *Object-Oriented Analysis and Design with Applications*. The Benjamin/Cummings Publishing Company, second edition, 1994.
- [3] Maurice Castro. *Erlang in Real Time*. Department of Computer Science, RMIT, 1998. ISBN 0864447434. Also available from <http://www.serc.rmit.edu.au/~maurice/erlbc/>.
- [4] Klas Eriksson. *Graphics System Application (GS) version 1.3*. Ericsson Telecom AB, 5 1997.
- [5] Magnus Fröberg. *Kernel Application (Kernel) version 2.1*. Ericsson Telecom AB, 11 1997.
- [6] Johanna Isaksson and Elinor Sturesson. Design guidelines for erlang. Technical Report SERC-0083, Software Engineering Research Centre, 1 1999.
- [7] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorenzen. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.
- [8] Rational Software, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing, IntelliCorp, i-Logix, IBM, ObjecTime, Platinum Technology, Ptech, Taskon, Reich Technologies, and Softeam. *UML Semantics version 1.1*. Rational Software, 9 1997. [http://www.rational.com/uml/\(ad/97-08-04\)](http://www.rational.com/uml/(ad/97-08-04)).

Protocol programming in Erlang using binaries

Claes Wikström Tony Rogvall*
Computer Science Laboratory
Ericsson Telecom AB

29 Sep 1998

1 Introduction

This text is an introduction to the binary data object found in an experimental version of ERLANG. It is also a general purpose tutorial on protocol programming in ERLANG in general.

Binaries are basically a sequence of octets. The operations we can perform on binary objects as well as the internal representation of binaries are primarily intended to be used as a means for efficient as well as beautiful protocol implementations. However a wide variety of applications can take advantage of binaries. In particular all applications that perform extensive IO, such as disk IO, GUI systems, or networking TCP/IP IO.

2 Background

Implementing protocols correctly is hard. Implementing them efficiently is even harder. In particular if the protocol implementation is supposed to be a part of a greater whole, i.e. a very large system application, it is hard. The protocol must be implemented in a flexible manner which fits into the general model of IO of the larger application. If the protocol implementation is completely stand alone from the application which uses the protocol, buffer management suffers since then typically buffers must be copied from the protocol implementation to the application utilizing the protocol and vice versa. In the extreme, buffers are copied between protocol layers, which hurts execution speed bad.

We believe that the binary data object implementation we propose in this document is well suited for protocol implementations. In particular in combination with the other facilities previously found in the ERLANG programming

*email: {klacke,tony}@erix.ericsson.se

language we have a set of tools which makes it remarkably easy to implement protocols.

The different protocol implementations found in this document have been chosen for their reputation, not because any of them is considered to be particularly hard to implement. The advantage of this is that all readers are familiar with the protocols at hand.

2.1 History

The binary data object saw the light of day in the Erlang 3.3 release approximately 1992. The first application for binaries was to provide an efficient container for object code. Object code was read from a file and loaded directly into the system. This work was done in the code loading BIFs themselves and this was deemed inappropriate for a variety of reasons, mainly portability and flexibility. Thus the binary data object was introduced and the code loading BIFs were changed to take a binary object as input parameter instead of a filename.

Later on, the binary object was used to carry messages in distributed Erlang. Once the binaries were introduced into the system ERLANG users started to use binaries for all kinds of different purposes. In particular they have been used as IO buffers in various protocol implementations. The original binary object was fairly suitable for this but not perfect. We have now tried to remedy this situation by both providing a new internal implementation of binaries as well as providing means at the language level to efficiently build and match binaries.

3 Introduction

Binaries can be constructed and pulled a part by means of matching in a way similar to regular ERLANG lists. New binary objects are created as:

```
< Part1:Size1/Modifier1 , Part2:Size2/Modifier2 ..... >
```

The **Part** is an erlang value we want to use for the construction of the binary, the **Size** part is the number of bits (or sometimes bytes) we want to use from the **Part** value and finally the **Modifier** is a dash separated sequence of type modifiers.

This is possibly best illustrated with a simple example. For example the expression

```
<777:16/integer>.
```

creates a 16 bit (two bytes) binary object consisting of the two least significant bytes from the integer 777. A wide variety of types, and modifiers are available.

Say we have some bit pattern bound to an integer value as:

```
X = 16#abcd,
```


Now we want to construct a four byte binary which has the equivalent contents as the list:

```
int32_to_list(X) ->
[(X bsr 24) band 255, (X bsr 16) band 255,
 (X bsr 8) band 255, X band 255].
```

Why would we want to do such an operation in the first place? Say we want to write an ERLANG term, represented as a binary object in the external term format to a file. Furthermore we want to prepend the size of the term in the file as in:

```
B = term_to_binary(SomeTerm),
write(FileDesc, [int32_to_list(size(B)) , B]).
```

I.e, we want to serialize an integer in order to use it in an output operation. This operation is directly supported in the language through the binary constructor syntax. The function `int32_to_list/1` returned a list, it is better to directly return a binary.

```
int32_to_bin(X) -> <X:32/unsigned-integer>.
```

Thus the previous sequence where we write a term to a file would become.

```
B = term_to_binary(SomeTerm),
write(FileDesc, <(size(B)):32/unsigned | B>).
```

Integers come in different varieties. Size, sign and endianness¹ may vary.

The size field in a binary expression indicates how many bits of the value we want to use. For example: `<255:4/integer, 255:4/integer>` evaluates to a one byte binary object holding the value 255. We can try this out at the shell prompt:

```
1> <255:4/integer, 255:4/integer>.
<255>
```

In a manner similar to how we build binaries we can match a binary object when we need to pull a binary apart. We use a syntax which is equivalent to the construction syntax when we match binaries. So for example to pull apart some of the objects we constructed in the previous section we can write:

¹The term "endian" can be traced to Jonathan Swift's novel "Gulliver's Travels". In one of Gulliver's adventures, he encounters an island whose inhabitants bitterly argue over the correct way to open soft-boiled eggs - the little end or the big end. Little endians and big endians are each convinced that their method is the only correct method for opening an egg.

```
X = <255:4/integer, 255:4/integer>,
<Hi:4/integer, Lo:4/integer> = X,
{Hi, Lo}.
```

The above code first constructs an 8-bit binary and then pulls it apart.

It is also possible to mix the match on different types. We wrote a term to a file by means of the call

```
write(FileDesc, <(size(B)):32/unsigned | B>).
```

Similarly we can pull the same binary apart as

```
case Binary of
  <X:32/unsigned-integer|Btail> when size(Btail) == X ->
  .....
```

The above construction is particularly common where we have a `size` field which determines the size of some later fields. When we match such values it is very convenient to be able to use a variable introduced in the actual match pattern as a size indicator for objects in the same pattern. This is indeed possible and the above code could be written in a better way as:

```
case Binary of
  <X:32/unsigned-integer, BValue:X/binary | Tail>
  .....
```

Note how the `X` variable is introduced first in the pattern and then later on used in the same pattern.

4 Types, Compiler directives and default values

Many different erlang types can be used to match and construct binaries. In this section we list all erlang types that are supported for binary construction and matching as well how the different modifiers and size specifications apply to the different types.

4.1 Integers

4.1.1 Constructing

The size of a constructed integer is not constrained by anything but the size of the largest integer that is possible to represent.

The following list of type modifiers are applicable to integers.

- `integer` Indicates that we want to pack an integer.

- **little** The integer is packed in little endian byte order.
Example: `<4042:16/unsigned-integer-big>` produces a 2 byte binary containing the integer 4042 represented in big endian byte order.
- **big** The integer is packed in big endian byte order. The **little** and **big** type modifiers are only applicable when the size indicator is a multiple of eight.
- **signed**, **unsigned** These type modifiers are allowed but don't mean anything when constructing a binary.

4.1.2 Matching

When we match integers from a binary we have the following valid type modifiers.

- **integer** Indicates that we want to match an integer.
- **little** The integer in the binary buffer is considered to be packed in little endian format.
- **big** The integer in the binary buffer is considered to be packed in big endian format.
- **signed** The integer in the binary buffer is considered to be represented with a 2-complement sign extension.

Example:

```
B = list_to_binary([255,255]),
<X:16/signed-integer> = B,
```

will bind the variable X to -1.

- **unsigned** The integer in the binary buffer is considered to an unsigned integer. Considering the same example as above with:

```
B = list_to_binary([255,255]),
<X:16/unsigned-integer> = B,
```

will bind X to the value 65535

4.2 Characters

The same modifiers that apply to integers apply to characters as well except that the modifier **integer** must be replaced with the modifier **char**.

4.3 Floating point numbers

Construction and matching of floating point numbers is supported. Floating point numbers are constructed and matched according to the IEEE 754 floating point standard. A float is considered to be packed in either 32 or 64 bits. The following type modifiers are applicable to floats.

- `float` The type is float.
- `big` The float is packed in big endian order.
- `small` The float is packed in small endian order.

For example the following code produces a 8-byte binary

```
F = 3.14,  
Bin = <F:64/float-little>,
```

and if we want to match such a binary we can extract the packed float as in:

```
<F2:64/float-little> = Bin.
```

Note that the two floats `F1` and `F2` are not necessarily equal since the conversion process may change the floating point value with a small fraction².

4.4 Tuples

If a sequence of equally sized and typed values need to be processed we can either pack the values from or to a tuple.

For example if we have an 8-byte binary `Bin` and we want to pull the binary apart in four parts each part consisting of 2 bytes, where each sequence of two bytes is considered to be a 16 bit integer we can unpack `Bin` as:

```
<Tup:4/tuple-integer:16> = Bin,
```

Where `Tup` will be bound to an arity four tuple.

The size indicator 4 indicates how large we want the tuple to be. The new type modifier `integer:16` indicates how many bits shall be used for each element in the newly constructed tuple.

This syntax provides an efficient way of processing several values in one sweep. Whenever the `tuple` type modifier is present in the type modifier list, the remaining type modifiers apply to each individual element that is processed and packed in the tuple.

Similarly if we have a tuple of equally typed values and want to pack the sequence of elements in a binary we can do that with the `tuple` type modifier.

Example:

²It is generally considered to be poor programming practice to compare floating point numbers for exact equality

```
Tup = {1.23, 5.66, 9.00, 6.87654},  
<Tup:4/tuple-float:64-little>
```

Will take the values in the tuple `Tup` from left to right and pack them in a binary object.

4.5 Binaries

It is possible to extract sub binaries from an input binary by explicitly providing a type modifier `binary`. The size field in this case applies to number of bytes instead of bits as is the case with i.e integers. Example:

```
<_:10/binary, X:10/binary |_> = Bin,
```

will strip off the first 10 bytes from the binary `Bin` and bind the variable `X` to the next 10 bytes in the `Bin` buffer.

It is also possible to concatenate binaries with the same syntax. For example if we have a binary object `B` and want to build a new binary object `B2` with the size of `B` prepended we can evaluate:

```
Size = size(B),  
<Size:32/unsigned-integer, B:Size/binary>.
```

or using a tail expression as described in the section about the binary tail:

```
Size = size(B),  
<Size:32/unsigned-integer | B>
```

A function that concatenates a list of binaries can be written as:

```
lists:foldl(fun(B, Ack) ->  
            <B/binary | Ack>  
            end, <>, ListOfBinaries)
```

If the size field is omitted in an expression with the `binary` type modifier, the default value is the size of the entire binary. Thus the expression `<B/binary?>` is equivalent with the more awkward `<B:(size(B))/binary>`.

4.6 Lists

Exactly the same technique and syntax we used in the previous section with tuples can be used with lists. For example to take the first 100 characters of a binary we can call:

```
<Str:100/list-char:8> = Bin,
```

The resulting variable `Str` will be bound to a list of 100 characters.

The ability to combine binaries and char-list is often useful.

4.7 The Tail

Whenever a tail is provided in a binary pattern, either in a matching left hand expression or in a right hand construction expression, the tail value is considered to be a binary object. For example in the match expression:

```
<X:8/char, Z:32/integer | Tail> = Bin,
```

The Tail value will be the remainder of the input binary Bin when first 5 bytes have been consumed by the X and Z variables. Thus the above expression is equivalent to

```
Sz = size(Bin),  
<X:8/char, Z:32/integer, B2:Sz/binary> = Bin,
```

The same principle applies when we construct binaries, if a tail is given in a righthandside construct expression, it is considered to be a binary tail.

```
<X:8/char, Y:8/char | Tail>
```

Will prepend the two characters X and Y on the head of Tail.

4.8 System endianism

Sometimes it can be convenient to be able to produce an array of integers formatted in the endianism of the underlying machine. We have the following bit types predefined:

- `sys_int` The endianism and size of a system int
- `sys_short`
- `sys_long`
- `sys_char`
- `sys_float`
- `sys_souble`

If we use these types to produce a binary object as in:

```
B = <77/sys_int, 88/sys_int>,
```

and then send the resulting binary to an Erlang port as in:

```
Port ! {self(), {command, B}},
```

The c program at the other end of the port can read the data as:

```
int a[2];
read(fd, a, sizeof(a));
```

The above code work on both little endian and big endian machines. Of course the c program can also use this feature to directly read binary data into a structure, thus :

```
B = <"abcd", X/sys_int, F/sys_float>,
Port ! {self(), {command, B}},
```

can be read from the port program as:

```
struct foo {
    char c[4];
    int x;
    float f;
};

struct foo f;
read(fd, &f, sizeof( struct foo));
```

4.9 Bit defaults

Two different compiler directives are applicable when we write code that manage binaries.

The compiler directive

```
-bitdefault(integer, unsigned-little)
```

will add the three modifiers `unsigned`, `integer` and `little` to each list of type modifiers in the remainder of the source code file where adding these type modifiers is applicable.

This can be useful in a number of different settings. Say that we are writing a module that only deals with unsigned 32 bit integers, in that case it can be awkward to specify the full modifier list throughout the entire file. Thus, the lazy programmer can save keystrokes by specifying

```
-bitdefault(integer, size:32-unsigned-little)
```

Here we added a size modifier saying that the default integer uses 32 bits. Similarly we can indicate the default type modifiers for floats as in:

```
-bitdefault(float, size:32-big).
```

where we say that the default floating point value is a 32 bit float in big endian byte order.

Another situation is when we want to write code that is independent on endianism of the integers. In that case we can have a single line at the top of the file that specifies endianism.

`-bitdefault(little)`.

If no bit default is given as a compiler directive and no type modifier is specified in the code, a bit pattern still has a meaning since almost everything has a default value. The same rules apply regardless of whether the pattern appear in a match or as a constructor. We have the following default values:

- `<V>` is equivalent with `<V:8/char>`
- `<V/integer>` is equivalent with `\verb|V:32/integer-signed-big|+`
- `<V/float>` is equivalent with `\verb|V:32/float-big|+`
- `<V/binary>` is equivalent with `\verb|Sz = size(V), |V:32/binary|+`
- `<V:Size>` is equivalent with `<V:Size/signed-integer-big>`

4.10 Bit typedefs

We can specify a name for a particular sequence of type modifiers as in:

`-bittype(uint, integer-unsigned)`.

This allows us to use the name `uint` as type modifier as in

`<X:32/uint>`.

or if we want to indicate the size we define:

`-bittype(uint, integer:32-unsigned)`.

4.11 Alignment

A sequence of match or construct values must always add up to a multiple of 8. The alignment requirements for binaries are very low and basically the only requirement is that the resulting binary in a build expression is byte aligned.

It is for example possible to produce a binary with a floating point inserted at non byte aligned positions in the output binary as in:

`<X:1/uint, Pi:32/float, Y:7/uint>`.

However

`<X:9/uint>`.

Will produce a runtime error.

4.12 The empty binary

The empty binary which has size 0 is specified as <>.

4.13 Bit Groups

.....

4.14 Variable field size

A very common situation in many protocols is to let the Protocol Data Unit (PDU) contain a size specifier which is indicating the size of some later field in the PDU. As a very simple example the Erlang Port mechanism typically prepend a two byte length indicator on each message it sends to an external process that is connected to the port. When we receive such a message we can both have the cake and eat it since we are able to first bind the initial two bytes to an integer and then use that integer in the same match expression. So to receive such a message we would write code such as:

```
receive
  {data, <Size:16/uint, Data:Sz/binary>} ->
    handle_data(Data),
  .....
```

For a more complex but truly useful example, assume that we have a port and receive data from the port where each packet from the port is prepended with a two byte header indicating the size of the entire packet, then the following code receives messages from the Port and sends complete packets to the user of the port.

```
-bitdefault(unsigned-integer).
```

```
port_loop(Port, User, Ack) ->
  case Ack of
    <H:16/unsigned, Pdu:H/binary | Tail>
      User ! {pdu, Pdu},
      port_loop(Port, User, Tail);
    Other ->
      receive
        {Port, {data, Data}} ->
          port_loop(Port, User, <Ack/binary | Data>)
      end
  end
end.
```

This code solves a fairly complicated problem. Assume we have a linked in driver which reads data from some input channel, in a UNIX environment, this would typically be a file descriptor. If the above code sits at the Erlang end of the port, the driver can freely read as much as it can from the file descriptor and whatever the driver reads, it simply passes on to the port. The alternative to this is to either let the driver perform two read operations, one where the first two bytes, being the length indicator are read, and then yet another read operation where the actual contents are read, or alternatively, the driver can blindly try to read as much data as it can. In this case, the driver must be prepared for all possible outcomes of the read operation. It can receive a half header, a header and a half message, a header a message and yet another half header and so on. If the PDU structure is more complicated than the one above, the c-code to perform all these operation correctly can be fairly complicated.

5 Are they strings ?

Yes, binaries can be used as builtin character strings.

Syntactic sugaring makes the expression:

```
<"funky">
```

equivalent to the more awkward expression:

```
<"funky":5/list-char:8>
```

When we use binaries for strings it is very convenient to be able to search a binary for a specific substring and split it into two different parts with the first part being the part upto the search string and the remainder the rest.

A binary match pattern may contain unbound variables with the size field being unbound too. This feature can be used to search a binary for a specific string. For example the match pattern

```
<X:Sz/binary, "cat" | Tail> = Bin,
```

Will search the binary `Bin` for the first occurrence of the string `"cat"` and divide `Bin` in two parts, the part leading upto the string `"cat"`, and the part following `"cat"`. If no `"cat"` is found in `Bin`, the pattern does not match.

This feature in combination with the rest of the binary system can be used to implement efficient string manipulation functions.

Many of the string manipulation functions that have complexity $O(n)$ when strings are implemented as lists of cons cells have complexity $O(1)$ when strings are implemented as binary objects in Erlang. For example the length of a string:

```
len(S) -> size(S).
```

As well as concatenation:

```
concat(S1, S2) ->
  <S1/binary | S2>.
```

have complexity $O(1)$, whereas the `chr/2` function that returns the position of the first occurrence of a char in a string:

```
chr(S, C) ->
  case S of
    <_:Size/binary, C:8/char |_> ->
      Size + 1;
    _ ->
      0
  end.
```

of course have complexity $O(1)$.

The module `bstring` implements a number of string utility functions that work on binary strings. The module has exactly the same interface functions as the original `string` module which operates on lists of integers/characters.

Since binary strings have completely different internal representation than list based strings, they behave differently concerning memory consumption and speed. Many programs do behave much better with binary strings than list based strings. In particular programs that manipulate large amounts of data. For example a function that opens a file, partition the file into lines and returns a list of {LineNumber, Line} tuples:

```
file_lines(File) ->
  case file:open(File, [read,raw,binary]) of
    {ok, Fd} ->
      Res = file_lines(Fd, <>, 1, []),
      file:close(Fd),
      Res;
    Err ->
      Err
  end.
```

```
file_lines(Fd, Old, Lno, Ack) ->
  case file:read(Fd, 2000) of
    {ok, B} ->
      {A2, T2, L2} = split_chunk(<Old/binary | B>, Ack, Lno),
      file_lines(Fd, T2, L2, A2);
    eof ->
      Ack;
    Err ->
      Err
  end.
```

```

split_chunk(<Line/binary, $\n/char | Tail>, Ack, Lno) ->
    split_chunk(Tail, [{Lno, Line} |Ack], Lno+1);
split_chunk(Other, Ack, Lno) ->
    {Ack, Other, Lno}.

```

This code is very efficient and works well even on very large input sets, unfortunately there are a number of different operation commonly performed on strings that are considerably less efficient. For example reversing a binary string.

```

breverse(S) ->
    breverse(S, <>).
breverse(<H/char|Tail>, Ack) ->
    breverse(Tail, <H/Char|Ack>);
breverse(<>, Ack) ->
    Ack.

```

The cons cell is a superior data structure in this case.

6 Classical internet style text based protocols

Many typical internet protocols are text and newline based. That means that the participating computers send readable newline terminated text strings to each other. Examples of such protocols are SMTP for electronic mail, NNTP for news postings and HTTP for the world wide web.

The binary strings of ERLANG are well suited for the implementation of such protocols. In particular the ability to search for substrings using the construct with a variable length binary followed by a bound expression, for example:

```
<B:Sz/binary, $:/char |_> = InputString
```

extracts the intial chars of the InputString upto and excluding the first occurrence of a : character.

For example a typical http request to a http server looks like

```

GET /index.html HTTP/1.0
User-Agent: Mozilla/4.04 [en] (X11; I; SunOS 5.5.1 sun4u)
Pragma: no-cache
Host: gin.du.etx.ericsson.se:5999
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Via: 1.0 proxy.du.etx.ericsson.se:82 (Squid/1.1.20)

```

```
X-Forwarded-For: unknown
Cache-control: Max-age=2592000
```

Each line is terminated with a pair of carriage return / newline except the last line which has two such pairs. An efficient parser to unpack such MIME like headers may look like:

```
parse(<"GET ", What/binary, "\r\n" | Tail> ->
  {Fields, Contents} = parse_tail(Tail, empty, []).
  {What, Fields, Contents}.

parse_tail(<B/binary, "\r\n\r\n" | Cont>, A) ->
  {Cont, Ack};
parse_tail(<B/binary, "\r\n" | Tail>, A) ->
  parse_tail(Tail, [B|A]).
```

This code uses substrings extensively. Http is more complicated than this with POST requests etc, but the above code can be augmented in a straightforward way to handle the entire http protocol.

The above code can form the heart of a microscopic http server.

As another example we could have the code to extract the "From:" field from SMTP emails. This could be done as:

```
from_header(<_/binary, "\nFrom:", Sender:Sz/binary, "\r\n" |_) ->
  Sender.
```

6.1 Case insensitivity

Many of the text based internet protocols are case insensitive. To exemplify, the SMTP mail protocol always start a session with a line of text:

```
HELO <domain> CRNL
```

where the client tells the SMTP server which his fully qualified host name is. The initial "HELO" string is case insensitive, so not only the string "HELO" is accepted, but also the string "HeLo". This can be addressed with the binary type modifier `icase`. So to match the HELO message in an SMTP server we would write:

```
case Message of
  <"HELO"/-icase, Domain/binary, "\r\n" |_)
    handle_domain(Domain);
```

6.2 An SMTP client

In this section we shall present a small SMTP client that constructs an email, talks the SMTP protocol with an SMTP server and sends away the email. This is not a complete implementation of SMTP, albeit a fully working implementation: We start off with some definitions and support functions:

```
-define(READY, "220").
-define(CLOSE, "221").
-define(OKAY, "250").
-define(START, "354").
-define(NOAVAIL, "550").
-define(CRNL, "\r\n").

-record(rec, {from,      %% The from address, who am i
             hname,     %% our hostname
             smtp}).    %% name of the smtp server
```

And a function to initialize the record:

```
getrec() ->
  #rec{from = "<" ++ user() ++ "@" ++ hname() ++ ">",
       hname = hname(),
       smtp = smtp()}.

user() ->
  case os:getenv("USER") of
  false -> "luzer";
  User -> User
  end.

hname() ->
  case inet:gethostname() of
  {ok, H} ->
    case inet:gethostbyname(H) of
    {ok, HE} ->
      HE#hostent.h_name;
    {error, _} ->
      "localhost"
    end;
  {error, _} ->
    "localhost"
```

```

end.

smtp() ->
  case os:getenv("SMTP") of
    false -> "localhost";
    H -> H
  end.

```

Then we have the two top level functions that are the API:

```

send(M, To) ->
  send(M, To, "No subject").
send(M, To, Subject) when binary(M), list(To), list(Subject) ->
  R = getrec(),
  To2 = map(fun([$<|T]) -> [$<|T];
            (X) -> "<" ++ X ++ ">")
        end, To),
  M2 = fix_mess(M, To2, R, Subject),
  case gen_tcp:connect(R#rec.smtp, 25, [binary, {packet, 0}]) of
    {ok, S} ->
      init(S, M2, To2, R);
    Other ->
      Other
  end;
send(,,,) ->
  {error, "Bad input type(s)}.

```

The message is a binary object and it is massaged by the `fix_mess/4` function:

```

%% We need to add the headers to the message body
fix_mess(M, To, R, Subj) ->
  S = <"Subject: ", Subj/char-list, ?CRNL>,
  F = <"From: ", (R#rec.from)/char-list, ?CRNL>,
  To2 = to(To, <>),
  T = <"To: ", To2/binary, ?CRNL>,
  <F/binary, T/binary, S/binary, ?CRNL, M/binary, ?CRNL, ".", ?CRNL>.

to([Last], Ack) ->

```

```

    <Ack/binary , Last/char-list>;
to([T|More], Ack) ->
    to(More, <T/char-list , "," | Ack>).

```

If the message is correctly massaged (according to the rules in SMTP, we try to connect to the SMTP server and enter the init state.

```

init(S, M, To, R) ->
    receive
        {tcp, S, <?READY | _>} ->
            ready(S, M, To, R)
    after 5000 ->
        {error, timeout}
    end.

```

If we get a READY message from the SMTP server we enter the ready state and wait for an OKAY.

```

ready(S, M, To, R) ->
    gen_tcp:send(S, <"HELO ", (R#rec.hname)/char-list, " \r\n">),
    wait_okay(S, M, To, R).

```

```

wait_okay(S, M, To, R) ->
    receive
        {tcp, S, <?OKAY | _>} ->
            gen_tcp:send(S, <"MAIL FROM: ", (R#rec.from)/char-list, ?CRNL>),
            receive
                {tcp, S, <?OKAY | _>} ->
                    send_messages(S, M, To, R, [], []);
                {tcp, S, <Err:3/char-list | _>} ->
                    err(S, Err)
            after 5000 ->
                err(S, timeout)
            end;
        {tcp, S, Other} ->
            <Err:3/char-list | _> = Other,
            err(S, Err)
    after 5000 ->
        {error, timeout}
    end.

```

If all is well so far, we start to transmit message to the list of recipients:


```

send_messages(S, M, [], R, Ok, Nok) ->
  gen_tcp:send(S, <"QUIT \r\n">),
  receive
    {tcp, S, <?CLOSE |_>} ->
      gen_tcp:close(S),
      {ok, Ok, Nok};
    {tcp, S, <Err:3/char-list |_>} ->
      err(S, Err)
  after 5000 ->
    err(S, timeout)
end;

send_messages(S, M, [To|More], R, Ok, Nok) ->
  gen_tcp:send(S, <"RCPT TO:", To/char-list, ?CRNL>),
  receive
    {tcp, S, <?NOAVAIL |_>} ->
      send_messages(S, M, More, R, Ok, [To | Nok]);
    {tcp, S, <?OKAY|_>} ->
      gen_tcp:send(S, <"DATA ", ?CRNL>),
      receive
        {tcp, S, <?START |_>} ->
          gen_tcp:send(S, M),
          receive
            {tcp, S, <?OKAY|_>} ->
              send_messages(S, M, More, R, [To|Ok], Nok);
            {tcp, S, <Err:3/char-list|_>} ->
              send_messages(S, M, More, R, Ok, [To|Nok])
          after 10000 ->
            err(S, timeout)
          end;
        {tcp, S, <Err:3/char-list|_>} ->
          send_messages(S, M, More, R, [To|Ok], Nok)
      after 10000 ->
        err(S, timeout)
      end;
    {tcp, S, <Err:3/char-list|_>} ->
      send_messages(S, M, More, R, Ok, [To | Nok])
  after 10000 ->
    err(S, timeout)
end.

err(S, Reason) ->
  gen_tcp:close(S),

```

```
{error, Reason}.
```

And that is all there is. In order to mail a mail, we merely call

```
smail:send(Mess, ["klacke@erix.ericsson.se",  
                 "tony@erix.ericsson.se"],  
           "Is this cool or what " ),
```

7 UDP/IP

In this section we provide a microscopic implementation of the protocol suite UDP/IP. This is used to exemplify a number of useful techniques. The reader who is not familiar with the internals of the IP protocol suite is well advised to read an introductory text on the topic.

The only magic we assume here is the ability receive ethernet frames into the ERLANG application. This can be achieved in a number of ways, however they differ wildly on different operating systems. No name resolution is performed at all, all addresses are supposed to be proper IP addresses. This is of course a ridiculous application since there already exists a large number of highly optimized implementations of the TCP/IP protocols. It is however interesting from an educational point of view since it shows how easily protocols can be developed in ERLANG.

We have a receive loop at the bottom most layer that receives Ethernet frames, decodes them and dispatches them to either of the ARP [?] or IP [?] layers. This receive loop is aware of the local ethernet address by means of the previously mentioned magic.

8 IDLs

Many telecommunications protocols make use of ASN.1[?] to define the datatypes that are used in a protocol. Different encodings such as BER[?] and PER[?] are typically used. In this section we show how what code an ASN.1 to ERLANG compiler should generate utilizing the bit syntax. In particular we show how a binary string produced by the application, inserted into an ASN.1 data structure can find its way all the way down to the IO system without ever being copied.

Lets assume we have an ASN.1 datatype Struct:

```
Struct ::= SEQUENCE {  
    b BOOLEAN,  
    s OCTET STRING }
```

The equivalent ERLANG record would of course be:

```
-record('Struct', {b, s}).
```

In this example we shall assume that the Basic Encoding Rules, BER is used. For a brief introduction to ASN.1 and BER [?] can be consulted.

BER is an encoding scheme which uses a Tag/Length/Value encoding. This means that every value is encoded as three parts. The tag, identifies which type is encoded, the length is the length of the encoded value, and the value is of course the coded value itself. This is often referred to TLV encodings.

In this case we have three ASN.1 components to encode, a SEQUENCE, a BOOLEAN and an OCTET STRING.

First we need a function to encode a boolean:

```
ebool(true) ->
    <?BOOL/8, 1:8, 1:8>;
ebool(false) ->
    <?BOOL/8, 1:8, 0:8>.
```

Here we made use a constant `BOOL` which is the tag that defines a boolean in BER code:

```
-define(BOOL,      2).
-define(OSTRING,  16).
-define(SEQUENCE, 48).
```

Furthermore we need to have a library function to encode octet strings. We have:

```
eostring(Bstr) ->
    Sz = size(Bstr),
    Bits = lensize(Sz)
    <?OSTRING:8/integer, Sz:(Bits*8)/integer-little, Bstr/binary>.
```

We need a library function `lensize/1` to calculate the number of bytes necessary to hold the BER length value. We have:

```
lensize(S) ->
    lensize(S bsr 7, 1).
lensize(0, I) -> I;
lensize(X, I) ->
    lensize(X bsr 8, I+1).
```

The function `lensize/1` is defined according to the rules for BER length encodings.

Now that we have the primitive library functions that we need we can look at the code that is generated by a proper ASN.1 to Erlang compiler to encode our `Struct` structure:

```

enc_Struct(S) ->
    B1 = ebool(S#'Struct'.b),
    B2 = estring(S#'Struct'.s),
    Sz = size(B1) + size(B2),
    Bits = lensize(Sz) * 8,
    <?SEQUENCE:8/integer, Sz:Bits/integer,
    B1/binary, B2/binary>.

```

This code is particular interesting since it shows how a string produced at the application layer, can find its way through a BER encoder without ever being copied. The result of the `enc_Struct/1` operation may in its turn be manipulated by lower layers in the protocol stack, still without ever copying the initial string provided by the user. This can be achieved since the memory management system for the user is the same as the memory management system for the protocol layers. So here we see how the binary object implementation acts as a general purpose buffer manger for protocol implementations.

This section made a point using ASN.1 as IDL, however exactly the same reasoning applies to other IDLs, such as Corba and XDR, albeit in the case of Corba, more complicated.

9 Internals

In this chapter we describe how binaries are represented internally in the runtime system. We believe that this information is important since one of the main objectives with this experimental syntax is to achieve ERLANG programs that are not only more beautiful but also faster and less memory consuming. Thus it is important for protocol implementors to be at least vaguely familiar with the internals of binaries, and thus with the characteristics of binaries.

A binary object can be represented in four different ways inside the runtime system depending on how it was created. All binaries consist of a tagged pointer to a structure on the ERLANG process heap. These structure, called thing structures come in four different varieties. The user of binaries, i.e. the ERLANG programmer doesn't see anything about what sort of binary a specific object is represente as internally.

At the language level all binaries are seen as consecutive series of octets.

9.1 Heap binaries

First we have the simplest variant which can be described by the c-code struct:

```

typedef struct heap_binary {
    uint32 thing_word; /* tagged thing, with subtag and tari */
    int size; /* number of bytes */
    uint32 data[1]; /* The data */

```

```
} HeapBin;
```

Where the `uint32` type is an unsigned 32 bit value. The `thing_word` field is used to identify the type of the structure, the `size` field is the number of bytes that this binary contains, and finally the `data` field is used to locate the beginning of the actual data.

This type of binary object is used for small binaries, for example the result of `<X:16>`. will be a tagged binary pointer pointing to a `HeapBin` structure on the regular process heap. These type of objects are garbage collected similar to tuples and all other regular ERLANG objects. Furthermore they are copied in message passing as well as when they are inserted into ets-tables.

9.2 Reference counted binaries

Secondly we represent large binary objects as a pointer to a structure:

```
typedef struct refc_bin {
    uint32 thing_word;    /* tagged thing, with subtag and tari */
    int size;            /* nuber of bytes in binary */
    RefcBin *next;      /* heap pointer to next RefcBin */
    Binary *val;        /* Pointer to refc'ed object */
    byte *bytes;       /* actual byte* pointer */
} RefcBin;
```

The principal goal of this data structure is to be able to send large binaries to other ERLANG processes without copying the actual data. Thus the `Binary*` field points to a structure:

```
typedef struct binary {
    int orig_size;      /* total length of binary */
    int refc;          /* number of references to this binary */
    char orig_bytes[1]; /* the data (char instead of byte!) */
} Binary;
```

The `Binary` structure contains the actual data and it is allocated off heap, typically by means of `malloc()`. It contains a reference counter, so whenever a `RefcBinary` object is sent in a message to another process the following things occur. First the reference counter is incremented, then a `RefcBin` structure is created on the receiving process heap. This structure is linked into a list chained list of `RefcBin` objects on that heap. This last step is important. All `RefcBins` created on a process heap are chained in a list visible to the process. When the process heap is garbage collected, the chain is traversed and any object in the chain which has not been moved by the garbage collector is first unlinked

from the chain, and then followed in order to decrement the reference counter. When the reference counter is zero, the entire Binary structure can be released, typically by means of a call to `free()`.

The ability to send large binary objects without copying them is especially important in implementations of protocols that carry data as opposed to pure signalling protocols.

9.3 Segmented binaries

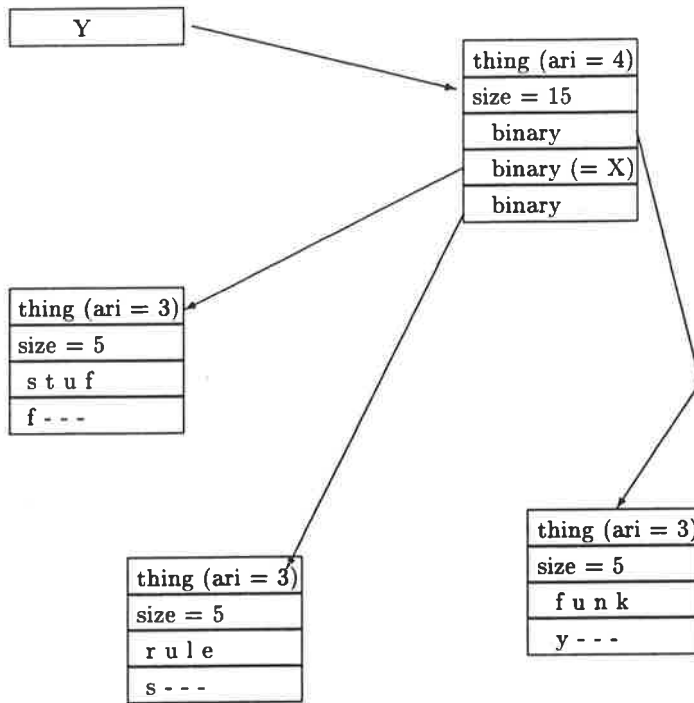
When constructing binaries by adding components as head or a trailer onto a PDU it is important to be able to do that in constant time. For example:

```
X = <" stuff ">,
Y = <"funky ", X/binary, " rules">.
```

Therefore a binary object can consist of an array of other binary objects, we call this a segmented binary. A segmented binary is represented on the ERLANG heap as a structure:

```
typedef struct segm_binary {
    uint32 thing_word; /* tagged thing, with subtag and tag */
    int size;          /* number of bytes in the total binary */
    uint32 data[1];
} SegmBin;
```

Both the HeapBin structure and the SegmBin structure are variable sized. The number of words that any of these structures consist of can be calculated by extracting the 16 least significant bits of the `thing_word`. In the case above we would have the following pointers and data structures:



In the case with the "funky stuff rules" string, all data is stored on the regular ERLANG heap and thus subject to the standard copying garbage collection. When a structure like this is copied, the structure is compactified and the result will be that the variable Y from above is bound to a single HeapBinary instead of a segmented binary.

9.4 Sub Binaries

Code that strips off a head from a binary and performs a calculation on the head, and then basen on the outcome of that calculation continues to process the Sub is typical for man applications. The file_lines/1 from a previous section is an example of this. Therefore the runtime system represents such a binary in an optimized way. If we have:

```
<X:16 | Tail> = Bin.
```

the tail will be represented as a pointer to a TailBin structure on the heap:

```
typedef struct Sub_binary {
    uint32 thing_word;
    int size;
    int offs;
    uint32 orig; /*original binary object to use for offset */
} SubBin;
```

This means that the tail is only represented as an offset into the original object. This is used for another interesting optimization in the execution machinery. When a match operation is initiated on a binary object, a match_buffer is initialized in the runtime system. If the match succeeds, a word in the match buffer is set to the value of the initial input to the match machinery. On the next call to the match machinery, this word is checked to see if it is the same word we set the last time we performed a match, if this is the case no initialization of the match buffer is necessary at all. For example:

```
funky() ->
    X = <"abcd">,
    count_16(X).

count_16(<X:16|Tail>) ->
    1 + count_16(Tail);
count_16(<>) ->
    0.
```

In the above code, the match buffer will only be initialized once. This is on the first call to count_16/1. On the second call to count_16/1 the match buffer will be both initialized as well as initialized to the correct value with regards to bit offset into the original binary object X.

The same data structure can also be used when a substring is extracted from an initial binary. For example:

```
sub(Str, From, To) ->
  Sz = To - From,
  case Str of
    <_:From/binary, Result:Sz/binary |_>
      Result
  end.
```

The result value `Result` will be represented as a `SubBinary` to the original input binary `Str`.

9.5 Binaries and IO

Most modern IO devices today support a mode of IO called gather/scatter IO. The UNIX system call `writenv()` takes an output buffer which consists of an array of `iovec` structures. An `iovec` structure contains the following members:

```
caddr_t   iov_base;
int       iov_len;
```

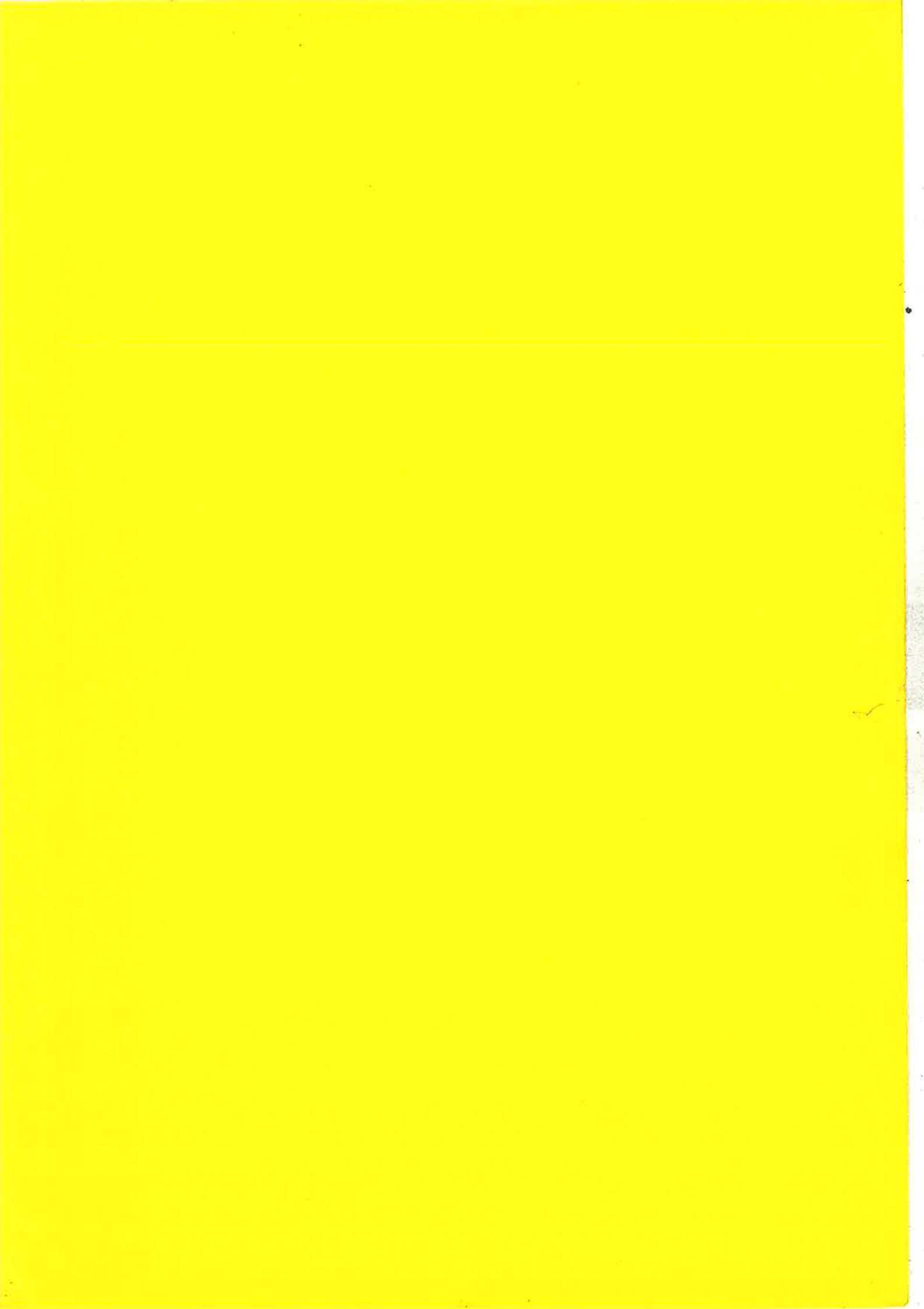
This maps perfectly with a segmented binary. The net result of this is that the ERLANG programmer can construct an output buffer as a number of small operations each operation either adds some data to the head or the tail of the buffer. Once the buffer is ready for output, the runtime system constructs an array of `iovec` structures by setting pointers, i.e. without copying any data. This array is then passed directly to the `writenv()` routine.

All IO in ERLANG is performed through a driver. In order for the `writenv()` scenario to take place, the driver must export a `writenv()` routine. How this is done is described in an appendix to this document.

Appendix

Scatter IO in a linked in driver

The scatter IO interface in linked drivers in the ERLANG system ought to be described in some proper OTP documentation. Since it is not and the scatter IO mechanism is of paramount importance for the implementation of efficient protocol drivers in ERLANG the mechanism is described here.



1870

1871

1872

1873

1874

1875

1876

1877

1878

1879

1880

1881

1882

1883

1884

1885

1886

1887

1888

1889

1890

1891

1892

1893

1894

1895

1896

1897

1898

1899

1900